Table of Contents

Copyright and Version

Take Command/16 for Microsoft Windows 3.x

Version 2.01 Help System

Text by Hardin Brothers, Tom Rawson, and Rex Conn

[2/98 - 2.01A]

Using the Take Command Interface

Using a Windows Command Line

Take Command is a new environment that lets you perform tasks easily under Windows. You can use it to execute commands, start applications, and perform other work at the command line.

In the past you may have accomplished some of these tasks by starting a Windows session to run 4DOS, JP Software's replacement command processor for DOS. Or you may have used an "MS-DOS Prompt" session to run the default DOS command processor (*COMMAND.COM*) under Windows.

In either case and especially if you are an experienced user of 4DOS you'll find plenty of familiar features in Take Command. You'll also find a lot that's new and different.

While Take Command includes most of the command-line, batch file, and other capabilities provided by 4DOS, and goes well beyond those provided by *COMMAND.COM*, the Windows environment places some limitations on how Take Command operates.

These limitations mostly affect the use of external programs, especially DOS programs. This topic is covered in detail under Take Command and DOS Applications. You can use Take Command without going over these details; however you should read through them before changing Take Command's default options for starting DOS programs (for example, those in the **VM Setup** dialog on the Options menu).

There are some other minor differences between using Take Command and using a 4DOS (or *COMMAND.COM*) session under Windows (for example, some keystrokes are interpreted differently to conform more closely to Windows conventions). There are also some considerations when running batch files or 4DOS aliases designed to work under DOS in a Windows program like Take Command. All of these differences are covered in more detail under Take Command for 4DOS Users.

Take Command also offers a wide range of new Windows-related features which are not available in 4DOS or *COMMAND.COM* sessions, including:

- A built-in scrollback buffer that lets you look back through the output from past commands.
- A standard Windows menu bar for access to many commonly-used Take Command features.
- A status bar showing memory and resource usage.
- A customizable tool bar that gives you quick access to commands and applications.
- Windows dialogs (accessible from the Options and Utilities menus) for editing environment variables, aliases, file descriptions, and startup parameters.
- Direct access to Program Manager groups through the Apps menu.
- High-speed, dialog-based file and text search (see "Find Files/Text" on the Utilities menu). The new FFIND command gives you the same capabilities at the Take Command prompt.
- Commands like ACTIVATE, MSGBOX, and QUERYBOX that allow you to use Windows features and control Windows applications from your batch files.
- A new technology, called "Caveman," which you can use to run many DOS utilities in the Take Command window (see Take Command and DOS Applications for details).

Starting Windows Applications

Take Command offers several ways to start Windows applications.

First, you can simply type the name of any Windows application at the Take Command prompt. As long as the application's executable file is in one of the standard search directories (see below), Take Command will find it and start it. If you type the full path name of the executable file at the prompt the application will be started even if it is not in one of the standard search directories.

Take Command offers two methods to simplify and speed up access to your Windows applications. One is to create an alias, for example:

```
c:\> alias myapp d:\apps\myapp.exe
```

You can also use the Tool Bar to start frequently-used applications. For example, a tool bar button named **MyApp** which invokes the command **d:\apps\myapp.exe** would accomplish the same thing as the alias shown above.

You can use these methods together. For example, if you define the alias shown above you can set up a tool bar button called **MyApp** and simply use the command **myapp**, which would then invoke the previously-defined alias.

You can start any application defined in a Program Manager group using the corresponding selection on the Apps Menu.

You can also start an application by typing the name of a data file associated with the application. Take Command will examine the file's extension and run the appropriate application, based on Windows file associations or Take Command executable extensions.

For additional flexibility, you can also start applications with the START command. START provides a number of switches to customize the way an application is started.

Using Internet URLs

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

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

Searching for Applications

When you start an application without specifying a path, Take Command searches for the application in the current directory, the Windows directory, the Windows system directory, and then all directories on the PATH. (If you do enter an explicit path, Take Command will only look in the directory you specified.)

If you enter a file name with no extension, Take Command will search each directory for a matching *.COM*, *.EXE*, *.BTM*, or *.BAT* file, then for a file matching a Windows file association or Take Command executable extension. If no such file is found Take Command will move on to the next directory in the search sequence.

This is the same search order used by Windows components like Program Manager and Explorer, and is similar to the search order used in DOS, with the addition of the Windows and Windows system

directories.

Waiting for Applications to Finish

When you start an application from the prompt, Take Command does not wait for the application to finish before displaying the next prompt. This allows you to continue your work in Take Command while the application is running.

To change this default behavior, and force Take Command to wait for applications started from the prompt, use the ExecWait option in the *TCMD.INI* file, or the corresponding **External Programs** setting on the Options 2 page of the configuration dialogs.

Changing the default behavior does not applications run from batch files, or those run under Caveman. Take Command always waits for such applications, regardless of the option setting.

Due to the way Windows handles URLs, you cannot wait for the browser software to finish when you enter an **http:** URL at the prompt.

You can also control whether Take Command waits for an application to finish by launching the application with the START command. If you use START's **/CM** or **/WAIT** switch, Take Command will always wait for the application; otherwise, it will not wait, even if the application is run from a batch file. START can also control many other aspects of how your applications are started.

Passing the Environment to Applications

Take Command normally does **not** pass its environment to Windows applications. For example, if you have a Windows program which uses the TEMP environment variable, and you modify this variable within Take Command, the application will not be affected. Instead, it will inherit the value of TEMP that was set in DOS, before Windows started.

You can force Take Command to pass the environment to Windows applications by using **START /E** to start the application program. However, due to a long-standing bug in Windows, your application may not start properly using this method (which is why it is not the default). Typically the bug causes Take Command to return to the prompt quickly, and the application never starts. In most cases you can work around the bug by varying the size of the environment by a few bytes, for example by adding one or two short "dummy" variables with the SET command:

```
c:\> set dum1=aaa
c:\> set dum2=bbb
```

Some experimentation may be required to determine what works for a particular application, and the requirements may change depending on your system configuration, the applications you have run recently, or Windows' internal state.

Take Command and DOS Applications

This topic and those following it explain in detail how Take Command works with your DOS applications. You can use Take Command without going over these details; however you should read through them before changing Take Command's default options for starting DOS programs (for example, those in the **VM Setup** dialog on the Options menu).

When you start an external program under Windows it normally runs in its own window, which opens when the program starts and closes when it exits. You can also start a DOS program inside a 4DOS or "MS-DOS Prompt" session, and the program will run within that session.

In its default configuration Take Command conforms to these norms. Whether you start a DOS or Windows program, the program will be assigned its own window, and that window will close when the program exits. When a DOS program is started in this way Take Command will wait for the program to exit before continuing.

However, this approach does not work well for command-line programs which display their output to the screen and then exit. As soon as the program exits, its window closes and the output is lost!

To make it easier to use this type of program from within Windows, Take Command includes a new technology, called "Caveman", which allows DOS programs to run within the Take Command window.

Due to limitations in the way DOS programs can operate under Windows, the techniques used by Caveman do not work well with all programs. The following topics tell you how to set up your system to make the best use of Take Command and Caveman for running DOS applications:

Starting DOS Applications

Running a DOS application under Take Command with Caveman is a quick, easy, and seamless way to run DOS utilities without starting a separate window. However, it works only for simple DOS utilities which perform standard input and output. It generally cannot be used for major DOS applications like word processors, spreadsheets, and databases, and its performance and compatibility will be limited with other applications.

Caveman is normally installed when you install Take Command. It runs only in Windows' 386 Enhanced mode; if you start Windows in Standard mode, you cannot use Caveman (you can determine the mode in which Windows is running from the Take Command status bar; look for the characters "Enh" or "Std" before the available memory figure). For more information on Caveman, including a more detailed description of how Caveman works, see the Caveman topic.

The second way to run a DOS program under Take Command is to start it in a separate DOS window. This is the same method used by Program Manager's File / Run menu option, and the similar options offered by other Windows shells. Any DOS application can be run using this method.

Each method will be appropriate for some DOS applications on your system, but not useful or even possible to use for others. Unfortunately, Take Command cannot determine automatically which method is best for any given application. Therefore, you must select the best default method for the particular mix of DOS programs you run from within Take Command. You can then use *.PIF* files, aliases, or other Take Command or Windows features to force the use of the other method for the specific applications that require it.

If you want to see how these two methods work, first make sure Caveman is installed. Look at the **VM Setup** choice on the **Options** menu. If this choice is "grayed out" then Caveman is not installed or you did not start Windows in 386 Enhanced mode. If Caveman is not installed, install it and restart Windows (see your Introduction and Installation Guide for manual installation instructions).

To try a DOS program with and without Caveman, start it with the **START** command. Use the **/CM** switch to start the program under Caveman, or leave the switch off to start the program in a separate window. For example:

```
c:\> start /cm pkunzip        Uses Caveman
c:\> start pkunzip      Uses a separate window
```

(There are easier ways to start DOS programs directly from the Take Command prompt, but this is the best method to use while you're experimenting.)

When you start a program under Caveman, its output will appear in the Take Command window. When you start the same program in a separate window, its output will appear in that window. In either case, the program will return to the Take Command prompt when it's finished.

There's no point in starting your word processor, spreadsheet, or communications program under Caveman they almost certainly won't work, and if they do they'll be pretty slow. Use the technique described above to experiment with simple DOS programs like PKUNZIP, XCOPY, and similar utilities. See the Caveman topic for more details on the kinds of applications which are likely to work properly under Caveman.

Caveman does its best to detect incompatible applications (for example, those that attempt to manipulate the keyboard hardware, or use unusual video modes) and terminate them gracefully. If a compatibility problem is detected, you'll see a dialog box explaining that the program cannot work properly under Caveman. Click **Cancel** to terminate the program at this point, or **Restart** to restart it in a separate window. Click **Always use Separate Window** to request that the application be "marked" as

incompatible with Caveman, so that it will be run in a separate window in the future (the "mark" is stored in the *TCMD.INI* file; your application itself is not affected).

Once you have worked with Caveman a bit you can select a default method for starting DOS applications. The method you select will be used automatically when you type the name of a DOS program at the command line or in a batch file. You can then use aliases, *.PIF* files, or the START command to start specific applications using a method other than the default. For complete details, see Caveman Default and Separate Window Default.

To select the default method, open the **VM Setup** dialog, accessible from the **Options** menu. Check the "Run DOS Apps in Caveman" checkbox if you want DOS programs to run under Caveman by default. Uncheck the box if you want DOS programs to run in a separate window by default.

Caveman Default

This section explains in detail how to configure your system with Caveman as the default method for starting DOS programs under Take Command. For a general description of how DOS applications run under Take Command see the previous topics, Take Command and DOS Applications and Starting DOS programs.

Under this method for starting DOS programs, Take Command assumes that DOS applications started from the command line or a batch file should be run under Caveman (for example, if Caveman is the default and you enter the FORMAT command, Take Command will run the FORMAT program under Caveman).

You must then create a *.PIF* file for each application you **don't** want to run under Caveman. When you start a DOS application and Take Command finds the corresponding *.PIF* file, it will ignore Caveman and run the program in a separate window.

Use this method if you typically run a small number of major DOS applications from Windows, and most of the other DOS programs you want to start from the Take Command prompt fall into the "simple utilities" category.

Simple utilities are programs like PKZIP, XCOPY, FORMAT, and other programs which display basic "teletype-style" scrolling output, without significant use of popup windows, full-screen displays, or direct access to your system's hardware devices.

To set up this method, open the VM Setup dialog (accessible from the Options menu), check the **Run DOS Apps in Caveman** box, and click on the **Save as Defaults** button.

Next, establish a *.PIF* file for each of your major DOS applications. You can do this with the Windows PIF editor. To start the PIF editor from the Take Command prompt, change to the Windows directory and enter the command PIFEDIT followed by the application name. For example:

```
c:\> cd windows
c:\windows> pifedit wp
```

This will create a *.PIF* file with the same name as the DOS application program. You should enter the appropriate path, filename, working directory, and other parameters from within the PIF editor, then save the file. See your Windows documentation for additional details on PIF files and PIFEDIT.

You may find that you already have some of the *.PIF* files you need, because it is not unusual to use them for major DOS applications even when Take Command and Caveman are not running. If a *.PIF* file already exists and the corresponding application runs properly, you don't need to make any changes to the file it will work as-is with Take Command.

If you try to start a program under Caveman and a separate window is started instead, it's probably because you already have a *.PIF* file defined for that program. You can remove the *.PIF* file if you wish, but in many cases programs that already have a *.PIF* file must be started with that *.PIF* file in order to work properly. Therefore it's probably best to leave existing *.PIF* files alone unless you know why they were created and are confident that you can remove them without causing problems.

Once the *.PIF* files are set up, you can simply type the name of a DOS application at the Take Command prompt. The major applications for which you have defined *.PIF* files will be run in separate windows, and your other DOS utilities will run inside the Take Command window, using Caveman.

If you don't want to use *.PIF* files, you can accomplish the same thing using aliases. Simply define an alias for each major DOS application which runs the program using the START command. For example:

```
alias wp start d:\wp60\wp.exe
```

Like a *.PIF* file, the use of START will force Take Command to ignore Caveman and start the application in a separate window.

The benefit of the "Caveman Default" approach is that you can run DOS utilities right in the Take Command window without any special commands or aliases to set up. All you have to do is create a few *.PIF* files or aliases for your major DOS applications.

The drawback to this approach is that you may try to start an application which won't work well under Caveman, but for which you have neglected to define a *.PIF* file or alias. The fact that the Take Command prompt looks so much like a 4DOS or *COMMAND.COM* "DOS prompt" can make it easy to make this error. This isn't likely to cause too much trouble you can always terminate an application if you make a mistake, and then set up a *.PIF* file or alias for it but you'll need to be aware of the possibility as you use Take Command.

Separate Window Default

This topic explains in detail how to configure your system with Caveman disabled, so that sepaarate windows are used automatically when starting DOS programs under Take Command. For a general description of how DOS applications run under Take Command see the previous topics, Take Command and DOS Applications and Starting DOS programs.

Under this method of starting DOS applications, Take Command assumes that DOS applications should be run in a separate window (for example, if a separate window is the default and you enter the FORMAT command, Take Command will ignore Caveman and run the FORMAT program in its own window). You must then create an alias for each application you **do** want to run under Caveman.

Use this method if you typically run many significant DOS applications from Windows, and only a few other simple utilities.

To set up this method, open the VM Setup dialog (accessible from the Options menu), remove any check mark in the **Run DOS Apps in Caveman** box, and click on the **Save as Defaults** button.

Next, create an alias for each of the DOS utilities you want to run under Caveman. Use the START /CM command to run the program under Caveman. For example:

```
alias pkunzip start /cm c:\util\pkunzip.exe
```

This alias will force Take Command to run the program under Caveman. You must create a separate alias for each utility you want to run under Caveman (unless you want to type the **START /CM** command each time you run the program).

The benefit of the "Separate Window Default" approach is that you can run any DOS application from the command line or a batch file, without thinking about whether it is the type of application that works well with Caveman.

The drawback to this approach is that you must explicitly create an alias for each utility you want to run under Caveman. This can mean creating a large number of aliases, and you will lose the benefits of the simple, seamless approach that Caveman offers if you don't remember to create an alias for a particular utility. Programs for which you forget to create an alias will run in their own window, and may exit before you have a chance to view their output.

Caveman (technical information)

This topic gives a technical description of Caveman, Take Command's facility for running DOS programs inside the Take Command window. For a more general description of how DOS applications run under Take Command see the previous series of topics, beginning with Take Command and DOS Applications.

If your system is configured and working properly based on the information in the previous topics, you can feel free to skip this topic altogether, or come back to it later when you want more detail.

Caveman is so named because it does all its work hidden in a "cave" an invisible DOS session and is not directly visible to you. Caveman creates the hidden DOS session for you, captures screen output from your DOS programs, and displays it in the Take Command window. It also accepts input from the Take Command window and routes it back to the DOS program.

Caveman consists of a Windows virtual device (VxD), stored in the file *CAVEMAN.386* and loaded when Windows starts. In order to use Caveman you must have the following statement in the **[386Enh]** section of the Windows *SYSTEM.INI* file:

```
device=d:\path\caveman.386
```

where **d:\path** refers to the drive and directory where Take Command is stored. This statement is normally added to *SYSTEM.INI* by the Take Command installation program; if necessary, you can add it (or remove it) yourself with Windows SysEdit, Notepad, or any other ASCII file editor (when you edit *SYSTEM.INI*, you must restart Windows for your changes to take effect). The position of this line within *SYSTEM.INI* is not important as long as it is in the proper section.

Caveman can be used in Windows 3.1 and above (including Windows for Workgroups). It will not work with IBM's WIN-OS/2, because WIN-OS/2 does not currently support the use of VxDs. Since OS/2 provides other methods for starting DOS applications, this is not likely to be a problem.

Virtual devices like *CAVEMAN.386* can only be loaded when Windows is running in 386 Enhanced mode. If you start Windows in Standard mode, Caveman will not be loaded (even if you include it in *SYSTEM.INI*).

A DOS application run from the Take Command prompt, either directly or through an alias or batch file, will be started in the invisible DOS session maintained by Caveman **unless** one of the following conditions is met:

- You have a defined a *.PIF* file for the application.
- You use the START command (without the /CM switch) to start the application.
- The application has been "marked" as one which requires a separate DOS session.
- Caveman is not selected as the default method for running DOS programs.
- Caveman is disabled or *CAVEMAN.386* is not loaded.

If any of these conditions are met, the application will be run in a separate DOS window, not under Caveman.

All output from an application run under Caveman will appear in the Take Command window, and all input requested by the program will be entered in that window.

Input and output requests from a DOS program must be passed back and forth between Caveman and Take Command. While this process is relatively fast, programs which "poll" the keyboard very rapidly to

see if input is ready may run more slowly under Caveman than they do outside of Windows. Programs which generate large amounts of output can also run slightly more slowly. In most cases these effects will not be particularly noticeable.

Caveman works best with, and is intended for, "TTY-style" programs which display simple, scrolling output for example, programs like the DOS FORMAT and XCOPY utilities, or the popular PKZIP and PKUNZIP file compression programs. In most cases, this is the only type of program you should run under Caveman other DOS applications should be started with a *.PIF* file, or with an alias which invokes the START command (see Caveman Default for an example), rather than trying to make them work under Caveman.

Some programs which use popup windows and full-screen displays do work under Caveman. For example, some DOS directory change utilities will pop up a window with a list of directories if you enter a partial directory name; otherwise they simply change the directory and exit. Utilities like this, which make limited use of popups or full-screen displays, typically are compatible with Caveman.

More complex "full-screen" DOS programs which write large amounts of data directly to video memory (for example, ASCII editors or file viewers) may work with Caveman, but their performance will be limited, especially if they update the screen frequently or in large blocks.

Programs which display graphics, modify the screen font or perform other unusual video functions, access the keyboard or other hardware directly, or use special or undocumented methods of accessing memory, will not work at all with Caveman.

Most memory-resident (TSR) programs also will not work properly under Caveman. TSRs loaded before starting Windows should not interfere with Caveman, but you should not attempt to load new DOS TSRs from the Take Command prompt or via START /CM.

Caveman does not provide access to the mouse for DOS programs, and will always inform DOS programs that no mouse is installed. If you have a DOS program which requires a mouse it must be run in a separate DOS window.

Caveman does its best to detect incompatible programs. When a program attempts an operation which can't be handled through Caveman, Take Command will display the Caveman Error dialog. If you click **Always use Separate Window** the application will be "marked" as incompatible with Caveman, and will nto be run automatically under Caveman again. When an application is "marked" in this way its full path and name are stored in the **[DOSApps]** section of *TCMD.INI*. If the application is moved to a different drive and directory, the "mark" will be lost and will have to be recreated the next time the application is run.

If Take Command does hang or behave improperly when you start a DOS application using Caveman, you can terminate the DOS application with the **End DOS app in Caveman** option on the Apps menu. In extreme cases you can also close Take Command entirely by double-clicking the box on the upper left corner of the window. To work around a problem with a DOS program, create a *.PIF* file or alias for the application as described under Cavemman Default.

Using Take Command as the Windows Shell

For complete command-line control of Windows you can install Take Command as your Windows shell. When Windows starts it will load Take Command rather than Program Manager or any other shell. You can then start applications and perform any other work you desire from the command line.

To install Take Command as the shell, first copy the *TC16DLL.DLL* file in your Take Command directory to your *\WINDOWS\SYSTEM* directory (otherwise Windows will not be able to find this file at startup).

Next use SysEdit, Notepad, or another ASCII file editor to edit the *SYSTEM.INI* file (in your *\WINDOWS* directory). In the **[Boot]** section of *SYSTEM.INI* find the **SHELL=** line. Add a semicolon at the start of the old line to turn it into a comment (this preserves the old setting if you want to return to it in the future). Then add the following new **SHELL=** line:

```
shell=d:\path\tcmd.exe
```

Substitute the drive and path of *TCMD.EXE* on your system for "d:\path\" in the line above. You can add any Take Command startup options to the shell line. Save *SYSTEM.INI*, close your editor, and then restart Windows for the line to take effect.

When Take Command is loaded as the Windows shell, it begins by reading the "Load=..." and "Run=..." lines in the [Windows] section of your *WIN.INI* file. Take Command starts each application listed on the "Run=" line and starts each application listed in the "Load=" line as an icon. After these items are processed, Take Command then starts any applications in your Program Manager "Startup" group (using the *STARTUP.GRP* and *AUTOSTART.GRP* files).

When you run Take Command as the Windows shell, it assumes that Program Manager or the equivalent has not been loaded. Therefore, Take Command will not try to establish a DDE connection with Program Manager, regardless of the setting of ProgmanDDE. Instead, Take Command will search for .GRP files and read them directly in order to build its Apps and Windows menus. If Take Command is the Windows shell, and you install a new Windows application that tries to create a Program Manager group for its own files, Take Command will start Program Manager in an invisible window and let it create the group.

When Take Command is the Windows shell, it will display a standard Exit Windows dialog box when you close it by issuing an Exit command, by selecting Exit from Take Command's File menu, or by pressing Alt-F4 when Take Command has the focus.

Windows File Associations

Windows includes the ability to associate file extensions with specific applications; this feature is sometimes called "file associations". For example, when you install Microsoft Word, it creates an association between files with a *.DOC* or *.DOT* extension and the Word program file, *WINWORD.EXE*.

Take Command includes a similar feature, called executable extensions, which allows you to set environment variables which associate a file extension with a particular application.

Under Windows 3.x file associations can be stored in two places: the *WIN.INI* file, and the Windows registry. Associations in *WIN.INI* are "direct" they simply list the extension and the application name, associating the extension directly with the application. Those in the registry are "indirect" they associate an extension with a "file type," and separately specify attributes of files of that type, including the command to execute in order to open such a file.

When Take Command starts, it retrieves the direct file associations from Windows, and treats them like Take Command executable extensions. To disable loading of these direct associations see the Startup page of the configuration dialogs, or the LoadAssociations directive in *TCMD.INI*. The indirect associations are accessed through built-in Windows features (see below), and can not be disabled.

When you attempt to execute a program from the command line or a batch file, Take Command first searches its list of executable extensions, including the standard extensions (*.COM*, *.EXE*, *.BTM*, etc.). It then checks Take Command executable extensions, followed by direct file associations inherited from Windows. If all of these tests fail, Take Command passes the command name to Windows to see if Windows can find an indirect association for it. Each of these tests is done in all of the standard search directories described under the PATH command.

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

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

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

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

```
c:\> unset .wav
```

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

Using Drag and Drop

Take Command is compatible with Windows' Drag-and-Drop facility.

To add a filename to the command line using drag and drop simply drag the file from another application using the mouse, and release the mouse button with the file icon anywhere inside the Take Command window. The full name of the file will be pasted onto the command line at the current cursor position.

Take Command is a drag and drop "client", which means it can accept files dragged in from other applications and paste their names onto the command line as described above. It is not a drag and drop "server", so you cannot drag filenames from the Take Command window into other applications. However you can copy filenames and other text from the Take Command screen to other applications using the clipboard; see Highlighting and Copying Text for details.

DDE Support

Take Command can communicate with other Windows applications by using Dynamic Data Exchange or DDE. To use Take Command as a "DDE client" and send a message from Take Command to another application, use the DDEEXEC command.

You can also use Take Command as a "DDE server" and send commands to it from another application. To do so, use an application or server name of "TCMD" and a topic name of "Execute". The message can be any valid command that you could enter on the Command line: any alias, internal command, batch file, or external command. To send more than one line in a single DDE string, separate the lines with carriage return and line feed characters.

When using Take Command as a DDE Server you should start Take Command, execute the desired command, and exit Take Command. The DDE server is not intended to be used with a copy of Take Command running at the prompt; attempting to do so may result in unusual displays or display of the prompt in an incorrect location.

For example, if you use Microsoft Word for Windows you could use the following WordBasic fragment to send a DIR /W command to Take Command from within Word:

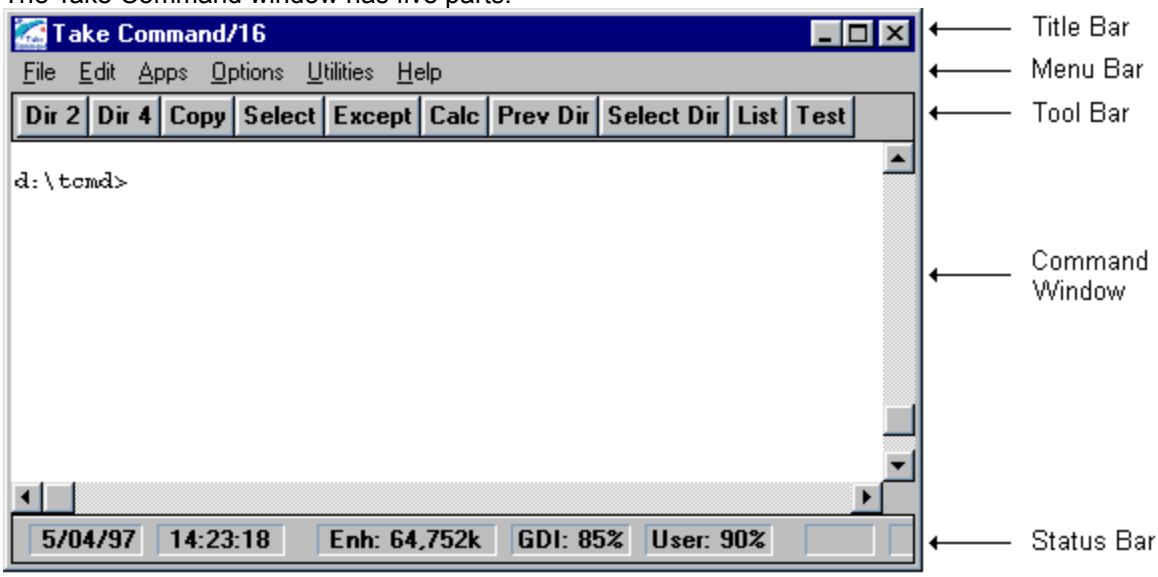
```
TCMDChannel = DDEInitiate("TCMD", "Execute")
DDEExecute TCMDChannel, "dir /w"
DDETerminate TCMDChannel
```

You could use the same approach to execute a batch file or any other Take Command command from within Word, and similar approaches are available in other applications which offer DDE support. Consult your application manual for complete details.

In most cases when you use Take Command as a DDE server you will want to redirect output from the commands you execute, because output on the Take Command screen is not likely to be useful to the client program which invokes a command.

The Take Command Screen

The Take Command window has five parts:



The **Title Bar** is the same as the one used in most Windows applications, with a control menu button on the left and the minimize and maximize buttons on the right. You can change the text that appears on the Title Bar, and adjust the size of the Take Command window, with the WINDOW command. You can also adjust the size of the Take Command window using standard window techniques, but see Resizing the Take Command Window for information about how Take Command's display changes when you do so.

See Take Command Menus for details about the **Menu Bar** and all of its menus.

The **Tool Bar** is used to execute internal or external commands, aliases, batch files, and applications with the click of a mouse. You can define up to 24 Tool Bar buttons; see Tool Bar Dialog for instructions. You can show or hide the Tool Bar with a choice on the Options Menu, with the configuration dialogs, or with the ToolBarOn directive in *TCMD.INI*.

The **Command Window** accepts your input and displays Take Command's output. You can use the scroll bars or the **Up arrow** and **Down arrow** keys to view text that has scrolled through the window. You can also save the contents of the Command Window and scrollback buffer to a file, restore a previous session from a file, copy text from Command Window to the clipboard, and copy text from the clipboard or from the Command Window to the command line. See Highlighting and Copying Text for information about saving and retrieving text in the Command Window and The Command Line for complete details about using the Command Line.

Finally, the **Status Bar** at the bottom of the Take Command window displays information about your system:

- The date and time, based on the Windows clock.
- The Windows mode (**Enh** for 386 enhanced mode, or **Std** for standard mode) and the amount of free virtual memory, in kilobytes. The amount of memory shown includes virtual memory that Windows creates on disk.

» The percentages of free GDI (Graphics Device Interface) and User resources. When either number drops below 25% the corresponding value will turn red. When resources are this low, Windows may begin to slow down and you may want to shut down some applications.

» The state of the Caps Lock key on the keyboard.

» The state of the Num Lock key on the keyboard.

You can show or hide the Status Bar with a choice on the Options Menu, with the configuration dialogs, or with the StatusBarOn directive in *TCMD.INI*.

If you use a laptop or LCD screen and find the "I-Beam" cursor in the Take Command window difficult to see, use an IBeamCursor = No directive in the **[TakeCommand]** section of the *TCMD.INI* file to force the use of an arrow cursor in all parts of the window.

Resizing the Take Command Window

You can resize the Take Command window at any time with standard Windows techniques (*e.g.*, by dragging a corner with the mouse). Resizing the window changes the number of rows and columns of text which will fit in the command window (the actual number of rows and columns for any given window size depends on the font you are using). Take Command reacts to these changes using two sets of rules: one for the height and one for the width.

When the height of the command window changes, future commands simply use the new height as you see it on the screen. For example, if you reduce the window to three rows high and do a DIR /P (display a directory of files and pause at the bottom of each visual "page"), DIR will display two lines of output, a prompt ("Press any key to continue ..."), and then pause. If you expand the window to 40 lines high and repeat the same command, DIR will display 39 lines, a prompt, and then pause.

However, when the width of the window changes, Take Command must check the current **virtual screen width**. The virtual width is the maximum number of characters on each line in Take Command's internal screen buffer. You can think of it as the width of the data which can be displayed in the Take Command window, including an invisible portion to the right of the window's right-hand edge. When the virtual width is larger than the actual width, a standard horizontal scroll bar is displayed to allow you to see any hidden output.

The screen height normally starts at 25 lines; you can alter this default with the ScreenRows directive in the *TCMD.INI* file, or the **Height** setting on the Display page of the configuration dialogs. The _ROWS internal variable can be used to determine the current screen height.

The virtual screen width starts at 80 columns or the number of columns which fit into the startup Take Command window, whichever is larger. You can alter the default minimum width of 80 columns with the ScreenColumns directive in the *TCMD.INI* file, or the **Width** setting on the Display page of the configuration dialogs. The _COLUMNS internal variable can be used to determine the current virtual screen width.

If you use keyboard commands or the mouse to expand the Take Command window beyond its previous virtual width, the virtual width is automatically increased. This ensures that the internal buffer can hold lines which will fill the newly enlarged window. If you contract the window, the virtual width is not reduced because this might require removing output already on the screen or in the scrollback buffer.

As a result, widening the window will make future commands use the new enlarged size (for example, as the window is widened DIR /W, which displays a "wide" directory listing, will display additional columns of file names). However, if the window is narrowed future commands will still remember the enlarged virtual width, and display data to the right of the window edge. The horizontal scroll bar will make this data visible.

When the font is changed, Take Command will recalculate the virtual screen width. The new virtual width will be the width set by the Screen Columns directive or the configuration dialogs, or the current width of the window in the new font, whichever is larger.

Take Command Menus

Like all Windows applications, Take Command displays a menu bar along the top of the Take Command window. To select a particular menu item, click once on the menu heading, or use **Alt-x** where "x" is the underlined letter on the menu bar (for example, **Alt-F** displays the File menu). You can also select a menu by pressing **Alt** or **F10** and then moving the highlight with the cursor keys.

The items on the menu bar allow you to select a variety of Take Command features:

File Menu
Edit Menu
Apps Menu
Options Menu
Utilities Menu
Help Menu

File Menu

The File menu allows you to save or print the screen buffer, or exit Take Command.

Save to File...

Saves the contents of the screen buffer to a file. A Save As dialog box appears in which you can enter the name of the file that you wish to use.

Print...

Sends the contents of the screen buffer to the printer. A Print dialog box appears in which you can choose the portion of the screen buffer you wish to print.

Setup Printer...

Displays a standard printer setup dialog box. The options available in the dialog box depend on the printer driver(s) you are using.

Refresh

Redraws everything on the Take Command screen (use this selection if the display appears incorrect, for example if it is not repainted properly after another application exits). You can also press F5 at the Take Command prompt to refresh the screen.

Exit Windows

Shuts down Windows and returns to the DOS prompt.

Restart Windows

Exits Windows and then automatically restarts it. This option may be useful for checking the effects of changes (for example, changes to Windows' *WIN.INI* or *SYSTEM.INI* files).

Exit

Ends the current Take Command session. If Take Command is running as the Windows shell, this option performs the same action as Exit Windows.

Edit Menu

The Edit menu allows you to copy text between the Take Command window and the Windows clipboard. (You can also access the clipboard with redirection to or from the CLIP: device, or with the @CLIP variable function.)

To use the the Copy command you must first select a block of text with the mouse or with the Select All command, below. If you hold down the mouse button 2 while you select a block of text, that block will be copied to the clipboard automatically when you release the button.

The commands on this menu can also be invoked with keystrokes. Some Windows applications support the use of **Ctrl-C** for Copy, **Ctrl-X** for Cut, and **Ctrl-V** for Paste. Take Command uses a different set of keystrokes (**Ctrl-Ins**, **Shift-Del**, and **Shift-Ins**) to avoid conflicts with the use of **Ctrl-C** under DOS, and **Ctrl-X** under 4DOS.

For more information on copying text see Highlighting and Copying Text.

Cut

Removes text from the Take Command command line and moves it to the clipboard.

Copy

Copies selected text from the Take Command screen buffer to the Windows clipboard.

Paste

Copies text from the Windows clipboard to the command line. If the text you insert contains a line feed or carriage return, the command line will be executed just as if you had pressed Enter. If you insert multiple lines, each line will be treated like a command typed at the prompt.

Delete

Removes text from the Take Command command line.

Copy + Paste

Copies the selected text directly to the command line.

Select All

Marks the entire contents of the Take Command screen buffer as selected text.

Apps Menu

Run

Displays the run dialog box from which you can run an application or batch file. Take Command remembers the commands you have run from this dialog in the current session. To select from this list click on the drop-down arrow to the right of the "Command Line" field, or press the down-arrow.

DOS Box

Starts a DOS session by running the default command processor. The DOS session is started using the Windows *_DEFAULT.PIF* file. To start a DOS session with your own PIF (Program Information File) settings enter the name of the corresponding *.PIF* file at the Take Command prompt.

Task List

Displays the Task List Dialog which shows all tasks currently running. When the dialog is displayed, you can choose to switch to a different task, display information about a task, or close a task. You can also close a specific window; doing so may also close the associated task (depending on how the program whose window you are closing was written).

Normally you can also call up the Take Command task list with Ctrl-Esc. However this link can be disabled (so that Ctrl-Esc invokes the standard Windows task manager) using the Task List **Enable** checkbox on the Options 2 page of the configuration dialogs, or the TCMDTaskList directive in the *TCMD.INI* file.

End DOS app in Caveman

Terminates the DOS program currently running under Caveman, and destroys the Caveman virtual machine (the virtual machine will be restarted when necessary to run another DOS program). Work you were doing in the application running under Caveman may be lost. Use this option if a DOS program "hangs" and does not respond to **Ctrl-C** or **Ctrl-Break**.

Take Command Group Lists

At the end of the Run Menu, you will see the names of the groups defined in your Windows shell (Program Manager or a similar program). If you select a group name, a sub-menu will appear with the items in the group. When you click on an item in the sub-menu, it will be launched for you. If the item is set to run minimized, Take Command will start it minimized. (The group lists only appear in Take Command/32 if you are using Windows NT 3.51 or earlier. Under Windows 95 and Windows NT 4.0 and above, use the Start button for access to your program groups.)

Take Command collects the group names and the list of applications in each group by communicating with the Program Manager (or another Windows shell which responds to Take Command's request for group information), or by reading the Program Manager's *.GRP* files.

For more information or to change the way group names are collected see Take Command and Windows Shells.

Options Menu

Configure Take Command

Opens a series of dialogs which you can use to change the configuration of Take Command.

Configure Tool Bar

Opens a dialog in which you can define up to 24 buttons for the Take Command tool bar.

Select Font

Opens a dialog box from which you can select the font and type size for the Take Command window. You can choose from any monospaced font that has been properly installed in Windows. (A monospaced font is one which uses the same fixed width for all characters, rather than varying the width based on each character's shape.) When you change the font, everything in the command window is displayed in the new font. You cannot mix fonts in the Take Command command window.

Caution: Some fonts will not display all of the characters used by Take Command. In particular the DRAWBOX, DRAWHLINE, and DRAWVLINE commands use line-drawing characters which are not included in many standard fonts. To ensure that these commands work properly, use the "Terminal" font, which typically does include line-drawing characters.

You can experiment with different fonts by using DRAWBOX or a similar command to display line drawing characters in the command window. Then use Select Font to choose a different font. When you leave the dialog, the new font will be used to display the entire command window, including the line drawing characters. You can select different fonts until you find one that satisfies you.

Logging

Controls logging via a submenu with two choices:

Command: Enables or disables command logging using the default TCMDLOG file or the file you have chosen with the LOG command or the LogName directive in the *TCMD.INI* file.

History: Enables or disables command history logging using the default TCMDHLOG file or the file you have specified with the LOG /H command or the HistLogName directive in the *TCMD.INI* file.

Show Tool Bar

Select this item to enable or disable the Take Command tool bar, which appears near the top of the Take Command window. The tool bar will not appear until you have defined at least one item for it with the Configure Tool Bar command, above.

Show Status Bar

Select this item to enable or disable the Take Command status bar, which appears near the bottom of the Take Command window.

VM Setup

Set the parameters for the Caveman "Virtual Machine" used to run DOS applications from Take Command. This option is only available if Caveman is enabled. For additional details on Caveman see the Take Command and DOS Applications and Caveman topics.

Utilities Menu

Find Files/Text

Opens the find files dialog box which lets you search for files or text interactively (see FFIND to search from the command line).

Once Take Command has created a list of files based on your specifications, you can double-click on a file name and Take Command will display an information box about the file. From the information box, you can choose to list, edit, or run the file.

Descriptions

Opens an edit window in which you can view and change the descriptions of files in any directory available on your system. See DESCRIBE for details on file descriptions.

Aliases

Opens an edit window in which you can view and change the list of current aliases. You can also use this window to import aliases from a file or save all current aliases in a file. Any changes you make will take effect as soon as you close the Aliases window.

Environment

Opens an edit window in which you can view and change the current environment. Any changes you make will be immediately recorded in Take Command's environment.

Editor

Starts the Windows Notepad editor or any other editor you have specified with the Editor directive in the *TCMD.INI* file or on the Commands page of the configuration dialogs.

Recorder

Starts the Windows Recorder to create keystroke macros.

Help Menu

See Take Command Help for more information about how to use the help system.

Contents

Displays the Table of Contents for Take Command Help.

How to Use Help

Displays the standard text that explains the Windows help system.

Search Topics

Displays the Search dialog for Take Command Help. This is the same dialog you will see if you click on the **Search** button from within the help system.

About

Displays Take Command version, copyright, and license information.

Take Command Dialogs

Like all Windows applications, Take Command includes a number of dialogs which allow you to enter information or configure the program. The dialogs are listed here for quick-reference, though in general you will find it easier to learn about each one from the context in which it is used (for example, the information referenced below on the tool bar dialog will be more useful after you have read the section on the tool bar itself).

Take Command uses standard Windows dialogs for tasks like printing, selecting a font, or browsing files and directories. These dialogs are provided by Windows, not Take Command, and are common to many different Windows programs; they are not documented within this help system.

The reference in parentheses after certain dialogs listed below shows the name of the menu you can use to access that dialog.

- Run Program Dialog (Apps)
- Windows and Task Lists Dialog (Apps)
- Configuration Dialogs: (Options)
  - Startup Options Dialog
  - Display Options Dialog
  - Command Line 1 Options Dialog
  - Command Line 2 Options Dialog
  - Options 1 Dialog
  - Options 2 Dialog
  - Command Options Dialog
- Tool Bar Dialog (Options)
- Virtual Machine Setup Dialog (Options)
- Caveman Error Dialog
- Find Files/Text Dialog (Utilities)
- Edit Descriptions Dialog (Utilities)
- Aliases Dialog (Utilities)
- Environment Dialog (Utilities)

Run Program Dialog

The Run Program dialog, started from the Apps menu, allows you to run a program by typing its name or browsing the disk.

Enter the name of the program you wish to run, along with any command-line parameters, in the Command Line field.

When the focus is on the Command Line field, the up and down arrow keys will display previous command lines, one at a time.

If you click on the arrow to the right of the Command Line field, you will see a list of previous commands you have issued from this dialog. You can select any item from the list to re-execute or to modify.

You can enter the name of a data file on the Command Line if an Executable Extension has been defined for the file.

The **Normal**, **Minimized**, and **Maximized** radio buttons determine the type of window that will be used for the program. If you select Minimized, the program will start as an icon on the desktop. Maximized starts the program in a full-screen window. The Normal button lets the operating system select the size and position of the program's window.

The **Browse** button leads to standard file browser from which you can select any executable program. Your choice will be placed in the Command Line edit box, and you can add parameters before selecting OK to run the program.

Windows and Tasks List Dialog

The Windows and Tasks List dialog, available from the Apps menu, shows the windows and tasks that are currently running.

The **Windows** list on the left shows each open window. The first name in each row is the internal shorthand name used for the window. The second name is the current title of the window. The buttons below the **Windows** list let you switch to an application or close a window and end its associated program. If you pick **Cancel**, no action is taken and you are returned to Take Command. Most tasks will have a single window, but a task may have no window, or more than one window.

The **Tasks** list, on right side of the dialog, shows the tasks that are currently running. If you select a task and then pick Info, you will see information about that task and about the program that started it. You can also choose to end the task.

Often, closing a window will also end the associated task and ending a task will close the associated window. However, depending on how a program is written, you may have to do both in order to completely end a program.

To disable the Take Command task list and use the default Windows Task Manager when Ctrl-Esc is pressed within the Take Command window, clear the Task List **Enable** checkbox on the Options 2 page of the configuration dialogs, or set the TCMDTaskList directive to No in the *TCMD.INI* file.

Configuration Dialogs

These dialogs, available from the Options menu, control the configuration of Take Command. Each option in one of the configuration dialogs sets a corresponding directive in the *TCMD.INI* file. You can start the configuration dialogs with the **Configure Take Command** selection on the Options menu.

Unless you select the **Cancel** button, any changes you make will take effect immediately. If you return to Take Command by selecting the **OK** button, new settings will only stay in effect until you end the current Take Command session. If you return to Take Command by selecting **Save**, the changes will be recorded in the **[TakeCommand]** section of the *TCMD.INI* file and will be in effect each time you start Take Command.

For details about the *TCMD.INI* file and *.INI* file directives, the allowable ranges for each, and the effect of each, see TCMD.INI.

While you are using the dialogs, you can move between sets of configuration options with the list box in the left-hand pane. The sets of options available in this dialog are:

Startup	Options 1
Display	Options 2
Command Line1	Commands
Command Line 2	

Startup Options Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

You can set the path to your **TCSTART** and **TCEXIT** files if they aren't in the same directory as Take Command. This field sets the TCStartPath directive.

In the **Buffer Sizes** section:

- Command History sets the size of the command history list, and the value of the History directive.
- Directory History sets the size of the directory history list, and the value of the DirHistory directive.

The **Cursor** section sets the shape of the cursor and the IBeamCursor directive. Use the I-Beam shape for normal systems, and the Arrow shape for laptop or other systems where the I-Beam cursor is hard to see.

The **Display** section sets the size and location of Take Command's window when it starts up:

- Standard, Max, Min, and Custom set the WindowState directive.
- The window position and size fields set the WindowX, WindowY, WindowHeight, and WindowWidth directives. Use these if you want Take Command to start up at a specific location on your desktop. These fields are ignored unless the item above is set to Custom.

The Load Associations item in the **Options** section determines whether direct Windows file associations are loaded when Take Command starts, and sets the LoadAssociations directive.

Display Options Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

The **Text Dimensions** section configures the way that text appears in Take Command's main window:

- » Width sets the minimum width of the Take Command virtual screen, and the ScreenColumns directive.
- » Height sets the initial height of the Take Command screen , and the ScreenRows directive.
- » Tabs selects the location of tab stops for Take Command's output (including the output from the LIST and TYPE commands), and sets the TabStops directive.

The **Window Configuration** section controls the appearance of the Tool Bar and Status Bar. For each bar:

- » If you check Enable, the bar will be visible (the Tool Bar will only be visible if you have defined at least one button for it). Enable sets the StatusBarOn or ToolBarOn directive. These settings are also modified when you enable and disable the toolbar from the Options menu.
- » The Font Height sets the size of text on the Tool Bar and Status Bar, and the StatBarText or ToolBarText directive.

The **Scrolling** section controls Take Command's screen scrollback buffer:

- » Buffer Size sets the size of the screen buffer, and the value of the ScreenBufSize directive. Valid sizes are from 16000 to 64000 bytes.
- » Scroll Lines controls how much the screen scrolls when Take Command's text has reached the bottom of the window, and sets the ScrollLines directive. Lower values produce smoother but slower scrolling.

The **Colors** section sets screen colors used by Take Command. When both boxes are set to "(Default)" the default colors are used:

- » Output establishes the colors Take Command uses for the text it displays, and sets the StdColors directive.
- » Input establishes the colors for echoing the commands you type, and sets the InputColors directive.

The **ANSI Support** section determines whether Take Command's ANSI support is enabled.

Command Line 1 Options Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

The **Editing** section controls command-line editing:

- » Default Mode selects whether you begin editing in Overstrike or Insert mode, and sets the EditMode directive.
- » Cursor sets the width of the cursor for both Overstrike and Insert modes, and sets the CursorIns and CursorOver directives. The width is expressed as a percentage of the width of a character cell.

The **Filename Completion** section controls filename completion at the command prompt:

- » The Add \ to Directories setting determines whether a backslash is added automatically at the end of a directory name during filename completion. (A trailing backslash is always appended to a directory name at the beginning of the command line to enable automatic directory changes, regardless of this setting.) This option sets the AppendToDir directive.
- » The Options setting allows you to customize filename completion for specific commands, and sets the FileCompletion directive. See Customizing Filename Completion for details on the use of this setting.

In the **Command History** section:

- » The Scroll / History keys setting controls whether Take Command or 4DOS defaults are used for scrolling through the scrollback buffer and the command history. The Normal setting uses the arrow and PgUp / PgDn keys for the scrollback buffer, and the corresponding control keys (Ctrl-Up, Ctrl-Down, etc.) for the command history. The Swapped setting reverses these assignments. For more details see Scrolling and History Keystrokes. This option sets the value of the SwapScrollKeys directive.
- » Minimum saved characters sets the size of the shortest line that will be saved in the command history, and sets the value of the HistMin directive.
- » Copy to end and Move to end, if checked, copy or move a recalled command to the end of the history list each time it is executed, and set the HistCopy and HistMove directives. Copy to end takes precedence over Move to end, so the Move to end setting will be ignored if Copy to end is also checked.
- » Wrap, if checked, enables the command history recall to "wrap" when you reach the top or bottom of the list. This control sets the HistWrap directive.

Command Line 2 Options Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

The **Popup Windows** section controls the position and size of the two different kinds of popup windows available from the command line:

- The History entries set the initial position and size of the command history window, directory history window, and filename completion window, and set the PopupWinLeft, PopupWinTop, PopupWinWidth, and PopupWinHeight directives.
- The Extended Directory Search entries set the initial position and size of the popup window used for extended directory searches, and set the CDDWinLeft, CDDWinTop, CDDWinWidth, and CDDWinHeight directives.

The **Extended Directory Search** section controls Extended Directory Searches:

- Search Level sets the type of search to use (the default of 0 disables extended searches). See Extended Directory Searches for details on the meaning of the different search levels. This option sets the FuzzyCD directive.
- Tree Path sets the location of the extended directory search database, and the TreePath directive.

Options 1 Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

The **Descriptions** section sets the way that Take Command handles file descriptions entered with the DESCRIBE command.

- » The Enable checkbox enables or disables the display and processing of descriptions, and sets the Descriptions directive.
- » The Maximum Length field determines the maximum size of file descriptions and sets the DescriptionMax directive.

The **Special Characters** section sets the characters that have special meaning for Take Command. See Special Character Compatibility for details on using compatible characters for different JP Software products.

- » Separator is the character that separates multiple commands. It can also be set with the CommandSep directive or SETDOS /C.
- » Escape is the escape character which suppress the normal meaning of the following character. It can also be set with the EscapeChar directive or SETDOS /E.
- » Parameter sets the character used after a percent sign to specify all or all remaining command-line arguments in a batch file or alias. It can also be set with the ParameterChar directive or with SETDOS /P.
- » Decimal and Thousands set the characters used as the decimal and thousands separators for displayed output, numeric IF and IFF tests, version numbers, numeric functions (*e.g.*, @COMMA or @EVAL), and other similar uses. The default of Auto tells Take Command to use the characters associated with your current country code. If you change one character you must also adjust the other so that the two characters are different. These settings can also be changed with the DecimalChar and ThousandsChar directives, or with SETDOS /G.

Default Beep sets defaults for the BEEP command and for "error" beeps. To disable error beeps, set the beep length to 0, and be sure to specify an explicit length each time you use the BEEP command.

- » Length sets the length of the beep in timer ticks that are approximately 1/18 of a second each. It also sets the BeepLength directive.
- » Frequency sets the frequency of the beep in Hz. It also sets the BeepFreq directive.

The **Options** section sets miscellaneous options.

- » Force upper case, when selected, forces Take Command to display file names in upper case in internal commands like DIR and COPY. You can use the UpperCase directive or the SETDOS /U command achieve the same result.
- » Default batch echo, if selected, turns on echoing in batch files by default. You can use the BatchEcho directive or the SETDOS /V command to achieve the same result.
- » Protect redirected output files, if selected, keeps Take Command from overwriting an existing file with redirected (>) output, and from creating a new file with output redirected in append (>>) mode. You can achieve the same result with the NoClobber directive or the SETDOS /N command.
- » The Time options determine how Take Command displays times. If you select Country, the time display is based on your country settings in Windows. The am/pm setting forces a 12-hour display with a trailing "a" or "p." The 24-hour setting forces a standard 24-hour display. You can also set the time display with the AmPm directive.

» The CUA option determines the keys used when copying text between the Take Command window and the clipboard. The default setting of CUA selects standard Common User Access keys (Ctrl-Del for cut, Ctrl-Ins for copy, and Shift-Ins for paste); the Windows setting selects standard Windows keys (Ctrl-X for cut, Ctrl-C for copy, and Ctrl-V for paste). For additional details see Highlighting and Copying Text. This option sets the CUA directive.

Options 2 Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

The **Logging** section enables or disables Command and History logging (see the LOG command) and sets the file name to use for each. It also sets the LogName and HistLogName directives.

The **Program Manager Interface** section sets the method that Take Command uses to communicate with Program Manager or a replacement Windows shell. See the ProgmanDDE directive and the section on Using Take Command as the Windows Shell for more details.

The **EVAL** section sets the minimum and maximum number of digits after the decimal point that @EVAL will display. You can achieve the same results with the EvalMin and EvalMax directives or with the SETDOS /F command.

The **External Programs** setting controls whether Take Command waits for applications to complete before displaying the prompt. See Waiting for Applications to Finish for details on the effects of this setting.

The **Task List** setting controls whether Take Command displays its own windows and task lists dialog or the default Windows Task Manager when you press Ctrl-Esc from the Take Command window. To use the Windows task manager, uncheck the **Enable** box. This option sets the TCMDTaskList directive.

Command Options Dialog

[If you are not familiar with the purpose or use of the configuration dialogs, or the effects of exiting with **OK**, **Save**, or **Cancel**, review the main configuration dialogs topic before continuing.]

The **DIR Colors** field sets the colors used by DIR and SELECT. You can achieve the same effect with the ColorDir directive or by setting the COLORDIR environment variable. See Color Coded Directories under the DIR command for details.

The **LIST** section sets the foreground and background colors for the LIST command, and the ListColors directive.

The **SELECT** section sets the foreground and background colors for the SELECT command, and the SelectColors directive.

The **Editor Filename** sets the path and name of the program you want to use when you use when you select Editor from the Utilities menu. It also sets the Editor directive.

Tool Bar Dialog

This dialog, available from the Options menu, allows you to define or modify buttons on the tool bar.

Select the button you want to define or modify in the box on the left.

Enter the button's label and the command to be executed when you select the button in the fields at the bottom of the dialog box. You can enter multiple commands in the Command field by separating them with the command separator character [**^**].

You can use the Browse button to find a path and filename to be entered at the beginning of the Command field.

Before entering a command to start an application program, be sure to check whether the application must be started in a particular directory. If so, and you have a *.PIF* file for the application, the startup directory should be specified there. Otherwise, you can use a CDD command in the Command field before the application name, for example:

```
cdd d:\docfiles ^ d:\winword\winword.exe
```

If the command line begins with an at-sign [**@**] it will not be added to the command history when the tool bar button is clicked. Otherwise, all commands executed from the tool bar are stored in the history.

Use the radio buttons to select how you want the command to be executed:

- **Echo** means display the tool bar command on the command line, but do not execute it. You can add additional text to the line if you wish, then press Enter to execute the command. This is the default setting.
- **Echo & Execute** means display the tool bar command on the command line, then execute it immediately, without waiting for you to press Enter.
- **Execute w/o Echo** means execute the tool bar command immediately, without waiting for you to press Enter, and without displaying the command.

The **Font Size** setting applies to all of the buttons on the tool bar. If you make the font size too large (or make the Take Command window too small), buttons on the right hand end of the toolbar may not be visible even if the Take Command window is expanded to fill the screen.

If you exit by choosing the OK button, any changes you have made will be saved in the *TCMD.INI* file, and reloaded automatically the next time you start Take Command. If you use the Cancel button, your changes will be discarded.

If you don't specify a label for a button, a small space is created on the tool bar. For example, if you define buttons 5 and 7 but leave button 6 blank, Take Command will leave a space between buttons 5 and 7 when the tool bar is displayed. You can use this feature to separate groups of buttons on the tool bar; however, the total number of buttons, including empty buttons, cannot exceed 24.

If you want to rearrange the order of the buttons on the tool bar, use an editor (*e.g.* Windows' Notepad) to edit the [Buttons] section of the *TCMD.INI* file. Simply rearrange the lines into the order you wish, and renumber the buttons accordingly.

Caveman Virtual Machine Setup Dialog

The internal facility used to run DOS applications inside the Take Command window is called "Caveman". For more details on Caveman see the separate topics on Take Command and DOS Applications and on Caveman. This dialog, available from the Options menu, allows you to configure the "Virtual Machine" (VM) created by Caveman to run DOS applications and display their output on the Windows screen.

Many of the settings in this dialog are similar to those in a standard PIF file, except that they apply to the Caveman VM.

Changes to the memory and priority settings will not take effect until the next time the Caveman VM Is created (Reuse Caveman VM setting below for details).

The memory **Required** settings establish the minimum amount of each type of memory that will be allocated to a Caveman VM. The **Wanted** field sets the amount of memory you would like to have for the VM, if possible. **Limit** settings define the maximum amount of memory of each type which the Caveman VM can use. If you set the **Wanted** or **Limit** field to -1, Caveman will request as much memory as Windows has available.

Windows will retain any memory marked **Locked** in RAM and not swap it to virtual memory (usually a swap file on your hard disk). This makes memory access somewhat faster but prevents other applications from using the same physical RAM space.

The **Foreground Priority** setting establishes how much processing time Caveman will receive from Windows when a DOS application is run under Caveman with Take Command visible in the foreground on your Windows desktop. The **Background Priority** field sets the processing priority when a DOS application is run under Caveman with Take Command behind another window, or minimized. These values can range from 0 (lowest priority) to 100 (highest priority).

If you select **Run DOS Apps in Caveman VM**, Take Command will run DOS applications in the Caveman VM by default. You can then override the default and force a DOS program to start in a separate window by using the START command, or creating a *.PIF* file for the application. If you deselect this option DOS applications will run in a separate window by default. You can then override the default and start a DOS program under Caveman with the START command's **/CM** switch. See Take Command and DOS Applications for complete details on this option.

When **Reuse Caveman VM** is selected (the default), the Caveman VM Is created the first time you run a DOS program under Caveman, and destroyed when Take Command exits. The VM will remain in existence between uses, and subsequent DOS programs will start more quickly. However, if you select this option, other Windows applications will not be able to use the RAM, virutal memory, and other resources allocated to the Caveman VM, and this may reduce their performance slightly.

If **Reuse Caveman VM** is not selected the Caveman VM is restarted for each DOS program. This will make DOS programs run under Caveman start more slowly, but will free the VM's resources for use by Windows applications when DOS programs are not running.

If you exit by using the **Save** button, the settings will be saved in the *TCMD.INI* file and used in future Take Command sessions. If you exit by using the **OK** button, any new settings will only remain in effect during the current session.

Caveman Error Dialog

This dialog appears when an error has occurred because you tried to run an incompatible DOS program under Caveman. For details on running DOS programs, including a description of the types of programs which work properly under Caveman, see the Take Command and DOS Applications and Caveman topics.

Select **Restart** to restart the application now in a separate window (rather than under Caveman).

Select **Cancel** to return to the Take Command screen without restarting the application.

In either case, if you click the **Always use separate window** button at the bottom of the screen Take Command will remember that this program will not work under Caveman, and will automatically run it in a separate window the next time you start it. The list of applications incompatible with Caveman is kept in the **[DosApps]** section of the *TCMD.INI* file.

The **Error code** shown on the screen may be helpful in explaining why your application does not work under Caveman. The two most common codes are:

09 00 00	The program attempted to retrieve keystrokes directly from the keyboard hardware. Caveman cannot support programs which access the keyboard hardware directly.
10 xx xx	The program tried to use a non-standard video mode (for example, a graphics mode), modify the video font or color palette, or perform other specialized video operations usually associated with full-screen programs. Caveman is designed for "teletype-style" programs and only supports standard text modes.
21 25 09	The program attempted to retrieve keystrokes directly from the keyboard hardware (see code 09 00 00 above).

Find Files/Text Dialog

The Find Files/Text dialog box, available from the Utilities menu, gives you the same features as the FFIND command, in dialog form.

Enter the file name or names you wish search in the **Files** field. You can use wildcards and include lists as part of the file name. To select files from previous searches in the same Take Command session, click on the down arrow beside the Files field, or press the up or down arrow while the input cursor is in the Files field. You can also use the **Browse** button to find files to include in the search.

Enter the text (or hexadecimal values) you are searching for in the **Text** field. You can use extended wildcards in the search string to increase the flexibility of the search. Use back-quotes [`] around the text if you want to search for characters which would otherwise be interpreted as wild cards, such as the asterisk [*****], question mark [**?**], or square brackets. For example, to search for an A, followed by some number of other characters, followed by a B, enter the A*B as your search string. To search for the literal string A*B (A, followed by an asterisk, followed by B), enter `A*B` as your search string search string (the closing back-quote is optional).

Enter the drive(s) you want to search in the **Disks** field. This field is ignored unless **Entire Disk** is selected in the **Search** portion of the dialog. If you select **All Hard Disks**, this field is set automatically to include all hard disk drive letters Take Command finds on your system.

The **Match Case** box, when it is selected, makes the search case-sensitive. The **Hex Search** option signals that you are searching for hexadecimal values, not ASCII characters.

The **Hex Search** option signals that you are searching for hexadecimal values, not ASCII characters. To search for specific hexadecimal values (for example, to look for non-printing characters), check **Hex Search** and enter the string in the **Text** field as a series of one- or two-digit hexadecimal characters, separated by spaces (*e.g.* 42 6F 70). See the ASCII table for hexadecimal values of ASCII and extended ASCII characters.

If you enable **All Lines**, every line from every file that contains the search text will be displayed. If this option is not enabled, only the first line from such a file will be displayed.

Unless you enable the **Hidden Files** option, files with the hidden attribute will not be included in the search.

The radio buttons in the **Search** area let you specify where you want Take Command to look for files. If you select **Dir Only** or **Dir & Subdirs**, the search will begin in the current default directory, shown above the Files box. If you select **Entire Disk**, Take Command will use the drives that you specified in the **Disks** field. If you select **All Hard Disks**, Take Command will search all the hard disk drives it finds on your system.

To start the search, press the **Find** button. Once the search has started the **Find** button changes to a **Stop** button, which can use to interrupt the search before it is finished.

Once the search is finished, you can save the list of matching files with the **Export** button.

If you select one of the matching files in the list (by double-clicking on it, or selecting it with the cursor and pressing Enter), Take Command will display another dialog with complete directory information about the file. From that dialog you can **Run** the file (if it is an executable file, a batch file, or has an executable extension), display the file with the **LIST** command, or **Edit** the file. If you choose List, the cursor will be placed on the first matching text within the file. When you exit from LIST or the editor, the original list of matching files will still be available.

Edit Descriptions Dialog

This dialog is available from the Utilities menu. Most of it looks and works like a standard file browser. The pane at the bottom of the dialog lets you view, enter, or edit the description for any file. See DESCRIBE for more information on file descriptions.

File descriptions can also be entered or changed with the DESCRIBE command, and are visible when you use the DIR and SELECT commands.

Changes you make in this dialog are not saved in the alias list until you click the **OK** button. If you click **Cancel** the changes are discarded.

Aliases Dialog

This dialog is available from the Utilities menu. The current list of aliases is shown in the pane on the left. As you move the cursor in the pane, the name of the alias under the cursor is shown in the **Name** field and its definition is shown in the **Value** field. You can use these fields to edit the alias.

To add a new alias to the list, use the **Add** button. The Name and Value fields will be cleared so you can enter new values in each. To save the new entry, switch to a different entry or press Enter.

The **Delete** button deletes the highlighted alias.

The **Import** button reads a list of aliases from a file (similar to the ALIAS /R command). The **Export** button writes the current list to a file.

Changes you make in this dialog are not saved in the alias list until you click the **OK** button. If you click **Cancel** the changes are discarded.

Environment Dialog

This dialog is available from the Utilities menu. The current environment variables are shown in the pane on the left. As you move the cursor in the pane, the name of each entry appears in the **Name** field and its value appears in the **Value** field. You can use these fields to edit the environment entry.

To add a entry to the list, use the **Add** button. The Name and Value fields will be cleared so you can enter new values in each. To save the new entry, switch to a different entry or press Enter.

The **Delete** button deletes the highlighted environment variable.

The **Import** button reads a list of environment variables from a file (similar to the SET /R command). The **Export** button writes the current list to a file.

Changes you make in this dialog are not saved in the environment until you click the **OK** button. If you click **Cancel** the changes are discarded.

Tool Bar

The Take Command screen has an optional Tool Bar that you can use to execute internal or external commands, aliases, or batch files with the click of a mouse.

To create buttons for the Tool Bar, select Configure Tool Bar from the Options menu. This selection displays the tool bar dialog.

You can define up to 24 Tool Bar buttons.

To enable or disable the Tool Bar, use the ToolBarOn directive in the *TCMD.INI* file, the Tool Bar Enable setting on the Display page in the configuration dialogs, or the Show Tool Bar command in the Options menu.

The configuration dialog and *TCMD.INI* settings are modified when you enable and disable the tool bar from the Options menu. This preserves the tool bar state when you close Take Command, and restores it the next time you start a Take Command session.

Status Bar

The Take Command screen has an optional Status Bar that shows information about your system. To enable or disable the Status Bar, use the StatusBarOn directive in the *TCMD.INI* file, the Status Bar Enable setting on the Display page in the configuration dialogs, or the Show Statusbar command in the Options menu.

The configuration dialog and *TCMD.INI* settings are modified when you enable and disable the status bar from the Options menu. This preserves the tool bar state when you close Take Command, and restores it the next time you start a Take Command session.

The status bar displays the following information:

- The date and time, based on the Windows clock.
- The Windows mode (**Enh** for 386 enhanced mode, or **Std** for standard mode) and the amount of free virtual memory, in kilobytes. The amount of memory shown includes virtual memory that Windows creates on disk.
- The percentages of free GDI (Graphics Device Interface) and User resources. When either number drops below 25% the corresponding value will turn red. When resources are this low, Windows may begin to slow down and you may want to shut down some applications.
- The state of the Caps Lock key on the keyboard.
- The state of the Num Lock key on the keyboard.

Using the Scrollback Buffer

Take Command retains the text displayed on its screen in a "scrollback buffer".

You can scroll through this buffer using the mouse and the vertical scroll bar at the right side of the Take Command window, just as you can in any Windows application.

You can also use the **Up Arrow** [] and **Down Arrow** [↓] keys to scroll the display one line at a time from the keyboard, and the **PgUp** and **PgDn** keys to scroll one page at a time.

If you scroll back through the buffer to view previous output, and then enter text on the command line, Take Command will automatically return to the bottom of the buffer to display the text.

If you prefer to use the **Arrow** and **PgUp** keys to access the command history (as in 4DOS), see the SwapScrollKeys directive in the *TCMD.INI* file, or the corresponding option on the Command Line 1 page of the configuration dialogs. SwapScrollKeys switches the keystroke mapping so that the , ↓, and **PgUp** keys manipulate the command history, and **Ctrl-**, **Ctrl-**↓, **Ctrl-PgUp**, and **Ctrl-PgDn** are used to control the scrollback buffer. For more details see Scrolling and History Keystrokes.

You can set the size of the scrollback buffer on the Display page of the configuration dialogs, or with the ScreenBufSize directive in the *TCMD.INI* file.

To clear the entire scrollback buffer, use the CLS /C command.

Highlighting and Copying Text

While you are working at the Take Command prompt you can use common Windows keystrokes to edit commands, and use the Windows clipboard to copy text between Take Command and other applications. You can also select all of the text in the Take Command screen buffer by using the Select All command on the Edit menu.

To copy text from the Take Command window to the clipboard, first use the mouse to highlight the text, then press **Ctrl-Ins**, or use the Copy command on the Edit menu.

If you double-click on a word in the Take Command window, the entire word is highlighted or selected. You can also mark the text using mouse button 2 (normally, the right mouse button); in this case the text will be copied to the clipboard immediately when you release the mouse button.

To highlight text on the command line use the **Shift** key in conjunction with the **Left**, **Right**, **Ctrl-Left**, **Ctrl-Right**, **Home**, and **End** cursor movement keys. The **Del** key will delete any highlighted text on the command line, or you can type new text to replace the highlighted text.

While the Take Command window contains text, it is not a document window like those used by word processors and other similar software, and you cannot move the cursor throughout the window as you can in text processing programs. As a result, you cannot use the Windows shortcut keys like **Shift-Left** or **Shift-Right** to highlight text in the window. These keys work only at the command line; to highlight text elsewhere in the window you must use the mouse.

To copy text from the clipboard to the command line use **Shift-Ins**, or the Paste command on the Edit menu.

To paste text from elsewhere in the Take Command window directly onto the command line, highlight the text with the mouse and press **Ctrl-Shift-Ins**, or use the Copy+Paste command on the Edit menu. This is equivalent to highlighting the text and pressing **Ctrl-Ins** followed by **Shift-Ins**. It's a convenient way to copy a filename from a previous DIR or other command directly to the command line.

Some Windows applications support the use of **Ctrl-C** for Copy, **Ctrl-X** for Cut, and **Ctrl-V** for Paste. Take Command uses a different set of keystrokes to avoid conflicts with the use of **Ctrl-C** under DOS, and **Ctrl-X** under 4DOS. If you prefer, you can configure Take Command to use **Ctrl-C**, **Ctrl-X**, and **Ctrl-V** for Copy, Cut, and Paste by changing the **Edit keys** setting on the Options 1 page of the configuration dialogs, or with the CUA directive in the *TCMD.INI* file.

You should use caution when pasting text containing carriage return or line feed characters onto the command line. If the text you insert contains one of these characters the command line will be executed just as if you had pressed Enter. If you insert multiple lines, the text will be treated just like multiple lines of commands typed at the prompt.

You can also use Windows' Drag and Drop facility to paste a filename from another application onto the command line, and you can access the clipboard with redirection to or from the CLIP: device, or with the @CLIP variable function.

Take Command for 4DOS, 4OS2, and 4NT Users

If you're a 4DOS, 4OS2, or 4NT user, many of the features in Take Command will seem very familiar. Because the underlying command processing in Take Command is based on our character-mode products, you'll find the features of those products are readily accessible. All the commands and switches you're familiar with work the same way and have the same meaning in Take Command; the only exceptions are those that don't make sense in the Windows environment.

Other 4DOS, 4OS2 and 4NT features are included in Take Command as well you'll find support for command line editing, command and directory histories, aliases, *.BTM* files, and virtually all the other features you already know.

Even if you've never used our other products, you'll notice plenty of familiar items in Take Command. Like these products, Take Command is compatible with the default DOS, OS/2, Windows 95, and Windows NT command processors (*COMMAND.COM* and *CMD.EXE*), which you may have used from DOS, or the OS/2 or Windows desktop.

There are also a few differences between running under 4DOS, 4OS2, and 4NT (or *COMMAND.COM* or *CMD.EXE*) and running under Take Command. The primary differences are related to different methods for starting DOS programs; this topic is covered in detail under Take Command and DOS Applications. You should read the information there before changing Take Command's default options for starting DOS programs.

In order to support the Take Command screen scrollback buffer, some Take Command keystrokes are different from what you may be used to in our character-mode products. See Scrolling and History Keystrokes for more details.

Some command-line editing defaults have also been changed to conform more closely to Windows conventions. In Take Command the default editing mode is insert, not overstrike, and the default insert-mode cursor is a line, not a block. You can change these defaults with statements in *TCMD.INI* or via the Command Line 1 page of the configuration dialogs.

Certain special characters (the command separator, escape character, and parameter character) also vary between 4DOS, 4OS2, 4NT, and Take Command. If you have batch files or aliases which utilize these special characters and you plan to use them under more than one product, or convert them for use under Take Command when you have been using them under another product, see Special Character Compatibility for details on using compatible characters for different JP Software products.

Before using your 4DOS, 4OS2, or 4NT batch files and aliases under Take Command, you should also see the detailed discussion of this topic under Using 4DOS, 4OS2, and 4NT Aliases and Batch Files.

Take Command also offers a range of new Windows-related features which are not available in 4DOS, 4OS2, or 4NT, including:

- A built-in scrollback buffer that lets you look back through the output from past commands.
- A standard Windows menu bar for access to many commonly-used Take Command features.
- A status bar showing memory and resource usage.
- A customizable tool bar that gives you quick access to commands and applications.
- Windows dialogs, accessible from the Options and Utilities menus, for editing environment variables, aliases, file descriptions, and startup parameters (the *TCMD.INI* file).
- Direct access to Program Manager groups through the Applications menu.

» High-speed, dialog-based file and text search (see "Find Files/Text" on the Utilities menu). The new FFIND command gives you the same capabilities at the Take Command prompt.

» Commands like ACTIVATE, MSGBOX, and QUERYBOX that allow you to use Windows features and control Windows applications from your batch files.

» A new technology, called "Caveman," which you can use to run many DOS utilities in the Take Command window (see Take Command and DOS Applications for details).

The Command Line

Take Command 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, how keystrokes are interpreted when you enter them at the command line, and how to transfer text between Take Command and other Windows applications.

The keystrokes discussed here are the ones normally used by Take Command. If you prefer using different keystrokes to perform these functions, you can assign new ones with key mapping directives in the *TCMD.INI* file.

The command line features documented in this section are:

- Command-Line Editing
- Command History and Recall
- Command History Window
- Filename Completion
- Automatic Directory Changes
- Directory History Window
- Multiple Commands
- Command-Line Length Limits
- Scrolling and History Keystrokes

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

Command-Line Editing

The command line works like a single-line word processor, allowing you to edit any part of the command at any time before you press **Enter** to execute it, or **Esc** to erase it. The command line extends to a maximum of 255 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 Keys:

Ins	Toggle between insert and overstrike mode.
Del	Delete the character at the cursor, or the highlighted text.
Bksp	Delete the character to the left of the cursor, or the highlighted text.
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.
Shift-Ins	Insert the text from the clipboard at the current cursor position on the command line.
Ctrl-Shift-Ins	Insert the highlighted text (from anywhere in the window) at the current cursor position on the command line.

Execution:

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

To highlight text on the command line use the mouse, or the **Shift** key in conjunction with the ←, →, **Ctrl-**←, **Ctrl-**→, **Home**, and **End** keys. You can select a complete word by placing the cursor anywhere in the word and double-clicking with the mouse.

Once you have selected or highlighted text on the command line, any new text you type will replace the highlighted text. If you press **Bksp** or **Del** while there is text highlighted on the command line, the highlighted text will be deleted.

While you are working at the Take Command prompt you can also use the Windows clipboard to copy text

between Take Command and other applications (see Highlighting and Copying Text for additional details). You can also use the Windows Drag and Drop facility to paste a filename from another application onto the command line.

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

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

Take Command 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 Ctrl-X (ASCII 24), which is shown in some fonts as an up-arrow []. For example:

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

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

To get around this problem, use the special keystroke **Alt-255**. You enter Alt-255 by holding down the **Alt** key while you type **0255** 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. Also, in Take Command/16 and Take Command/32 the leading **0** before the **255** is required.) This forces Take Command to interpret the next keystroke literally and place 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 *TCMD.INI* file, and Alt-255 itself can be reassigned with the CommandEscape directive.

Command History and Recall

Each time you execute a command, the entire command line is saved in a **command history list**. You can display the saved commands, search the list, modify commands, and rerun commands. The command history is available at the command prompt and in a special command history window.

Command History Keys:

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

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

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

If you prefer to use the arrow keys to access the command history without having to press **Ctrl** (as in 4DOS), see the SwapScrollKeys directive in *TCMD.INI*, or the corresponding option on the Command Line 1 page of the configuration dialogs. SwapScrollKeys switches the keystroke mapping so that the , ↓, and **PgUp** keys manipulate the command history, and **Ctrl-**, **Ctrl-↓**, **Ctrl-PgUp**, and **Ctrl-PgDn** are used to control the scrollback buffer. For more details see Scrolling and History Keystrokes.

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 **Ctrl-**. You only need to enter enough characters to identify the command that you want to find. If you press the **Ctrl-** 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 **Ctrl-** again.

You can specify the size of the command history list with the History directive in the *TCMD.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 or moved to the end of the list when it is re-executed, set HistCopy or HistMove to Yes in the *TCMD.INI* file or select **Copy to End** or **Move to End** on the Command Line 1 page of the configuration dialogs. If you select either of these options, the list entry identified as current (the entry from which commands are retrieved when you press **Ctrl-Up Arrow**) is also adjusted to refer to the end of the list after each command is executed.

Command History Window

You can view the command history in a scrollable **command history window**, and select the command to re-execute or modify from those displayed in the window.

Command History Window Keys:

Ctrl-PgUp or **Ctrl-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 or **Double Click**	Execute the selected line.
Ctrl-Enter or **Ctrl-Double Click**	Move the selected line to the command line for editing.

To activate the command history window press **Ctrl-PgUp** or **Ctrl-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** or double-click with the mouse to execute it immediately. Press **Ctrl-Enter** or hold down the Ctrl key while you double-click with the mouse to move the line to the prompt for editing (you cannot edit the line directly in the history window).

You can view a "filtered" history window by typing some characters on the command line, then pressing **Ctrl-PgUp** or **Ctrl-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 the *TCMD.INI* file, or the corresponding items on the Command Line 2 page of the configuration dialogs. You can also change the keys used in the window with key mapping directives in the *TCMD.INI* file.

If you prefer to use the PgUp key to access the command history without having to press **Ctrl** (as in 4DOS), see the SwapScrollKeys directive in *TCMD.INI*, or the corresponding option on the Command Line 1 page of the configuration dialogs. SwapScrollKeys switches the keystroke mapping so that the , ↓, and **PgUp** keys manipulate the command history, and **Ctrl-**, **Ctrl-**↓, **Ctrl-PgUp**, and **Ctrl-PgDn** are used to control the scrollback buffer. For more details see Scrolling and History Keystrokes.

Filename Completion

Filename completion can help you by filling in a complete file name on the command line when you only remember or want to type part of the name. Filename completion can be used at the command line, which is explained here, and in a filename completion window.

Filename Completion Keys:

F8 or **Shift-Tab**	Get the previous matching filename.
F9 or **Tab**	Get the next matching filename.
Ctrl-Shift-Tab or **F11**	Keep the current matching filename and display the next matching name immediately after the current one.

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. Take Command 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 Take Command 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 **Ctrl-Shift-Tab** or **F11** 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**, Take Command will match all files. For example, if you enter the above command as "COPY ", without the "*.TXT", and then press **Tab**, the first filename in the current directory is displayed. Each time you press **Tab** or **F9** after that, another name from the current directory is displayed, until all filenames have been displayed.

If you type a filename without an extension, Take Command 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.

Several topics are related to filename completion. See

Appending Backslashes to Directory Names
Customizing Filename Completion
Filename Completion Window

Appending Backslashes to Directory Names

If you set the AppendToDir directive the *TCMD.INI* file, or the corresponding option in the configuration dialogs, Take Command will add a trailing backslash [\] to all directory names. This feature can be especially handy if you use filename completion to specify files that are not in the current directory a succession of **Tab** (or **F9**) and **Ctrl-Shift-Tab** keystrokes can build a complete path to the file you want to work with

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

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

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

Customizing Filename Completion

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

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

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

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

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

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

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

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

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

```
c:\> set filecompletion=cd:dirs; winedit:c asm
```

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

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

Filename Completion Window

You can also view filenames in a **filename completion window** and select the file you want to work with. To activate the window, press **F7** or **Ctrl-Tab** at the command line. You will see a window in the upper-right corner of the screen, with a sorted list of files that match any partial filename you have entered on the command line. If you haven't yet entered a file name, the window will contain the name of all files in the current directory. You can search for a name by typing the first few characters; see Popup Windows for details.

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 or **Double Click**	Insert the selected filename into the command line.

Automatic Directory Changes

[Automatic directory changes are part of a set of comprehensive directory navigation features built into Take Command. For a summary of these features, and more information on the Extended Directory Searches and CDPATH features mentioned below, see the Directory Navigation section.]

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

```
c:\> tcmd\
c:\tcmd>
```

This feature can make directory changes very simple when it's combined with Extended Directory Searches or CDPATH. If you have enabled either of those features, Take Command will use them in searching for any directory you change to with an automatic directory change (see Directory Navigation for more information on CDPATH and Extended Directory Searches).

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

```
c:\tcmd> win\
e:\win>
```

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

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

```
c:\> d:\data\finance\
c:\> archives\
c:\> ...\util\win95\
c:\> \\server\vol1\george\
```

The first and last examples change to the named directory. The second changes to the *ARCHIVES* subdirectory of the current directory, and the third changes to the *UTIL\WIN95* subdirectory of the directory which is two levels "up" from the current directory in the tree.

Directory History Window

[The directory history window is part of a set of comprehensive directory navigation features built into Take Command. For a summary of these features, and more information on enhanced directory navigation features, see Directory Navigation.]

Directory History Window Keys:

F6	Open the directory history window.
	Scroll the display up one line.
↓	Scroll the display down one line.
←	Scroll the display left 4 columns.
→	Scroll the display right 4 columns.
PgUp	Scroll the display up one page.
PgDn	Scroll the display down one page.
Ctrl-PgUp or **Home**	Go to the beginning of the directory list.
Ctrl-PgDn or **End**	Go to the end of the directory list.
Ctrl-D	Delete the selected line from the directory list.
Enter or **Double Click**	Change to the selected drive and directory.
Ctrl-Enter or **Ctrl-Double Click**	Move the selected line to the command line for editing.

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

You can view the directory history from the scrollable **directory history window** and change to any drive and directory on the list. To activate the directory history window, press **F6** at the command line. You can then select a new directory with the **Enter** key or by double-clicking with the mouse.

If the directory history list becomes full, old entries are deleted to make room for new ones. You can set the size of the list with the DirHistory directive the *TCMD.INI* file or with the corresponding items on the Startup page of the configuration dialogs. You can change the keys used in the window with key mapping directives in the *TCMD.INI* file.

In order to conserve space, each directory name is recorded just once in the directory history, even if you move into and out of that directory several times.

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

Multiple Commands

You can type several commands on the same command line, separated by a caret **[^]**. 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 255 characters.

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

If you don't like using the default command separator, you can pick another character using the SETDOS /C command, the CommandSep directive in the *TCMD.INI* file or on the Options 1 page of the configuration dialogs. If you plan to share aliases or batch files between Take Command and 4DOS, 4OS2, or 4NT, see the %+ variable and the section on Special Character Compatibility for details about choosing compatible command separators for two or more products.

Expanding and Disabling Aliases

A few command line options are specifically related to aliases, and are documented briefly here for completeness. If you are not familiar with aliases, see Aliases and the ALIAS command for complete detail.

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

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

```
c:\> *dir
```

Command-Line Length Limits

When you first enter a command at the prompt or in an alias or batch file, it can be up to 255 characters long.

As Take Command 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 255 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 511 characters. If your use of aliases or environment variables causes the command line to exceed either of these limits as it is expanded, you will see a "Command line too long" error and the remainder of the line will not be executed.

Scrolling and History Keystrokes

In order to support the scrollback buffer, some Take Command keystrokes are different from what you may be used to in 4DOS, 4OS2, or 4NT. The differences are:

	4DOS	Take Command
Command Line:		
Previous command	**Up** []	**Ctrl-Up**
Next command	**Down** [↓]	**Ctrl-Down**
History window	**PgUp**	**Ctrl-PgUp**
Directory history	**Ctrl-PgUp**	**F6**
Screen Scrollback:		
Up one line	N/A	**Up** []
Down one line	N/A	**Down** [↓]
Up one page	N/A	**PgUp**
Down one page	N/A	**PgDn**

If you prefer to reverse this arrangement and use the arrow and PgUp keys to access the command history without having to press **Ctrl** (as in 4DOS), see the SwapScrollKeys directive in the *TCMD.INI* file, or the corresponding option on the Command Line 1 page of the configuration dialogs. SwapScrollKeys switches the keystroke mapping so that the , ↓, and **PgUp** keys manipulate the command history, and **Ctrl-**, **Ctrl-**↓, **Ctrl-PgUp**, and **Ctrl-PgDn** are used to control the scrollback buffer. (SwapScrollKeys does not affect the use of **F6** for the directory history).

You can also change the way any individual key operates with the corresponding key mapping directive in *TCMD.INI*. The directives associated with the history and scrolling keys are:

NextHistory, PrevHistory

HistWinOpen, DirWinOpen

ScrollUp, ScrollDown, ScrollPgUp, ScrollPgDn

File Selection

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

Most of the features explained in this section apply to Take Command 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

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

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

Wildcards

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

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

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

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

```
c:\> dir *.*
```

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

```
c:\> dir *.txt
```

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

```
c:\> dir st*.d*
```

Take Command also lets you 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 Take Command, 4DOS, 4OS2, and 4NT.

Advanced Wildcards

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

```
c:\> dir letter?.doc
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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. Take Command will examine the each file's size and timestamp (a record of when the file was created or last modified) to determine if the file meets the range criteria that you have specified.

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

A range begins with the switch character (*/*), followed by a left square bracket ("**[**") and a character that specifies the range type: "s" for a size range, "d" for a date range, or "t" for a time range. The "s", "d", or "t" is followed by a start value, and an optional comma and end value. The range ends with a right square bracket ("**]**").

See the individual range types for details on specifying ranges:

Size Ranges

Date Ranges

Time Ranges

Using Ranges

All ranges are inclusive. For example, a size range which selects files from 10,000 to 20,000 bytes long will match files that are exactly 10,000 bytes and 20,000 bytes long, as well as all sizes in between; a date range that selects files last modified between 102797 and 103097 will include files modified on each of those dates, and on the two days in between.

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

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

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

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

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

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

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

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

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

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

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, **/[d12-1-97,12-5-97]** selects files that were last modified between December 1, 1997, and December 5, 1997.

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

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

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

```
Specification    Selects Files

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

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

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

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

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

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

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
```

The separator character used in the time may vary depending upon your country information.

File Exclusion Ranges

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

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

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

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

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

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

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

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

Multiple Filenames

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

```
c:\> copy *.txt *.doc a:
```

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

```
c:\> copy a:\details\file1.txt a:\details\file1.doc c:
```

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

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

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

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

Include Lists

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

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

```
c:\> copy a:\details\file1.txt a:\details\file1.doc c:
```

to this using an include list:

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

Include lists are similar to multiple filenames, but have three important differences. First, you don't have to repeat the path to your files if you use an include list, because all of the included files must be in the same directory.

Second, if you use include lists, you aren't as likely to accidentally overwrite files if you forget a destination path for commands like COPY, because the last name in the list will be part of the include list, and won't be seen as the destination file name. (Include lists can only be used as the *source* parameter the location files are coming from for COPY and other similar commands. They cannot be used to specify a destination for files.)

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

```
c:\> dir *.txt *.doc
```

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

```
c:\> dir *.txt;*.doc
```

will display all the files in one list.

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

The maximum length of an include list is 260 characters.

Executable Extensions

Normally, when you type a filename (as opposed to an alias or internal command name) as the first word on the command line, Take Command 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, *.PIF* to indicate that it contains information on how to execute a program under Windows, *.BTM* or *.BAT* to indicate a batch file, or *.REX* to indicate a file to be run with a REXX interpreter.

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

Windows also includes the ability to associate file extensions with specific applications. See Windows File Associations for details on this feature and its relationship to Take Command executable extensions.

You can use environment variables to define the internal command, external program, batch file, or alias to run for each defined file extension. To create an executable extension for use only in Take Command, 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.

The syntax for creating an executable extension is:

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

This tells Take Command 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.

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, Take Command 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.

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

The next example defines *WRITE.EXE* as the processor for *.TXT* files:

```
c:\> set .txt=c:\windows\write.exe
```

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

```
c:\source> hello
```

Take Command will execute the command:

```
c:\windows\write.exe c:\source\hello.txt
```

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

```
c:\source> hello -w
```

Take Command would execute the command:

```
c:\windows\write.exe c:\source\hello.txt -w
```

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

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

Directory Navigation

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

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

This section begins with a summary of all Take Command directory navigation features. It also provides detailed documentation on the enhanced directory search features: Extended Directory Searches and the CDPATH..

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

Under Windows 3.x, any change you make to the current directory is "global," and may affect other applications. For example, after you change the current directory in Take Command, you may find that the File / Open dialog in your word processor starts in the new Take Command current directory, rather than the directory you last used in the word processor. This behavior is due to the design of Windows and many Windows applications, and is not due to Take Command.

Finding Directories

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

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

Initiating a Directory Change

Take Command supports the traditional methods of changing directories, and also offer several more flexible approaches:

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

CDD, PUSHD, and automatic directory changes can also change to a network drive and directory mapped to a drive letter or specified with a UNC name (see File Systems for details).

Returning to a Previous Directory

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

- The **CD -** and **CDD -** commands can be used to return to the previous working directory (the one you used immediately before the current directory). Use these commands if you are working in two directories and alternating between them.
- The **directory history window** allows you to select one of several recently-used directories from a popup list and return to it immediately. The window displays the contents of the directory history list.
- The **POPD command** will return to the last directory saved by PUSHD. The directory stack holds 511 characters, enough for 20 to 40 typical drive and directory entries.

Extended Directory Searches

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

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

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

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

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

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

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

The Extended Search Database

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

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

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

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

Directories can only be added automatically to *JPSTREE.IDX* if the new entry needs to be placed less than 64K bytes from the end of the file. If a directory cannot be added automatically, an error message appears. Automatic deletions will work from any location in the file.

Enabling Extended Searches

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

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

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

FuzzyCD Setting	CDD Command
0	cdd c:\data\mydir
1	cdd mydir
2	cdd myd
3	cdd yd

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

Forcing an Extended Search with Wildcards

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

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

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

Wildcards can only be used in the final directory name in the path (after the last backslash in the path name). For example you can find *COMM*A*.** (all directories whose parent directory is COMM and

which have an A somewhere in their names), but you cannot find *CO?M*A*.** because it uses a wildcard before the last backslash.

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

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

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

CDPATH

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

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

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

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

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

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

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

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

```
c:\wp\letters\january> cdd d:\software\util
```

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

```
c:\wp\letters\january> cdd util
```

or, using an automatic directory change:

```
c:\wp\letters\january> util\
```

to change to D:\SOFTWARE\UTIL.

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

Other Features

Page and File Prompts

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

When Take Command 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;" **R** for "process the Remainder of the files without further prompting. You can also press **Esc**, **Ctrl-C**, or **Ctrl-Break** at this prompt to cancel the remainder of the command.

If you press **Ctrl-C** or **Ctrl-Break** while a batch file is running, you will see a "Cancel batch job" prompt. For information on responses to this prompt see Interrupting a Batch File

Redirection and Piping

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

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

Redirection and piping allow you to change these assignments temporarily.

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

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

Redirection

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

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

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

Here are the standard redirection options supported by Take Command:

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

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

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

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

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

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

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

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

Advanced Redirection Options

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

When output is appended to a file with **>>**, **>>&**, or **>>&>**, the file will be created if it doesn't already exist.

However, if NoClobber is set as described above, append redirection will not create a new file; instead, if the output file does not exist a "File not found" or similar error will be displayed.

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

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

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

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

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. You may also have 2 or more pipes operating simultaneously if, for example, you have the pipes running in different windows or sessions.

When running Take Command under OS/2, you cannot use a pipe to transmit information to a DOS program. This is a necessary limitation of OS/2's Windows support, not a problem in Take Command/16.

Take Command creates one or two temporary files to hold the output of a pipe. The files are given unique names. By default, these files are stored in the root directory of the boot drive, but you can override this with the **TEMP** environment variable.

If you want to pipe information to a command inside an IFF, use command grouping around the IFF command. If you do not, piping will affect only the **first** command after the IFF.

Keystack

The Keystack overcomes two weaknesses of input redirection: some programs ignore standard input and read the keyboard through the operating system, and input redirection doesn't end until the program or command terminates. You can't, for example, use redirection to send the opening commands to a program and then type the rest of the commands yourself. But the Keystack lets you do exactly that.

The Keystack sends keystrokes to an application program. Once the Keystack is empty, the program will receive the rest of its input from the keyboard. The Keystack is useful when you want a program to take certain actions automatically when it starts. It is most often used in batch files and aliases.

The Keystack is invoked with the KEYSTACK command. To place the letters, digits, and punctuation marks you would normally type for your program into the keystack, enclose them in double quotes:

```
c:\> keystack "myfile"
```

Many other keys can be entered into the Keystack using their names. This example puts the **F1** key followed by the **Enter** key in the keystack:

```
c:\> keystack F1 Enter
```

See Keys and Keynames for details on how key names are entered. See the KEYSTACK command for information on using numeric key values along with or instead of key names, and other details about using the Keystack.

You must start or activate the window for the program that will receive the characters before you place them into the Keystack. See KEYSTACK for additional details; see ACTIVATE for information on activating a specific window.

In Take Command/16, the Keystack will not work under WINOS2 (Windows running under OS/2). It can only be used under "true" Microsoft Windows.

ANSI Support

ANSI control sequences are standardized sequences of text characters which allow you to control colors on the screen, manipulate the cursor, and redefine keys. You may have used ANSI sequences to display text or control the color and appearance of your prompt in DOS or OS/2 character mode.

Take Command includes built-in support for most standard ANSI color and cursor control sequences (key substitutions are not supported). To use Take Command's ANSI support you must enable it from the Display page of the configuration dialogs, with the ANSI directive in the *TCMD.INI* file, or with the SETDOS /A command. You can determine whether ANSI support is enabled with the _ANSI internal variable. For information on the specific ANSI codes supported by Take Command see the ANSI Codes Reference.

Several Take Command features provide simpler ways to accomplish the tasks usually performed with ANSI control sequences. For example, there are commands to set the screen colors, display text in specific colors, and position the cursor. These commands are generally easier to understand and use than the corresponding ANSI control sequences; however, we have included ANSI support in Take Command for situations where it is useful (*e.g.*, in the prompt, or when using the ECHO command).

Critical Errors

Windows 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, Take Command, or your application program may not be able to proceed until the error is resolved.

You will typically see a dialog asking you to choose one of two error handling options:

Cancel Tell the program that the operation failed. This option returns an error code to Take Command or to the application program that was running when the error occurred. Take Command generally stops the current command when an operation fails.

Retry Retry the operation. Choose this option if you have corrected the problem.

Conditional Commands

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

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

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

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

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

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

Command Grouping

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

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

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

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

```
c:\> global (ren *.wx1 *.wxo ^ ren *.tx1 *.txo)
```

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

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

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

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

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

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 places some header lines (including today's date) and directory displays in an output file using redirection. The first ECHO command creates the file using **>**, and the other commands append to the file using **>>**:

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

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

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

The redirection, which appears outside the parentheses, applies to all the commands within the parentheses. Because the redirection is performed only once, the commands will run slightly faster than

if each command was entered separately. The same approach can be used for input redirection and for piping.

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

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

A group of commands in parentheses is like a long command line. The total length of the group may not exceed 511 characters, whether the commands are entered from the prompt, an alias, or a batch file. The limit **includes** the space required to expand aliases and environment variables used within the group. In addition, each line you type at the normal prompt or the **More?** prompt, and each individual command within the line, must meet the usual 255-character line length limits.

Escape Character

Take Command 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 Ctrl-X, which appears in the Take Command window as an up-arrow []. (The appearance of control characters depends on the font you use. In many fonts the Ctrl-X character is displayed as a "block" or other non-descript character, but the Terminal font used by default in Take Command typically does display this character as an up-arrow.)

If you don't like using the default escape character, you can pick another character using the SETDOS /E command, the configuration dialogs, or the EscapeChar directive in the *TCMD.INI* file. If you plan to share aliases or batch files between 4DOS, 4OS2, 4NT and Take Command, the %= variable and the section on Special Character Compatibility for details about choosing compatible escape characters for two or more products.

Ten special characters are recognized when they are preceded by the escape character. The combination of the escape character and one of these characters is translated to a single character, as shown below. These are primarily useful for redirecting codes to the printer. The special characters which can follow the escape character are:

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

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

```
c:\> echo 2 is > 4
```

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

```
c:\> echos feY > prn
```

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

Aliases

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

This section shows you some examples of the power of aliases. See the ALIAS command for complete details about writing your own aliases. You can create aliases either from the command line, as described in this section, or with the Aliases dialog which is available from the Utilities menu.

The simplest type of alias gives a new name to an existing command. For example, you could create a command called **R** (for Root directory) to switch to the root directory this way:

```
c:\> alias r = cd \
```

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

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

```
c:\> alias de = dir /oe /p
```

Aliases can be used to execute sequences of commands as well. The following command creates an alias called MUSIC which saves the current drive and directory, changes to the SOUNDS directory on drive C, runs the program *E:\MUSIC\PLAYER.EXE*, and, when the program terminates, returns to the original drive and directory (enter this on one line):

```
c:\> alias music = `pushd c:\sounds ^ e:\music\player.exe ^ popd`
```

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

When an alias contains multiple commands, the commands are executed one after the other. However, if any of the commands runs an external Windows application (such as the fictitious *PLAYER.EXE* shown above), you must be sure the alias will wait for the application to finish before continuing with the other commands. See Waiting for Applications to Finish for additional details.

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

```
c:\> alias play = e:\music\player.exe
c:\> alias music = `pushd c:\sounds ^ play ^ popd`
```

If you enter MUSIC as a command, Take Command will execute the PUSHD command, detect that the next command (PLAY) is another alias, and execute the program *E:\MUSIC\PLAYER.EXE*, and when the program exits return to the first alias, execute the POPD command, and return to the prompt.

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

```
c:\> alias del = *del /p
```

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

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

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

```
c:\> alias desc = c:\winutil\describe.exe
```

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

```
c:\> alias filedesc = *describe
c:\> alias describe = c:\winutil\describe.exe
```

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

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

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

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

The next example clears the Take Command window whenever you press **Ctrl-F1**:

```
c:\> alias @@Ctrl-F1 = cls
```

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

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

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

```
c:\> alias in = `pushd %1 & %2& & popd`
```

Now if you type:

```
c:\> in c:\sounds play furelise.wav
```

you will change to the *C:\SOUNDS* subdirectory, execute the command PLAY FURELISE.WAV, and then return to the current directory.

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

Your copy of Take Command includes a sample alias file called *ALIASES* which contains several useful aliases and demonstrates many alias techniques. Also, see the ALIAS and UNALIAS commands for more information and examples.

Batch Files

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

The topics included in this section are:

.BAT 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, and the file is opened and closed to read each line. In the second mode the batch file is opened once, the entire file is read into memory, and the file is closed. The second mode can be 5 to 10 times faster, especially if most of the commands in the batch file are internal commands. However, only the first mode can be used for self-modifying batch files (which are rare).

The batch file's extension determines its mode. Files with a *.BAT* 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. A batch file run in BTM mode must be less than 64K (65536) bytes long.

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 *TCMD.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
```

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

Batch File Parameters

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

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

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

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

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

```
cd %1
```

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

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

(See the IF and IFF commands.)

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

Using Environment Variables

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

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

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

Batch File Commands

Several commands are particularly suited to batch file processing. Each command is explained in detail in the **Command Reference**. Here is a list of some of the commands you might find most useful:

ACTIVATE activates another window.

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

CALL executes one batch file from within another.

CANCEL terminates all batch file processing.

CLS and **COLOR** set the Take Command display colors.

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

DRAWBOX draws a box on the screen.

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

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

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

GOTO branches to a different location in the batch file.

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

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

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

KEYSTACK sends keystrokes to applications.

LOADBTM changes the batch file operating mode.

MSGBOX displays a dialog box with standard buttons like Yes, No, OK, and Cancel, and returns the user's selection.

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

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

QUERYBOX displays a dialog box for text input.

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

REM places a remark in a batch file.

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

SCRPUT displays a message in color.

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

SHIFT changes the numbering of the batch file parameters.

START starts another session or window.

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

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

TIMER starts or reads a stopwatch.

TITLE changes the window title.

VSCRPUT displays a vertical message in color.

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

Interrupting a Batch File

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

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

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

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

TCSTART and TCEXIT

Take Command supports two "automatic" batch files, files that run without your intervention, as long as Take Command can find them.

Each time Take Command starts, it looks for an automatic batch file called *TCSTART.BTM* or *TCSTART.BAT*. If the *TCSTART* batch file is not in the same directory as Take Command itself, you should use the configuration dialogs or the TCSTARTPath directive in the *TCMD.INI* file to specify its location. *TCSTART* is optional, so Take Command will not display an error message if it cannot find the file.

TCSTART is a convenient place to change the color or content of the prompt for each session, LOG the start of a session, or execute other special startup or configuration commands. It is also one way to set aliases and environment variables.

With the exception of some initialization switches, the entire startup command line passed to Take Command is available to *TCSTART* via batch file parameters (%1, %2, etc.). This can be useful if you want to see the command line passed to Take Command. For example, to pause if any parameters are passed you could include this command in *TCSTART*:

```
if "%1" != "" pause Starting Take Command with parameters [%$]
```

Whenever an Take Command session ends, it runs a second automatic batch file called *TCEXIT.BTM* or *TCEXIT.BAT*. This file, if you use it, should be in the same directory as your *TCSTART* batch file. Like *TCSTART*, *TCEXIT* 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 Take Command ends, or LOG the end of the session.

Transient Sessions and TCSTART

When you set up the TCSTART file remember that it is executed **every** time Take Command starts, including when running a transient copy of Take Command started with the /C startup option. For example, suppose you set up a desktop object with a command line like this, which starts a transient session:

```
Command:                d:\tcmd201\tcmd.exe /c list myfile.txt
Working Directory:c:\data
```

Normally this command would LIST the file *C:\DATA\MYFILE.TXT*. However, if you have a TCSTART file which changes to a different directory, Take Command will look for *MYFILE.TXT* there not in *C:\DATA*.

This is because when Take Command starts it runs TCSTART before processing the startup command line. If TCSTART changes directories, the startup command line will be executed in the new directory. Similarly, any changes to environment variables or other settings in TCSTART will affect all copies of Take Command, including those used for pipes and transient sessions.

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

```
if %_transient != 0 quit
```

Detecting Take Command

From a batch file, you can determine if Take Command, 4DOS, 4OS2, or 4NT is loaded by testing for the variable function @EVAL, with a test like this:

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

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

Using Aliases in Batch Files

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

```
setlocal
unalias *
alias error `echo. ^ echo ERROR: %& ^ goto dispmenu`
alias fatalerror `echo. ^ echo FATAL ERROR: %& & quit`
alias in `pushd %1 ^ %2& ^ popd`
if not exist setup.btm fatalerror Missing setup file!
call setup.btm
cls
:dispmenu
text


          1. Word Processing
          2. Spreadsheet
          3. Communications
          4. Exit
endtext
echo.
inkey Enter your choice:  %%userchoice
switch %userchoice
case 1
   input Enter the file name:  %%fname
   if not exist fname error File does not exist
   in d:\letters c:\wp60\wp.exe
case 2
   in d:\finance c:\quattro\q.exe
case 3
   in d:\comm c:\comsw\pcplus.exe
case 4
   goto done
default
  error Invalid choice, try again
endswitch
goto dispmenu
:done
endlocal
```

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

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

The following 9 lines print a menu on the screen and then get a keystroke from the user and store the keystroke in an environment variable called *userchoice*. Then the SWITCH command is used to test the user's keystroke and to decide what action to take.

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

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

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

```
dir /b > filelist
```

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

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

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

Debugging Batch Files

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

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

```
c:\> setdos /y1 ^ mybatch.btm
```

If you use the debugger regularly you may want to define a simple alias to invoke it, for example (in Take Command/16, use **%&** rather than **%$**):

```
c:\> alias trace `setdos /y1 ^ %&`
```

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

```
c:\> trace mybatch.btm
```

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

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

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

If you use a "prefix" command like EXCEPT, FOR, GLOBAL, or SELECT, the prefix command is

considered one command, and each command it invokes is another. For example, this command line executes four commands the FOR and three ECHO commands:

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

You cannot use the batch debugger with REXX files. It can only be used with normal Take Command batch files.

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

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

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

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

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

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

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

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

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

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

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

If you're using redirection to see the output, remember that any prompts for input will probably go to the output file and not to the screen. Therefore, you will need to know in advance the sequence of keystrokes required to get through the entire batch file, and enter them by hand or with KEYSTACK.

Batch File String Processing

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

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

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

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

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

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

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

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

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

The IFF command in the third block of lines tests the length of both the first and last names. If either is longer than 8 characters, the batch file displays an error message and ends. (QUERYBOX can limit the length of input text more simply with its **/L** switch. We used a slightly more cumbersome method above in order to demonstrate the use of string functions in batch files.)

Finally, in the last block, the batch file executes the LOGIN program with the appropriate parameters, then uses the ENDLOCAL command to restore the original environment and alias list. At the same time, ENDLOCAL discards the temporary variables that the batch file used (NAME, FIRST, FLEN, etc.).

When you're processing strings, you also need to avoid some common traps. The biggest one is handling special characters.

Suppose you have a batch file with these two commands, which simply accept a string and display it:

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

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

```
echo some > none
```

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

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

```
"some > none"
```

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

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

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

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

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

Batch File Line Continuation

Take Command will combine multiple lines in the batch file into a single line for processing when you include the escape character as the very last character of each line to be combined (except the last). The default escape character is Ctrl-X, which appears in the Take Command window as an up-arrow []. For example:

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

You cannot use this technique to extend a batch file line beyond the normal line length limit of 255 characters.

Batch File Compression

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

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

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.BAT* and save the result as *MYBATCH.BTM*, you can use any of these three commands:

```
[c:\] batcomp mybatch.bat
[c:\] batcomp mybatch.bat mybatch
[c:\] batcomp mybatch.bat 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.

Compressed *.BTM*s must be less than 64K bytes long. You can usually work around this limitation by breaking a very long batch file into two or more smaller files that CALL or chain to each other, and then compiling the shorter files separately.

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

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

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

BATCOMP is a DOS and OS/2 character-mode application designed to run in any environment where our command processors run. Normally it can be run successfully from within Take Command without manually starting a separate DOS session. Each of our command processors includes the same version

of *BATCOMP.EXE*, and a batch file compressed with any copy of BATCOMP can be used with any current JP Software command processor.

If you plan to distribute batch files to users of different platforms, see Special Character Compatibility for important information on the command separator, escape character, and parameter character used in each product.

Using 4DOS, 4OS2, and 4NT Aliases and Batch Files

Take Command and 4DOS aliases are separate and independent; Take Command does not automatically "inherit" aliases from a previously loaded copy of 4DOS, and it cannot pass aliases on to a copy of 4DOS started from the Take Command prompt. However, you can load aliases from your Take Command startup batch file. These can be the same aliases you use in 4DOS, or a set that is just for Take Command.

While many of your 4DOS, 4OS2, or 4NT aliases will work well under Take Command, you'll probably want to create a separate set of Take Command aliases. This will allow you to account for differences in running DOS or other console applications, and to create new aliases that take advantage of Take Command features that are unavailable in our character-mode products.

Certain special characters (the command separator, escape character, and parameter character) also vary between 4DOS, 4OS2, 4NT, and Take Command. If you have aliases or batch files which utilize these special characters and you plan to use them under more than one product, or convert them for use under Take Command when you have been using them under another product, see Special Character Compatibility for details on using compatible characters for different JP Software products.

If you want to write aliases or batch files that are used in both Take Command and in 4DOS, 4OS2, or 4NT, but that behave differently in each environment, use the %_DOS variable to make the distinction. For example, this batch file fragment uses the INPUT command to accept a string if it is run under 4DOS, but uses the Windows-style QUERYBOX if it is run under Take Command/16:

```
iff "%_dos" == "WIN" then
querybox "Enter your name:  " %%name
else
input "Enter your name:  " %%name
endiff
```

Aliases and batch files which simply manipulate files or use other internal commands should work with little or no change under Take Command. However, as a general rule, you should test any batch file developed for 4DOS, 4OS2, 4NT, *COMMAND.COM*, or *CMD.EXE* before assuming it will do exactly what you want under Take Command. Pay particular attention to batch files which run complex sequences of external programs.

If you use aliases or batch files to perform a sequence which mixes internal commands and DOS applications, the sequence may not work the way you expect under Take Command. For example, suppose you have an alias that changes the screen color, starts a DOS application, and then resets the color again. If the DOS application is started in a separate window the color changes will not affect it a contingency you probably didn't have to consider when you wrote the batch file.

Similarly, if you run a sequence of several DOS applications which depend on each others' results (for example, through the use of error levels), they may not run the same way under Take Command that they do under a character-mode command processor. For example, if one DOS application runs in its own window and another runs under Caveman, error levels will not be passed between the applications and your batch file or alias won't run the way you expect.

You may also find that you want to take advantage of some of the new features of Take Command to improve your batch files. For example, the START command offers additional flexibility in starting applications. MSGBOX and QUERYBOX can be used to create Windows-style input prompts, and KEYSTACK and ACTIVATE will help control your Windows applications.

Once you get used to these enhancements and minor differences you'll find that you can use Take

Command to manage your system using the same techniques and features you are already familiar with from your experience with 4DOS, 4OS2, 4NT, *COMMAND.COM*, or *CMD.EXE*.

Special Character Compatibility

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

The default values of each of these characters in each product is shown in the following chart. (In this section, an up-arrow [] is used for the ASCII Ctrl-X character, numeric value 24. The appearance of control characters depends on the font you use. In many fonts the up-arrow character is displayed as a "block" or other non-descript character, but the Terminal font used by default in Take Command typically does display the Ctrl-X character as an up-arrow.):

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

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

1) Select a consistent set of characters from the Options 1 page of the configuration dialogs, or with configuration directives in *TCMD.INI*. For example, to set the Take Command/16 characters to match Take Command/32, use these lines in *TCMD##.INI*:

```
CommandSep = &
EscapeChar = ^
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)
```

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

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

```
c:\> setdos /e^
```

would **not** set the escape character to a caret [**^**] if the standard Take Command/16 special characters were currently in effect. The **^** would be seen as a command separator, and would terminate the

SETDOS command before the escape character was set. To work around this, use the escape character variable **%=** before each setting to ensure that the following character is not treated with any special meaning.

For example, the following sequence of commands in a batch file will always set the special characters correctly to their standard Take Command/32 and Take Command for OS/2 values, no matter what their current setting, and will restore them when the batch file is done:

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

A similar sequence can be used to select the standard Take Command/16 characters, regardless of the current settings:

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

(The symbol represents the Ctrl-X character, ASCII value 24.)

Command Parsing

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

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

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

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

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

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

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

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

Once it has finished with the aliases, Take Command next tries to match the command name with its list of internal commands. If it is unsuccessful, it knows that it will have to search for a batch file or external program to execute your command.

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

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

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

Before it can actually execute your command, Take Command must scan the command tail to see if it includes redirection or piping. If so, the proper internal switches are set to send output to an alternate

device or to a file, instead of to the screen.

Finally, it is time to execute the command. If the command name matches an internal command, Take Command will perform the activities you have requested. Otherwise, Take Command searches for an executable (*.COM* or *.EXE*) file, a batch file, or a file with an executable extension that matches the command name (see the detailed description of this search in Executable Files and File Searches).

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

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

Argument Quoting

As it parses the command line, Take Command looks for the caret [**^**] command separator, conditional commands (**||** or **&&**), white space (spaces, tabs, and commas), percent signs [**%**] which indicate variables or batch file arguments 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 Take Command to pass the string to the batch file as a single argument. The quotes keep characters together and reduce the number of arguments in the line.

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

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

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

```
c:\> set FORVAR=for
```

Now, if you enter the command

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

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

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

But if you enter the command

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

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

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

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

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

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

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

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

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

REXX Support

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

The REXX language is not built into Take Command, and must be obtained separately through add-on REXX software, such as Enterprise Alternatives' Enterprise REXX or Quercus's Personal REXX. (Both packages are available for Windows 3.x, Windows 95, and Windows NT. If you want to learn about or purchase one of these REXX packages, contact JP Software's sales department for more information.)

REXX programs are stored in *.REX* files. When Take Command loads, it asks Windows to locate the Enterprise REXX or Personal REXX libraries. If they are available, Take Command checks to see if you are running a *.REX* file. If so, it passes the file to Enterprise REXX or Personal REXX for processing.

When working with a REXX processor, Take Command automatically handles all input and output for the REXX program, and any standard REXX processor window for input and output is not displayed. If you need to run a REXX program inside your REXX processor's window, and not under Take Command, you should start the REXX processor's executable file explicitly, then load and run the REXX program from there.

The REXX packages described above extend the interface between REXX and the command processor by allowing you to invoke Take Command commands from within a REXX program. When you send a command from a REXX program back to the command processor to be executed (for example, if you execute a DIR command within a REXX script), the REXX software must use the correct "address" for the command processor. Take Command functions as the default command processor for REXX, and therefore handles only the default address of **CMD**.

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

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.

You can create, alter, view, and delete environment variables with the Environment dialog (available from the Utilities menu) as well as with the SET, ESET, and UNSET commands.

The following environment variables have special meanings in Take Command:

CDPATH
CMDLINE
COLORDIR
COMSPEC
FILECOMPLETION
PATH
PROMPT
TEMP

Also see:

COPYCMD
DIRCMD

Take Command also supports two special types of variables:

Internal variables are similar to environment variables, but are stored internally within Take Command, 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 Take Command, the size of the environment is set automatically and expanded as necessary. You do not need to specify the size as you do under 4DOS or some traditional command processors.

Due to the method Take Command uses to start applications, it normally passes the original Windows environment to application programs. As a result, applications will not "see" changes or additions you may have made to the environment from within Take Command. You can alter the default behavior by using the START /E command to start applications. See Passing the Environment to Applications for complete details on how Take Command/16 passes the environment to applications.

The trailing percent sign that was traditionally required for environment variable names is not usually required in Take Command, which accepts 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.

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

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

CDPATH

CDPATH tells Take Command 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 [**;**]. See the section on the CDPATH feature for more information about using this variable.

CMDLINE

CMDLINE is the fully expanded text of the currently executing command line. CMDLINE is set just before invoking any *.COM*, *.EXE*, *.BTM*, or *.BAT* 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. See Color-Coded Directories for a complete description of the format of this variable.

COMSPEC

COMSPEC contains the full path and name of the character-mode command processor. Take Command uses it when starting a character-mode or console session (*e.g.*, from the **Apps** menu).

COPYCMD

COPYCMD is used by some versions of *COMMAND.COM* and *CMD.EXE* to hold default options for the COPY command. Take Command does not support this variable, but you can achieve the same effect with an alias. For example, if you want the COPY command to default to prompting you before overwriting an existing file, you could use this alias:

```
c:\> alias copy = `*copy /r`
```

If you wish to use COPYCMD for compatibility with systems that do not use Take Command, you can define the alias this way:

```
c:\> alias copy = `*copy %copycmd`
```

DIRCMD

DIRCMD is used by some versions of *COMMAND.COM* and *CMD.EXE* to hold default options for the DIR command. Take Command does not support this variable, but you can achieve the same effect with an alias. For example, if you want the DIR command to default to a 2-column display with a vertical sort and a pause at the end of each page, you could use this alias:

```
c:\> alias dir = `*dir /2/p/v`
```

If you wish to use DIRCMD for compatibility with systems that do not use Take Command, you can define the alias this way:

```
c:\> alias dir = `*dir %dircmd`
```

FILECOMPLETION

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

PATH

PATH is a list of directories that Take Command 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.

TEMP

TEMP specifies the directory where Take Command/16 should store temporary pipe files.

Internal Variables

Internal variables are special environment variables built into Take Command 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 Take Command, 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.

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

```
c:\> echo %variablename
```

Variables which return a path or file name will return their result in upper or lower case depending on the value of the SETDOS /U switch or the UpperCase directive in the *TCMD.INI* file.

Some variables return values based on information provided by your operating system. These variables will only return correct information if the operating system provides it. For example, _APMBATT will not return accurate results if your operating system and Advanced Power Management drivers do not provide correct information on battery status to Take Command.

Internal Variable Categories

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

See the discussion after the variable list for some additional information, and examples of how these variables can be used. For a more comprehensive set of examples see the *EXAMPLES.BTM* file which came with Take Command.

Hardware status

_APMAC	Advanced Power Management AC Line status (on-line or off-line)
_APMBATT	APM battery status (high, low, critical, charging, unknown)
_APMLIFE	APM remaining battery life (0 - 100 or unknown)
_CPU	CPU type (386, 486, 586, 686)
_KBHIT	Returns 1 if a keyboard input character is waiting
_MONITOR	Returns the monitor type (mono or color)
_NDP	Coprocessor type (0, 387)
_VIDEO	Returns the video adapter type (i.e., CGA, EGA, or VGA)

Operating system and software status

_ANSI	ANSI status (1 if enabled, 0 if not)
_BOOT	Boot drive letter, without a colon

_CODEPAGE	Current code page number
_COUNTRY	Current country code
_DOS	Operating system type (WIN)
_DOSVER	* Operating system version (6.2, 4.0, etc.)
_DPMI	DPMI version level or 0 if none.
_GDIFREE	Returns the percentage of free GDI resources
_MOUSE	Mouse driver flag (1 if loaded, 0 if not)
_SYSFREE	Returns the percentage of free System resources
_USERFREE	Returns the percentage of free User resources
_WIN	Returns 2 for Windows 3.x in enhanced mode, 3 for real or standard mode.
_WINDIR	Windows directory pathname
_WINSYSDIR	Windows system directory pathname
_WINTITLE	Current window title
_WINVER	Windows version number

Command processor status

_4VER	Take Command version (2.0, etc.)
_BATCH	Batch nesting level
_BATCHLINE	Line number in current batch file.
_BATCHNAME	Full path and filename of current batch file.
_CMDPROC	Command processor name
_DNAME	Name of the description file.
_HLOGFILE	Current history log file name
_LOGFILE	Current log file name
_SHELL	Shell level (0, 1, 2, ...)
_TRANSIENT	* Transient shell flag (0 or 1)

Screen, color, and cursor

_BG	Background color at cursor position
_CI	Current text cursor shape in insert mode
_CO	Current text cursor shape in overstrike mode
_COLUMN	Current cursor column
_COLUMNS	Virtual screen width
_FG	Foreground color at cursor position
_ROW	Current cursor row
_ROWS	Screen height
_SELECTED	First line of highlighted text
_XPIXELS	Physical screen horizontal size in pixels
_YPIXELS	Physical screen vertical size in pixels

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 valid drive (E, F, etc.)

Dates and times

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

Error codes

?	* Exit code, last external program
_?	* Exit code, last internal command
_SYSERR	* Last Windows 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 couple of 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
```

Use the IFF command to check whether there are enough resources free before running an application:

```
iff %_GDIFREE lt 40 then
   echo Not enough GDI resources!
   quit
```

```
else
   d:\mydir\myapp
endiff
```

Call another batch file if today is Monday:

```
if "%_DOW" == "Mon" call c:\cleanup\weekly.bat
```

? 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 Take Command) 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 a caret [**^**] (the default in Take Command) 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 Take Command version (for example, "3.0"). The current decimal character is used to separate the major and minor version numbers (see DecimalChar for details).

_ANSI is "1" if Take Command's ANSI support is enabled, and "0" if it is not. To enable ANSI support use the Display page of the configuration dialogs, the ANSI directive in the *TCMD.INI* file, or the SETDOS /A command See ANSI Support for more information on ANSI support; see the ANSI Codes Reference for information on the specific ANSI codes supported by Take Command.

_APMAC is the Advanced Power Management AC line status ("on-line", "off-line", or "unknown"). An empty string is returned if APM is not installed on your system. See the Glossary for a short description of APM.

_APMBATT is the Advanced Power Management battery status ("high", "low", "critical", "charging", or "unknown"). An empty string is returned if APM is not installed. See the Glossary for a short description of APM.

_APMLIFE is the Advanced Power Management remaining battery life (0 100 or "unknown"). An empty string is returned if APM is not installed. See the Glossary for a short description of APM.

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

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

_BOOT is the boot drive letter, without a colon.

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

_CMDPROC is the name of the current command processor. Each JP Software command processor returns a different value, as follows:

Product	Returns
4DOS	"4DOS"
4OS2	"4OS2"
4NT	"4NT"
Take Command/16	"TCMD"
Take Command/32	"TCMD32"
Take Command for OS/2	"TCMDOS2"

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

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

_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 virtual screen columns (for example, "80"). See Resizing the Take Command Window for additional details on the virtual screen width.

_COUNTRY is the current country code.

_CPU is the CPU type:

386	i386
486	i486
586	Pentium
686	Pentium Pro

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

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

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

_CWPS is the current working directory in the format *pathname*\.

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

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

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

_DNAME is the name of the file used to store file descriptions. It can be changed with the DescriptionName directive in the *TCMD.INI* file, or the SETDOS /D command.

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

	DOS	OS/2	Windows 3.x	Windows 95	Windows 98	Windows NT
4DOS	DOS	DOS	DOS	DOS	DOS	
4OS2		OS2				
4NT				WIN95C	WIN98C	NT
Take Command/16		WIN	WIN			
Take Command/32				WIN95	WIN98	WIN32
Take Command for OS/2		PM				

This variable is useful if you have batch files running in more than one environment, and need to take different actions depending on the underlying operating environment or command processor. If you want the current command processor name, use _CMDPROC. To differentiate between different versions of Windows within Take Command/16 and Take Command/32, use the _WIN variable.

_DOSVER is the current operating system version (for example, "3.10").

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

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

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

_DPMI returns the DPMI version level, or 0 if DPMI isn't present. See the Glossary for a short description of DPMI.

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

_GDIFREE is the percentage of free GDI resources. This information is also displayed on the Take Command status bar.

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

_HOUR is the current hour (0 - 23).

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

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

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

_MINUTE is the current minute (0 - 59).

_MONITOR is the monitor type (mono or color).

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

_MOUSE is "1" if a mouse driver is loaded, and "0" otherwise.

_NDP is the coprocessor type:

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

_PIPE is "1" if the current process is running inside a pipe, and "0" otherwise.

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

_ROWS is the current number of screen rows (for example, "25"). See Resizing the Take Command Window for additional details on screen size.

_SECOND is the current second (0 - 59).

_SELECTED returns the first line of text highlighed in the Take Command window. If no text has been highlighted, SELECTED returns an empty string.

_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.

_SYSFREE is the percentage of free System resources. This information is also displayed on the Take Command status bar.

_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.

_USERFREE is the percentage of free User resources. This information is also displayed on the Take Command status bar.

_VIDEO is the current video adapter type (cga, ega, or vga).

_WIN is the current Microsoft Windows version and mode:

2	Windows 3.x in 386 enhanced mode
3	Windows 3.x in real or standard mode

_WINDIR returns the pathname of the Windows directory.

_WINSYSDIR returns the pathname of the Windows system directory.

_WINTITLE returns the title of the current window.

_WINVER returns the current Windows, Windows 95, or Windows NT version number. The current decimal character is used to separate the major and minor version numbers (see DecimalChar for details).

_XPIXELS returns the physical screen horizontal size in pixels.

_YEAR is the current year (1980 to 2099).

_YPIXELS returns the physical screen vertical size in pixels.

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.

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

The variable functions are useful in aliases and batch files to check on available system resources, manipulate strings and numbers, and work with files and filenames.

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.

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

See the discussion after the function list for some examples. For a much more comprehensive set of examples see the *EXAMPLES.BTM* file which came with Take Command.

The variable functions are:

System status

@DOSMEM[b|k|m]	Size of largest free memory block
@MASTER[varname]	Returns value from the master (DOS) environment
@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
@LABEL[d:]	Volume label
@LPT[n]	Printer ready status (0 or 1)
@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

@ATTRIB[filename,rhsda]	Test or return file attributes

@DESCRIPT[filename]	File description
@FILEAGE[filename]	File age (date and time)
@FILECLOSE[n]	Close a file
@FILEDATE[filename]	File date
@FILEOPEN[filename,mode[,type]]	Open a file
@FILEREAD[n[,length]]	Read next line from a file
@FILES[filename]	Count files matching a wildcard
@FILESEEK[n,offset,start]	Move a file pointer
@FILESEEKL[n,line]	Move a file pointer to a specified line
@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,text]	Write data to a file
@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
@TRUENAME[filename]	Find "true" name for a file
@UNIQUE[d:\path]	Create file with unique name

File names

@EXPAND[filename,nrhsda]	Returns all names that match filename
@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]	Reformat a string
@INDEX[string1,string2]	Position of one string in another
@INSERT[n,string1,string2]	Inserts string1 into string2
@INSTR[start,length,string]	Extract a substring
@LEFT[n,string]	Returns leftmost n characters of string
@LEN[string]	Length of a string
@LOWER[string]	Convert string to lower case

@REPEAT[c,n]	Repeat a character
@REPLACE[string1,string2,text]	Replace string1 with string2 in text
@RIGHT[n,string]	Returns rightmost n characters of string
@STRIP[chars,string]	Strips all chars from string.
@SUBSTR[string,start,length]	Extract a substring
@TRIM[string]	Remove blanks from a string
@UPPER[string]	Convert string to upper case
@WILD[string1,string2]	Compares strings using wildcards
@WORD[["xxx",]n,string]	Extract a word from a string
@WORDS[["xxx"],string]	Count words in a string

Numbers and arithmetic

@COMMA[n]	Insert commas into a number
@CONVERT[input,output,value]	Convert value from input base to output base
@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

@DAY[mm-dd-yy]	Returns day of month for date
@DATE[mm-dd-yy]	Convert date to number of days
@DOW[mm-dd-yy]	Returns day of week for date
@DOWI[mm-dd-yy]	Returns day of week as integer
@DOY[mm-dd-yy]	Returns day of year for date
@MAKEAGE[date[,time]]	Convert date to file timestamp format
@MAKEDATE[n]	Convert number of days to date
@MAKETIME[n]	Convert number of seconds to time
@MONTH[mm-dd-yy]	Returns month for date
@TIME[hh:mm:ss]	Convert time to number of seconds
@YEAR[mm-dd-yy]	Returns year for date

Input dialog boxes

@GETDIR[d:\path]	Prompt for a directory name.
@GETFILE[d:\path\filename[,filter]]	Prompt for a path and file name.

Utility

@ALIAS[name]	Value of an alias

@CLIP[n]	Returns line n from clipboard
@EXEC[command]	Execute a command
@EXECSTR[command]	Execute a command and return the first output line
@IF[condition,true,false]	Evaluates an expression
@INIREAD[filename,section,entry]	Return an entry from an .INI file
@INIWRITE[filename,section,entry,string]	Write an entry in an .INI file
@SELECT[file,t,l,b,r,title]	Menu selection
@TIMER[n]	Get split time from timer.

Examples

You can use variable functions in a wide variety of ways depending on your needs. Here are a couple of examples to give you an idea of what's possible. For a much more comprehensive set of examples, see the file *EXAMPLES.BTM*, which comes with Take Command.

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[%&]`
```

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

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

The attributes (other than **N**) can be combined (for example **%@ATTRIB*[MYFILE*,HS]**). Normally ATTRIB will only return a "1" if **all** of the attributes match. However, if a final **,p** is included (for **p**artial match), then @ATTRIB will return a "1" if **any** of the attributes match. For example, **%@ATTRIB[*MYFILE*,HS,p]** will return a "1" if *MYFILE* has the hidden, system, or both attributes. Without **,p** the function will return a "1" only if *MYFILE* has both attributes.

You can prefix an attribute with "**-**" to mean "this attribute must be off". For example, **%@ATTRIB*[MYFILE*,H-R]** will return a "1" if *MYFILE* has the hidden attribute but not the read-only attribute.

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_" (without the quotes). If the file does not exist, @ATTRIB will return an empty string.

@CDROM[d:]: Returns "1" if the drive is a CD-ROM or "0" otherwise. The drive letter **must** be followed by a colon.

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

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

@COMMA[n]: Returns the number with commas (or the appropriate thousands separator for your current country setting) inserted where appropriate.

@CONVERT[input, output, value]: Converts a numeric string (***value***) from one number base (***input***) to another (***output***). Valid bases range from 2 to 36. The ***value*** must be between 0 and 2**32-1 (4,294,967,295). No error is returned if ***value*** is outside that range. For example, to convert "1010101" from binary to decimal, use this command:

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

@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). The year can be entered as a 4-digit or 2-digit value. Two-digit years between 80 and 99 are interpeted as 1980-1999; values between 00 and 79 are interpreted as 2000 - 2079.

@DAY[mm-dd-yy]: Returns the numeric day of the month for the specified date. @DAY uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be entered as a 4-digit or 2-digit value.

@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. The drive letter **must** be followed by a colon. DOS networks with large server disk drives (over 2 GB) may report disk space values that are too small when @DISKFREE is used. If this occurs, it is because the network software does not report the proper values to Take Command.

The "**b|k|m**" argument specifies the format of the returned value:

- **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.

@DISKTOTAL[d:,b|k|m]: Returns the total disk space on the specified drive. The drive letter **must** be followed by a colon.

The "**b|k|m**" argument specifies the format of the returned value:

- **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.

@DISKUSED[d:,b|k|m]: Returns the amount of disk space in use by files and directories on the specified drive. The drive letter **must** be followed by a colon.

The "**b|k|m**" argument specifies the format of the returned value:

- **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.

@DOSMEM[b|k|m]: Returns the amount of free space in Windows' global heap.

The "**b|k|m**" argument specifies the format of the returned value:

- **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.

@DOW[mm-dd-yy]: Returns the first three characters of the day of the week for the specified date ("Mon", "Tue", "Wed", etc.). @DOW uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be entered as a 4-digit or 2-digit value. Two-digit years between 80 and 99 are interpeted as 1980-1999; values between 00 and 79 are interpreted as 2000 - 2079.

@DOWI[mm-dd-yy]: Returns the day of the week for the specified date as an integer (1 = Sunday, 2 = Monday, etc.). @DOWI uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be entered as a 4-digit or 2-digit value. Two-digit years between 80 and 99 are interpeted as 1980-1999; values between 00 and 79 are interpreted as 2000 - 2079.

@DOY[mm-dd-yy]: Returns the day of year for the specified date (1-366). @DOY uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be entered as a 4-digit or 2-digit value. Two-digit years between 80 and 99 are interpeted as 1980-1999; values between 00 and 79 are interpreted as 2000 - 2079.

@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.

To ensure that your @EVAL expressions are interpreted correctly, spaces should be placed on both sides of each operator, for example:

```
%@eval[(20 %% 3) + 4]
```

The maximum precision is 16 digits to the left of the decimal point and 0 to 8 digits to the right of the decimal point. You can alter the default precision with the configuration dailogs, EvalMax and EvalMin directives in the *TCMD.INI* file and with the SETDOS /F command. You can alter the decimal character with the configuration dialogs, the DecimalChar directive or the SETDOS /G command.

You can alter the precision for a single evaluation with the construct @EVAL[**expression=x.y**]. The ***x*** value specifies the the minimum decimal precision (the minimum number of decimal places displayed); the ***y*** value sets the maximum decimal precision. You can use ***=x,y*** instead of ***=x.y*** if the comma is your decimal separator. If ***x*** is greater than ***y***, it is ignored. You can specify either or both arguments. For example,

```
@eval[3 / 7=.4]         returns 0.4286
@eval[3 / 7=2]          returns 0.42857143
@eval[3 / 6=2.4]        returns 0.50
```

Also see @DEC and @INC.

@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.

@EXECSTR[command]: Runs the specified ***command*** and returns the first line written to STDOUT by that ***command***. @EXECSTR is a "back door" entry into command processing and should be used with extreme caution. Incorrect or recursive use of @EXECSTR may hang your system.

@EXECSTR is useful for retrieving a result from an external utility for example, if you have an external utility called *NETTIME.EXE* which retrieves the time of day from your network server and writes it to standard output, you could save it in an environment variable using a command like this:

```
set server_time=%@execstr[d:\path\nettime.exe]
```

If the same utility returned a result properly formatted for the TIME command you could also use it to set the time on your system:

```
c:\> time %@execstr[d:\path\nettime.exe]
```

@EXPAND[filename[,-nrhsda]]: Returns, on a single line, the names of all files and directories that match the ***filename***, which may contain wildcards and include lists. Returns an empty string if no files match. If the file list is longer than the allowed command line length, it will be truncated without an error message.

The second argument, if included, defines the attributes of the files that will be included in the search. The attributes are:

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

The attributes (other than **N**) can be combined. You can prefix an attribute with "**-**" to mean "this attribute must be off".

If the attribute argument is not used, hidden files, system files, and directories will be excluded from the returned list; all other files which match the **filename** will be included.

@EXT[filename]: Returns the extension from a ***filename***, without a leading period. The extensioin can be up to 3 characters long.

@FILEAGE[filename]: Returns the date and time of the file as a single numeric value. The number can be used to compare the relative ages of two or more files, but can not be used for date and time calculations as it is not returned in identifiable units.

@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.

This function should only be used with file handles returned by @FILEOPEN! If you use it with any other number you may damage other files opened by Take Command (or by the program which started Take Command), or hang your system.

@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. The optional third parameter controls whether the file is opened in binary or text mode. Text mode (the default) should be used to read text using @FILEREAD **without** a "length", and to write text using @FILEWRITE. Binary mode should be used to read binary data with @FILEREAD **with** a "length", and to write binary data with @FILEWRITEB. Returns "1" if the file cannot be opened.

@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.

This function should only be used with file handles returned by @FILEOPEN! If you use it with any other number you may damage other files opened by Take Command (or by the program which started Take Command), or hang your system.

@FILES[filename [,-nrhsda]]: Returns the number of files that match the ***filename***, which may contain wildcards and include lists. Returns "0"if no files match. The ***filename*** must refer to a single directory; to check several directories, use @FILES once for each directory, and add the results together with @EVAL.

The second argument, if included, defines the attributes of the files that will be included in the search. The attributes are:

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

The attributes (other than **N**) can be combined. You can prefix an attribute with "**-**" to mean "this attribute must be off".

@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).

This function should only be used with file handles returned by @FILEOPEN! If you use it with any other number you may damage other files opened by Take Command (or by the program which started Take Command), or hang your system.

@FILESEEKL[n,line]: Moves the file pointer to the specified line in the file whose handle is ***n***. The first line in the file is numbered 0. Returns the new position of the pointer, in bytes from the start of the file. @FILESEEKL must read each line of the file up to the target line in order to position the pointer, and will therefore cause significant delays if used in a long loop or on a large file.

This function should only be used with file handles returned by @FILEOPEN! If you use it with any other number you may damage other files opened by Take Command (or by the program which started Take Command), or hang your system.

@FILESIZE[filename,b|k|m[,a]]: Returns the size of a file, or "1" if the file does not exist. If the ***filename*** includes wildcards or an include list, returns the combined size of all matching files. The optional third argument ***a*** (allocated), if used, instructs @FILESIZE to return the amount of space allocated for the file(s) on the disk, rather than the amount of data in the file. Network drives and compressed drives may not always report allocated sizes accurately, depending on the way the network or disk compression software is implemented.

@FILETIME[filename]: 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.

This function should only be used with file handles returned by @FILEOPEN, or with the handles 1 (for standard output) or 2 (for standard error)! If you use it with any other number you may damage other files opened by Take Command (or by the program which started Take Command), or hang your system.

@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.

This function should only be used with file handles returned by @FILEOPEN! If you use it with any other number you may damage other files opened by Take Command (or by the program which started Take Command), or hang your system.

@FINDFIRST[filename [,[-]nrhsda]]: Returns the name of the first file that matches the filename, which may include wildcards and/or an include list.

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. @FINDFIRST will only find a file if all of the attributes match. You can prefix an attribute with "**-**" to mean "this attribute must be off".

@FINDNEXT[[filename [,[-]nrhsda]]]: Returns the name of the next file that matches the filename(s) in the previous @FINDFIRST call. Returns an empty string when no more files match. @FINDNEXT should only be used after a successful call to @FINDFIRST. You do not need to include the filename parameter, because it must be the same as the previous @FINDFIRST call, unless you also want to specify file attributes for @FINDNEXT.

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. @FINDNEXT will only find a file if all of the attributes match. You can prefix an attribute with "**-**" to mean "this attribute must be off".

@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, **Echo %@format[7,Hello]** displays " Hello" while **Echo %@format[.3,Hello]** displays "Hel".

@FULL[filename]: Returns the full path and filename of a file.

@GETDIR[d:\path]: Pops up a dialog box to select a directory. "**d:\path**" specifies the initial directory; if it is not specified, GETDIR defaults to the current directory. Returns the chosen directory as a string, or an empty string if the user selects "Cancel" or presses Esc.

@GETFILE[d:\path\filename[,filter]]: Pops up a dialog box to select a file. "**d:\path\filename**" specifies the initial directory and filename shown in the dialog, and may include wildcards. If it is not specified, GETFILE defaults to *.* in the current directory. Returns the full path and name of the selected file or an empty string if the user selects "Cancel" or presses Esc. The optional second argument specifies the file extension to use. You can specify multiple extensions by separating them with a semicolon. For example, **%@getfile[c:\windows,*.exe;*.btm]** lets the user select from .EXE and .BTM files only.

@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.

@INIREAD[filename,section,entry]: Returns an entry from the specified .INI file or an empty string if the entry does not exist. For example,

```
%@iniread[c:\tcmd\tcmd.ini,TakeCommand,history]
```

returns the size of the command history if it is specified in *TCMD.INI*. If you don't specify a path for the .INI file, @INIREAD will look in the Windows and Windows\System directories.

@INIWRITE[filename,section,entry,string]: Creates or updates an entry in the specified .INI file. For example,

```
%@iniwrite[c:\tcmd\tcmd.ini,TakeCommand,history,2048]
```

will set the size of the command history to 2,048 bytes the next time Take Command is started. @INIWRITE returns "0" for success or "-1" for failure. If you don't specify a path for the .INI file, @INIWRITE will look in the Windows and Windows\System directories.

@INSERT[n, string1, string2]: Inserts ***string1*** into ***string2*** starting at position ***n***. The first position in the string is postion 0. For example, **%@insert[1,arm,wing]** returns the string "warming."

@INSTR[start, length, string]: Returns a substring, starting at the position ***start*** and continuing for ***length*** characters. If the ***length*** is omitted, it will default to the remainder of the ***string***. If the ***length*** is negative, the start is relative to the right side of the ***string***. The first character in the ***string*** is numbered 0; if the ***length*** is negative, the last character is numbered 0. For example, %@INSTR[0,2,%_TIME] gets the current time and extracts the hour; %@INSTR[1,-2,%_TIME] extracts the seconds. If the ***string*** includes commas, it must be quoted with double quotes [**"**] or back-quotes [`]. The quotes **do** count in calculating the position of the substring. @SUBSTR is an older version of the same function.

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

@LABEL[d:]: Returns the volume label of the specified disk drive. The drive letter **must** be followed by a colon.

@LEFT[n,string]: Returns the leftmost ***n*** characters from the ***string***. If ***n*** is greater than the length of the ***string***, returns the entire ***string***. For example, **%@LEFT[2,jpsoft]** returns the string "jp."

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

@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.

@LINE works with files having lines of no more than 1023 characters; longer lines will not be counted accurately.

The @LINE function must read each line of the file to find the line you request, and will therefore cause significant delays if used in a long loop or on a large file. For a more effective method of processing each line of a file in sequence use the FOR command, or @FILEOPEN and a sequence of @FILEREADs.

You can retrieve input from standard input if you specify CON as the filename. If you are redirecting input to @LINE using this feature, you must use command grouping or the redirection will not work properly (you can pipe to @LINE without a command group; this restriction applies only to input redirection). For example:

```
(echo %@line[con,0]) < myfile.dat
```

@LINES[filename]: Returns the line number of the last line in the file, or "1" if the file is empty. The first line in the file is numbered 0, so (for example) @LINES will return 0 for a file containing one line. To get the actual number of lines, use %@INC[%@LINES[filename]].

@LINES works with files having lines of no more than 1023 characters; longer lines will not be counted accurately.

@LINES must read each line of the file in order to count it, and will therefore cause significant delays if used in a long loop or on a large file.

@LOWER[string]: Returns the ***string*** converted to lower case.

@LPT[n]: Returns a "1" if the specified printer is ready; otherwise, returns "0". n=1 checks the printer connected to LPT1, n=2 checks LPT2, and n=3 checks LPT3. The value returned from @LPT may reflect the status of the printer port, or the printer itself, depending on your BIOS and printer port hardware. You may need to experiment to determine what values are returned on your system.

@MAKEAGE[date[,time]]: Returns the ***date*** and ***time*** (if included) as a single value in the same format as @FILEAGE. @MAKEAGE can be used to compare the time stamp of a file with a specific date and time, for example:

```
if %@fileage[myfile] lt %@makeage[1/1/85] echo OLD!
```

The value returned by @MAKEAGE can be used for comparisons, but can not be used for date and time calculations because it is not in identifiable units.

@MAKEDATE[n]: Returns a date (formatted according to the current country settings). ***n*** is the number of days since 1/1/80. This is the inverse of @DATE.

@MAKETIME[n]: Returns a time (formatted according to the current country settings). ***n*** is the number of seconds since midnight. This is the inverse of @TIME.

@MASTER[varname]: Returns the value of a variable from the master (DOS) environment.

@MONTH[mm-dd-yy]: Returns the month number for the specified date (1-12). @MONTH uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be entered as a 4-digit or 2-digit value.

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

@NUMERIC[string]: Returns "1" if the ***string*** is composed entirely of digits (0 to 9), signs (**+** or **-**), and the thousands and decimals separators. Otherwise, returns "0". If the ***string*** begins with a decimal separator it is not considered numeric unless the next character is a digit, and there are no more decimal separators within the string. For example, ".07" is numeric, but ".a" and ".07.01" are not.

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

@RANDOM[min, max]: Returns a random value between ***min*** and ***max***, inclusive. ***Min***, ***max***, and the returned value are all integers. The random number generator is initialized from the system clock the first time it is used after the command processor starts, so it will produce a different sequence of random numbers each time you use it.

@READSCR[row,col,length]: Returns the text displayed in the Take Command window at the specified location. The upper left corner of the window is location 0,0. The ***row*** and ***column*** can be specified as an offset from the current cursor location by preceding either value with a [**+**] or [**-**]. For example:

```
%@readscr[-2,+2,10]
```

returns 10 characters from the screen, starting 2 rows above and 2 columns to the right of the current cursor position.

You can also specify the row and column as offsets from the current cursor position. Begin the value with a plus sign [**+**] to read the screen the specified number of rows below (or columns to the right of) the current position, or with a minus sign [**-**] to read the screen above (or to the left of) the current position.

@READY[d:]: Returns "1" if the specified drive is ready; otherwise returns "0". The drive letter **must** be followed by a colon.

@REMOTE[d:]: Returns "1" if the specified drive is a remote (network) drive; otherwise returns "0". The drive letter **must** be followed by a colon.

@REMOVABLE[d:]: Returns "1" if the specified drive is removable (*i.e.*, a floppy disk or removable hard disk); otherwise returns "0". The drive letter **must** be followed by a colon.

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

@REPLACE[string1, string2, text]: Replaces all occurrences of ***string1*** in the ***text*** string with ***string2***. For example, **%@replace[w,ch,warming]** returns the string "charming." The search **is** case-sensitive.

@RIGHT[n,string]: Returns the rightmost ***n*** characters from the ***string***. If ***n*** is greater than the length of the ***string***, returns the entire ***string***. For example, **%@right[4,jpsoft]** returns the string "soft."

@SEARCH[filename[,path]]: Searches for the ***filename*** using the PATH environment variable or the specified ***path***, appending an extension if one isn't specified. (See Executable Files and File Searches for details on the default extensions used when searching the PATH, the order in which the search proceeds, and the search of the *\WINDOWS* and *\WINDOWS\SYSTEM* directories.) Returns the fully-expanded name of the file including drive, path, base name, and extension, or an empty string if a matching file is not found. If wildcards are used in the ***filename***, **@SEARCH** will search for the first file that matches the wildcard specification, and return the drive and path for that file plus the wildcard filename (*e.g.*, *E:\UTIL*.COM*).

@SELECT[filename,top,left,bottom,right,title[,1]]: Pops up a selection window with the lines from the specified file, allowing you to display menus or other selection lists from within a batch file. You can move through the selection window with standard popup window navigation keystrokes, including character matching (see Popup Windows for details; to change the navigation keys see the Key Mapping directives in the *TCMD.INI* file). @SELECT returns the text of the line the scrollbar is on if you press **Enter**, or an empty string if you press **Esc**. The file size is limited only by available memory. To select from lines passed through input redirection or a pipe, use CON as the ***filename***. If you use the optional ***1*** argument after the window title, the list will be sorted alphabetically.

@STRIP[chars,string]: Removes the characters in ***chars*** from the ***string*** and returns the result. For example, %@STRIP[AaEe,All Good Men] returns "ll Good Mn". The test is case sensitive. To include a comma in the ***chars*** string, enclose the entire first argument in quotes. @STRIP will remove the quotes before processing the ***string***.

@SUBSTR[string,start,length]: An older version of @INSTR. The ***string*** parameter is at the start of the @SUBSTR argument list, and therefore cannot contain commas (because any commas in the string would be taken as argument separators). @INSTR, which has the ***string*** argument last, does not have this restriction.

@TIME[hh:mm:ss]: Returns the number of seconds since midnight for the specified time. The time must be in 24-hour format; "am" and "pm" cannot be used.

@TIMER[n]: Returns the current split time for a stopwatch started with the TIMER command. The value of **n** specifies the timer to read and can be 1, 2, or 3.

@TRIM[string]: Returns the string with the leading and trailing white space (space and tab characters) removed.

@TRUENAME[filename]: Returns the true, fully-expanded name for a file. TRUENAME will "see through" a JOIN or SUBST. Wildcards may not be used in the filename.

@TRUENAME can handle simple drive substitutions such as those created by JOIN, SUBST, or most network drive mappings. However, it may not be able to correctly determine the true name if you use "nested" JOIN or SUBST commands, or a network which does not report true names properly.

@UNIQUE[d:\path]: Creates a zero-length file with a unique name in the specified directory, and returns the full name and path. If no ***path*** is specified, the file will be created in the current directory. The file name will be FAT-compatible regardless of the type of drive on which the file is created. This function allows you to create a temporary file without overwriting an existing file.

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

@WILD[string1,string2]: Performs a comparison of the two strings, and returns "1" if they match or "0" if they don't match. The second argument, ***string2***, may contain wildcards or extended wildcards; the first argument, ***string1***, may not. The test is not case sensitive. The following example tests whether the \UTIL directory (or any directory that begins with the characters UTIL) is included in the PATH:

```
if %@wild[%path,*\UTIL*] == 1 command
```

@WORD[["xxx",]n,string]: Returns the ***n***th word in a string. The first word is numbered 0. If ***n*** is negative, words are returned from the end of the string. You can use the first argument, "xxx" to specify the separators that you wish to use. If you want to use a double quote as a separator, prefix it with an escape character. If you don't specify a list of separators, @WORD will consider only spaces, tabs, and commas as word 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***. 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. If you don't specify a list of separators, @WORDS will consider only spaces, tabs, and commas as word separators. If the string argument is enclosed in quotation marks, you **must** enter a list of delimiters as well.

@YEAR[mm-dd-yy]: Returns the year for the specified date. @YEAR uses the date format and separators mandated by your country code (for example dd.mm.yy in Germany, or yy-mm-dd in Japan). The year can be specified as two digits or four digits; @YEAR returns the same number of digits included in its argument.

TCMD.INI

Part of the power of Take Command is its flexibility. You can alter its configuration to match your style of computing. Most of the configuration of Take Command is controlled through a file of initialization information called *TCMD.INI*.

This topic contains general information on *TCMD.INI*. 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 description of 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.

We also discuss many ways of configuring Take Command in other parts of the online help.

With **aliases** you can set default options for internal commands and create new commands (see Aliases and the ALIAS command.).

With **executable extensions** you can associate data files with the applications you use to open them.

With the **FILECOMPLETION** environment variable and the FileCompletion .INI directive (explained below) you can customize filename completion to match the command you are working with.

With the **COLORDIR** environment variable and the ColorDir .INI directive you can set the colors used by the DIR command.

With the **SETDOS** command you can change some aspects of Take Command's operation "on the fly."

With **command-line options** you can specify where Take Command looks for its startup files and how it operates for a specific session.

Modifying the *.INI* File

You can create, add to, and modify the *.INI* file in 3 ways: with the configuration dialogs, available on the Options menu or via the OPTION command, and by editing the file with any ASCII editor.

The configuration dialogs allow you to modify the settings that are used most often. When you exit from the dialogs, you can select the **Save** button to save your changes in the *.INI* file for use in the current session and all future sessions, select the **OK** button to use your changes in the current session only, or discard the changes you have made by selecting the **Cancel** button.

When you exit the configuration dialogs, **Save** saves all changes since the last **Save**, or since the last time you started the command processor. If you start the dialogs and exit with **OK**, changes will not be saved in the *.INI* file at that time. However, if you use the dialogs later, and exit with **Save**, any earlier changes will automatically be saved in the *.INI* file along with any new changes from your most recent use of the dialogs.

Changes you make in the **Startup** section of the configuration dialogs will only take effect when you restart Take Command.

The dialogs handle most standard *.INI* file settings. The Advanced directives, the Key Mapping directives, and a few other individual directives noted below do not have corresponding fields in the configuration dialogs, and must be entered manually.

Take Command reads its *.INI* file when it starts, and configures itself accordingly. The *.INI* file is not re-read when you change it manually. For manual changes to take effect, you must restart Take Command. If you edit the *.INI* file manually, make sure you save the file in ASCII format.

Each item that you can include in the *.INI* file has a default value. You only need to include entries in the file for settings that you want to change from their default values.

The *.INI* file has several sections. All of the directives described here go into the **[TakeCommand]** section, which is usually first in the file. You can edit this section manually. Take Command uses the other sections to record information you set while you are using it, including its window size and position, the font you are using, and the buttons you create on the tool bar. You should use Take Command's menu commands to change the settings in these other sections of the *.INI* file instead of editing them directly.

Using the *.INI* File

Some settings in the *.INI* file are initialized when you install Take Command and others (such as window size and position) are modified as you use Take Command, so you will probably have an *.INI* file even if you didn't create one yourself. You should not delete this file.

Take Command searches for the *.INI* file in three places:

- » If there is an "@d:\path\inifile" option on the startup command line, Take Command will use the path and file name specified there, and will not look elsewhere. See the *Introduction and Installation Guide* for details about the startup command line.
- » If there is no *.INI* file name on the startup command line, the search proceeds to the same directory where the Take Command program file (*TCMD.EXE, TCMD32.EXE*, or *TCMDOS2.EXE*) is stored. This is the "normal" location for the *.INI* file. Take Command determines this directory automatically.
- » If the *.INI* file is not found in the directory where the program file is stored, a final check is made in the Windows directory.

.*INI* File Directives

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 1,023 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.

If you use the configuration dialogs to modify the *.INI* file, comments on lines modified from within the dialogs will not be preserved when the new lines are saved. To be sure *.INI* file comments are preserved, put them on separate lines in the file.

When Take Command detects an error while processing the *.INI* file, it displays an error message and prompts you before processing the remainder of 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.

If you need to test different values for an *.INI* directive without repeatedly editing the *.INI* file, use the OPTION command or see the INIQuery directive.

If you want to include the text of one *.INI* file within another (for example, if you have a set of common directives used by several JP Software products), see the Include directive.

The SETDOS command can override several of the *.INI* file directives. For example, the cursor shape used by Take Command 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.

Types of Directives

There are 8 types of directives in the *.INI* file. The different types of directives are shown in the lists below as follows:

Name = nnnn (1234): This directive takes a numeric value which replaces the "nnnn." The default value is shown in parentheses.

Name = c (X): This directive accepts a single character as its value. The default character is shown in parentheses. You must type in the actual character; you cannot use a key name.

Name = CHOICE1 | Choice2 | ... : This directive takes a choice value. The possible choices are listed, separated by vertical bars. The default value is shown in all upper case letters in the directive description, but in your file any of the choices can be entered in upper case or lower case. For example, if the choices were shown as "YES | No" then "YES" is the default.

Name = Color: This directive takes a color specification. See Colors and Color Names for the format of color names.

Name = Key (Default): This directive takes a key specification. See Keys and Keynames for the format of key names.

Name = Path: This directive takes a path specification, but not a filename. The value should include both a drive and path (*e.g.*, *C:\TCMD*) to avoid any possible ambiguities. A trailing backslash [****] at the end of the path name is acceptable but not required. Any default path is described in the text.

Name = File: This directive takes a filename. We recommend that you use a full filename including the drive letter and path to avoid any possible ambiguities. Any default filename is described in the text.

Name = String: This directive takes a string in the format shown. The text describes the default value and any additional requirements for formatting the string correctly. **No comments are allowed**.

Take Command contains a fixed-length area for storing strings entered in the *.INI* file, including file names, paths, and other strings. This area is large and is unlikely to overflow; if it does, you will receive an error message. If this occurs, reduce the complexity of your *.INI* file or contact our technical support department for assistance.

Initialization Directives

The directives in this section control how Take Command starts and where it looks for its files. The initialization directives are:

DirHistory	Size of directory history list
History	Size of history list
IBeamCursor	Select I-beam or arrow cursor
INIQuery	Query for each line in *TCMD.INI*
LoadAssociations	Controls loading of Windows' file associations
ScreenBufSize	Size of screen buffer.
TCStartPath	Path for TCSTART and TCEXIT
TreePath	Path for directory database, JPSTREE.IDX
WindowState	Initial state for the Take Command window
WindowX, WindowY, WindowWidth, WindowHeight	Initial size and position of the Take Command window

DirHistory = nnnn (256): Sets the amount of memory allocated to the directory history list in bytes. The allowable range of values is 256 to 32767.

History = nnnn (1024): Sets the amount of memory allocated to the command history list in bytes. The allowable range of values is 256 to 32767 bytes.

IBeamCursor = YES | No: If set to **Yes**, Take Command will display the standard "I-Beam" cursor in text areas of its window. If IBeamCursor is set to **No**, an arrow is used in all areas of the window (this can be helpful on laptop systems where the I-Beam cursor is hard to see).

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 Take Command 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 dialog displayed when INIQuery = Yes gives you three options:

Yes	Executes the directive
No	Skips the directive
Cancel	Executes the directive and all remaining directives in the [TakeCommand] section of the *.INI* file (*i.e.*, cancels the INIQuery = Yes setting)

+**LoadAssociations** = YES | No: **No** prevents Take Command from loading Windows' direct file associations from the *WIN.INI* file and using them when searching for executable files. The default of **Yes** allows loading of the file associations. See Windows File Associations for additional details.

ScreenBufSize = nnnn (64000): Sets the size of the screen scrollback buffer in bytes. The allowable range is from 16000 to 64000 bytes.

TCStartPath = Path: Sets the drive and directory where the *TCSTART* and *TCEXIT* batch files (if any) are located.

TreePath = Path: Sets the location of JPSTREE.IDX, the file used for the extended directory search database. By default, the file is placed in the root directory of drive C:\.

WindowState = STANDARD | Maximize | Minimize | Custom: Sets the initial state of the Take Command window. **Standard** puts the window in the default position on the Windows desktop, and is the default setting. **Maximize** maximizes the window; **Minimize** minimizes it, and **Custom** sets it to the position specified by the WindowX, WindowY, WindowWidth, WindowHeight directives.

WindowX = nnnn, **WindowY** = nnnn, **WindowWidth** = nnnn, **WindowHeight** = nnnn: These 4 directives set the initial size and position of the Take Command window. The measurements are in pixels or pels. **WindowX and WindowY** refer to the position of the top left corner of the window relative to the top left corner of the screen. These directives will be ignored unless WindowState is set to **Custom**.

Configuration Directives

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

AmPm	Time display format
ANSI	Enables ANSI support
AppendToDir	"\" on directory names in filename completion
BatchEcho	Default batch file echo state
BeepFreq	Default beep frequency
BeepLength	Default beep length
CDDWinLeft, CDDWinTop, CDDWinWidth, CDDWinHeight	Initial size and position of the extended directory search window
CommandSep	Multiple command separator character
CUA	Key set used for cut, copy, and paste
CursorIns	Cursor width in insert mode
CursorOver	Cursor width in overstrike mode
DecimalChar	Select decimal separator for @EVAL, etc..
DescriptionMax	Maximum length of file descriptions
DescriptionName	Name of file to hold file descriptions
Descriptions	Enable / disable description processing
EditMode	Editing mode (insert / overstrike)
Editor	Program to run for "Editor" menu choice
EscapeChar	Select escape character.
EvalMax	Maximum precision returned by @EVAL
EvalMin	Minimum precision returned by @EVAL
ExecWait	Forces Take Command to wait for external programs to complete
EscapeChar	Take Command escape character
FileCompletion	Select files selected for file completion
FuzzyCD	Selects Extended Directory Search mode
HistCopy	Copy recalled commands to end of history
HistLogName	History log file name
HistMin	Minimum command length to save
HistMove	Move recalled commands to end of history
PopupWinLeft, PopupWinTop, PopupWinWidth, PopupWinHeight	Initial size and position of popup windows
HistWrap	Behavior of the command history list
LogName	Log file name
NoClobber	Overwrite protection for output redirection

ParameterChar	Alias / batch file parameter character
ProgmanDDE	Establish communications with Program Manager
PromptShellExit	Control exit prompting when Take Command is the shell
ScreenColumns	Virtual screen width
ScreenRows	Virtual screen height
ScrollLines	Number of lines to scroll up when at the bottom of the window
StatusBarOn	Set status bar mode at startup
StatBarText	Point size of status bar text
SwapScrollKeys	Switch to 4DOS-style history and scrolling keys
TabStops	Sets the tab positions for Take Command's output.
TCMDTaskList	Enables or disables the Take Command task list.
ThousandsChar	Thousands separator for @EVAL, etc.
ToolBarOn	Set toolbar mode at startup
ToolBarText	Point size of toolbar text
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.

ANSI = Yes | NO: Sets the initial state of ANSI support. **Yes** enables ANSI string processing in the Take Command window. The default of **No** disables ANSI strings. See the ANSI Codes Reference for a reference list of the ANSI sequences supported by Take Command. Also see SETDOS /A and the _ANSI internal variable.

AppendToDir = Yes | NO: **Yes** appends a trailing "\" to directory names when doing filename completion. Regardless of the setting of this directive, a trailing backslash is always appended to a directory name at the beginning of the command line to enable automatic directory changes.

BatchEcho = YES | No: Sets the default batch echo mode. **Yes** enables echoing of all batch file commands unless ECHO is explicitly set off in the batch file. **No** disables batch file echoing unless ECHO is explicitly set on. Also see SETDOS /V.

BeepFreq = nnnn (440): Sets the default BEEP command frequency in Hz. This is also the frequency for "error" beeps (for example, if you press an illegal key). To disable all error beeps set this or BeepLength to 0. If you do, the BEEP command will still be operable, but will not produce sound unless you explicitly specify the frequency and duration.

BeepLength = nnnn (2): Sets the default BEEP length in system clock ticks (approximately 1/18 of a second per tick). BeepLength is also the default length for "error" beeps (for example, if you press an illegal key).

CDDWinLeft, **CDDWinTop**, **CDDWinWidth**, **CDDWinHeight** = nnnn: These values set the initial position and size of the popup window used by extended directory searches, in characters, including the border. The defaults are 3, 3, 72, and 16, respectively (*i.e.*, a window beginning in column 3, row 3, 72 columns wide and 16 rows high). The position is relative to the top left corner of the screen. The width and height values include the space required for the window border. The window cannot be smaller than than 10 columns wide by 5 rows high (including the border). The values you enter will be adjusted if necessary to keep a minimum-size window visible on the screen.

CommandSep = c: This is the character used to separate multiple commands on the same line. The default for Take Command is the caret [^]; the default for Take Command/32 is the ampersand [**&**]. You cannot use any of the redirection characters (**| > <**) or any of the whitespace characters (space, tab, comma, or equal sign). The command separator is saved by SETLOCAL and restored by ENDLOCAL. Also see SETDOS /C. See the %+ internal variable and the section on Special Character Compatibility for information on using compatible command separators for two or more products.

CUA = YES | No: With the default setting of "Yes", Take Command will use the Common User Access (CUA) standard keys for cut and paste: Ctrl-Del for cut, Ctrl-Ins for copy, and Shift-Ins for paste. If set to "No," Take Command will use the Windows non-CUA editing keys: Ctrl-X for cut, Ctrl-C for copy, and Ctrl-V for paste. Note that if set to "No," these keys will not be available for their normal usage (Ctrl-X for the Take Command/16 escape character and Ctrl-C for interrupting commands or text-mode applications).

CursorIns = nnnn (15): This is the width 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%. 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. If you set CursorIns and CursorOver to -1, the cursor shape won't be modified at all. If you set them to 0, the cursor will be invisible. Also see SETDOS /S.

CursorOver = nnnn (100): This is the width of the cursor for overstrike mode during command-line editing and all commands which accept line input. The size is a percentage of the total character cell size, between 0% and 100%. For more details see the CursorIns directive; also see SETDOS /S.

DecimalChar = . | , | AUTO: Sets the character used as the decimal separator for @EVAL, numeric IF and IFF tests, version numbers, and other similar uses. The only valid settings are period [**.**], comma [**,**], and **Auto** (the default). A setting of Auto tells the command processor to use the decimal separator associated with your current country code. If you change the decimal character you must also adjust the thousands character (with ThousandsChar) so that the two characters are different. Also see SETDOS /G.

DescriptionMax = nnnn (511): Controls the description length limit for DESCRIBE. The allowable range is 20 to 511 characters.

DescriptionName = File: Sets the file name in which to store file descriptions. The default file name is *DESCRIPT.ION*. **Use this directive with caution** because changing the name from the default will make it difficult to transfer file descriptions to another system. Also see SETDOS /D.

Descriptions = YES | No: Turns description handling on or off during the file processing commands COPY, DEL, MOVE, and REN. If set to **No**, Take Command 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.

Editor = File: Specifies the path and filename of the program that Take Command will execute when you select "Editor" from the Utilities menu. The defaul is *NOTEPAD.EXE*.

EscapeChar = c : Sets the character used to suppress the normal meaning of the following character. The default is Ctrl-X []. See Escape Character for a description of the special escape sequences. You cannot use any of the redirection characters (**|**, **>**, or **<**) or the whitespace characters (space, tab, comma, or equal sign) as the escape character. The escape character is saved by SETLOCAL and restored by ENDLOCAL. Also see SETDOS /E. See the **%=** internal variable and the section on Special Character Compatibility for information on using compatible escape characters for two or more products.

EvalMax = nnnn (8): Controls the maximum number of digits after the decimal point in values returned by @EVAL. You can override this setting with the construct @EVAL[expression=n,n]. The allowable range is 0 to 8. 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. You can override this setting with the construct @EVAL[expression=n,n]. Also see SETDOS **/F**.

ExecWait = Yes | NO: Controls whether Take Command waits for an external program started from the command line to complete before redisplaying the prompt. See See Waiting for Applications to Finish for details on the effects of this directive.

FileCompletion = cmd1: ext1 ext2 ...; cmd2: ext3 ext4 ... Sets the files made available during filename completion for selected commands. The format is the same as that used for the FILECOMPLETION environment variable. See Customizing Filename Completion for a detailed explanation of selective filename completion.

FuzzyCD = 0 | 1 | 2 | 3 Enables or disables extended directory searches, and controls their behavior. A setting of 0 (the default) disables extended searches. For complete details on the meaning of the other settings see Extended Directory Searches.

HistCopy = Yes | NO: Controls what happens when you re-execute a line from the command history. If this option is set to **Yes**, the line is appended to the end of the history list. By default, or if this option is set to **No**, the command is not copied.. The original copy of the command is always retained at its original position in the list, regardless of the setting of HistCopy. Set this option to No if you want to use HistMove = Yes; otherwise, the HistCopy setting will override HistMove.

HistLogName = File: Sets the history log file name and path. If only a path is given, the default log file name (*TCMDHLOG*) will be used. 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 256, which disables all command history saves.

HistMove = Yes | NO: If set to Yes, a recalled line is moved to the end of the command history. The difference between this directive and HistCopy, above, is that HistCopy = Yes copies each recalled line to the end of the history but leaves the original in place. HistMove = Yes places the line at the end of history and removes the original line. This directive has no effect if HistCopy = Yes.

HistWrap = YES | No: Controls whether the command history "wraps" when you reach the top or bottom of the list. The default setting enables wrapping, so the list appears "circular". If HistWrap is set to No, history recall will stop at the beginning and end of the list rather than wrapping.

LogName = File: Sets the log file name and path. If only a path is given, the default log file name (*TCMDLOG*) will be used. Using LogName does not turn logging on; you must use a LOG ON command to do so.

NoClobber = Yes | NO: If set to Yes, will prevent standard output redirection from overwriting an existing file, and will require that the output file already exist for append redirection. Also see SETDOS /N.

ParameterChar = c: Sets the character used after a percent sign to specify all or all remaining command-line arguments in a batch file or alias (*e.g.*, **%&** or **%n&**; see Batch Files and ALIAS). The default is the ampersand [**&**]. The parameter character is saved by SETLOCAL and restored by ENDLOCAL. Also see SETDOS /P. See Special Character Compatibility for information on using compatible parameter characters for two or more products.

PopupWinLeft, **PopupWinTop**, **PopupWinWidth**, **PopupWinHeight** = nnnn: These values set the initial position and size of the command-line and directory history windows, in characters, including the border. The defaults are 40, 1, 36, and 12, respectively (*i.e.*, a window beginning in column 40, row 1, 36 columns wide and 12 rows high). The position is relative to the top left corner of the screen. The width and height values include the space required for the window border. The window cannot be smaller than than 10 columns wide by 5 rows high (including the border). The values you enter will be adjusted if necessary to keep a minimum-size window visible on the screen.

ProgmanDDE = Yes | No | AUTO: Sets the method Take Command uses to retrieve group names and data for the Apps menu. If set to Auto, Take Command first tries to establish DDE communications with Program Manager (or a replacement Windows shell which emulates it). If the DDE link fails, Take Command will try to read the Program Manager's .GRP (group) files directly. If set to Yes, Take Command will use DDE and ignore the .GRP files. If set to No, Take Command will read the .GRP files and not attempt to use DDE communications.

If your Windows shell supports Program Manager DDE and does not keep the Program Manager .GRP files up to date, you may want to set ProgManDDE to Yes to prevent use of the .GRP files.

If your Windows shell does not support Program Manager DDE you may find that Take Command's Apps menu takes a long time to display, because Take Command must wait to see if Program Manager is responding. In this case set ProgManDDE to No and Take Command will read the Program Manager .GRP files immediately, without waiting for a DDE response.

Some Windows shells (for example, HP Dashboard) start Program Manager whenever an application attempts to send a Program Manager DDE message. If you have this type of shell you will see Program Manager start when you try to use Take Command's Apps menu. To avoid this side effect, set ProgManDDE to No.

PromptShellExit = Yes | NO: Controls whether Take Command/16 asks for confirmation when exiting when it is the Windows shell (see Take Command and Windows Shells). The prompt will appear whether you exit Take Command with the EXIT command, by pressing Alt-F4, or with the File/Exit menu option.

ScreenColumns = nnnn: Sets the number of virtual screen columns used by the Take Command window. If the virtual screen width is greater than the physical window width, Take Command will display a horizontal scrollbar at the bottom of the window. See Resizing the Take Command Window for more information on the virtual screen size.

ScreenRows = nnnn (25): Sets the initial height of the Take Command window. See Resizing the Take Command Window for more information on screen size.

ScrollLines = nnnn (2): Sets the number of lines displayed before the screen is physically scrolled. Take Command will scroll up when output reaches the bottom of the window. Higher values will speed up the display but also make it jerky; lower values will make scrolling smoother but will slow it down.

StatusBarOn = YES | No: Yes enables the status bar when Take Command starts. No disables it. The status bar can be enabled or disabled while Take Command is running by using the Options menu. The StatusBarOn setting is automatically updated to reflect the current state of the status bar each time Take Command exits; this preserves the status bar state between sessions.

StatBarText = nnnn (8): Sets the point size of the text on the status bar. The allowable range is 4 to 16.

SwapScrollKeys = Yes | NO: Yes switches to 4DOS-style keystrokes for manipulating the scrollback buffer.

If SwapScrollKeys is set to Yes, the **Up** and **Down** arrow keys will scroll through the command history list and the **PgUp** key will pop up the history window. The **Ctrl-Up**, **Ctrl-Down**, **Ctrl-PgUp**, and **Ctrl-PgDn** keys will scroll the text in the screen buffer.

If SwapScrollKeys is set to No, these keys will assume their default meanings. The **Up** and **Down** arrow keys and the **PgUp** and **PgDn** keys will scroll the text in the screen buffer. The **Ctrl-Up** and **Ctrl-Down** keys will scroll through the command history list and the **Ctrl-PgUp** key will pop up the history window.

For additional details see Scrolling and History Keystrokes.

Do **not** set SwapScrollKeys to Yes if you use key mapping directives to reassign the scrolling or history keys individually. SwapScrollKeys takes effect before other key mappings, and using both methods at the same time will be confusing at best. Setting SwapScrollKeys to Yes has essentially the same effect as including the following key mapping directives in *TCMD.INI* individually:

```
PrevHistory = Up
NextHistory = Down
HistWinOpen = PgUp
HistWinOpen = PgDn
ScrollUp = Ctrl-Up
ScrollDown = Ctrl-Down
ScrollPgUp = Ctrl-PgUp
ScrollPgDn = Ctrl-PgDn
```

TabStops = nnnn (8): Sets the tab stops for Take Command's output (including the output from the LIST and TYPE commands). The allowable range is 1 to 32.

TCMDTaskList = YES | No: If set to No, disables the internal Take Command task list manager called up by Ctrl-Esc within the Take Command window, and uses the Windows Task Manager instead.

ThousandsChar = . | , | AUTO: Sets the character used as the thousands separator for numeric output. The only valid settings are period [**.**], comma [**,**], and **Auto** (the default). A setting of Auto tells the command processor to use the thousands separator associated with your current country code. If you change the thousands character you must also adjust the decimal character (with DecimalChar) so that the two characters are different. Also see SETDOS /G.

ToolBarOn = YES | No: Yes enables the tool bar when Take Command starts. No disables it. The tool bar can be enabled or disabled while Take Command is running by using the Options menu. The ToolBarOn setting is automatically updated to reflect the current state of the tool bar each time Take Command exits; this preserves the tool bar state between sessions.

ToolBarText = nnnn (8): Sets the point size of text on the tool bar. The allowable range is 4 to 16.

UpperCase = Yes | NO: Yes specifies that filenames should be displayed in the traditional upper-case by internal commands like COPY and DIR. No allows the normal Take Command lower-case style. Also see SETDOS /U.

Color Directives

These directives control the colors that Take Command use for its displays. For complete details on color names and numbers, see Colors and Color Names. The color directives are:

ColorDir	Directory colors
InputColors	Input colors
ListColors	Colors used in the LIST display
SelectColors	Colors used in the SELECT display
StdColors	Standard display colors

ColorDir = ext1 ext2 ...:colora;ext3 ext4 ... :colorb; ...: Sets the directory colors used by DIR. 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.

SelectColors = Color: Sets the colors 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 directivel.

StdColors = Color: Sets the standard colors to be used when CLS is used without a color specification. Using this directive is similar to placing a COLOR command in the TCSTART file.

Key Mapping Directives

These directives allow you to change the keys used for command-line editing and other internal functions. They cannot be entered via the configuration dialogs; you must enter them manually (see *TCMD.INI* for details).

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 and Scrollback Buffer Keys
Popup Window Keys
LIST Keys

Using a key mapping directive allows you to assign a different or additional key to perform the function described. For example, to use function key **F3** to invoke the HELP facility (normally invoked with **F1**):

```
Help = F3
```

Any directive can be used multiple times to assign multiple keys to the same function. For example:

```
ListFind = F            ;F does a find in LIST
ListFind = F4           ;F4 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, NormalEditKey, NormalPopupKey, or NormalListKey directive.

General Input Keys

These directives apply to all input. They are in effect whenever Take Command 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.

EndLine = Key (End): Moves the cursor to the end of the line.

EraseLine = Key (Esc): Deletes the entire line.

ExecLine = Key (Enter): Executes or accepts a line.

Ins = Key (Ins): Toggles insert / overstrike mode during line editing.

Left = Key (Left): Moves the cursor left one character on the input line; scrolls the display left 8 columns in LIST; scrolls the display left 4 columns in the command-line, directory history, or %@SELECT window.

NormalKey = Key: Deassigns a general input key in order to disable the usual meaning of the key within Take Command and/or make it available for keystroke aliases. This will make the keystroke operate as a "normal" key with no special function. For example:

```
NormalKey = Ctrl-End
```

will disable Ctrl-End, which is the standard "delete to end of line" key. Ctrl-End could then be assigned to a keystroke alias. Another key could be assigned the "delete to end of line" function with the DelToEnd directive.

Right = Key (Right): Moves the cursor right one character on the input line; scrolls the display right 8 columns in LIST; scrolls the display right 4 columns in the command-line history, directory history, or %@SELECT window.

Up = Key (Up): Scrolls the display up one line in LIST; moves the cursor up one line in SELECT and in the command-line history, directory history, or %@SELECT window.

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 Take Command prompt. The command-line editing keys are:

AddFile	Keeps filename completion entry and adds another
AliasExpand	Expands aliases on command line
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
LineToEnd	Copies a line to the end of the history, then executes it.
NextFile	Gets the next matching filename
NextHistory	Recalls the next command from the history
NormalEditKey	Deassigns a command-line editing key
PopFile	Opens the filename completion window
PrevFile	Gets the previous matching filename
PrevHistory	Recalls the previous command from the history
SaveHistory	Saves the command line without executing it

AddFile = Key (Ctrl-Shift-Tab): Keeps the current filename completion entry and inserts the next matching name.

AliasExpand = Key (Ctrl-F): Expands all aliases in the current command line without executing them.

CommandEscape = Key (Alt-0255): Allows direct entry of a keystroke that would normally be handled by the command line editor (*e.g.* **Tab** or **Ctrl-D**).

DelHistory = Key (Ctrl-D): Deletes the displayed history list entry and displays the previous entry.

EndHistory = Key (Ctrl-E): Displays the last entry in the history list.

Help = Key (F1): Invokes the HELP facility.

LFNToggle = Key (Ctrl-A): Toggles filename completion between long filename and short filename modes on LFN drives.

LineToEnd = Key (Ctrl-Enter): Copies the current command line to the end of the history list, then executes it.

NextFile = Key (F9, Tab): Gets the next matching filename during filename completion. See ClearKeyMap if you need to remove the default mapping of **Tab** to this function.

NextHistory = Key (Ctrl-Down): Recalls the next command from the command history. Also see Scrolling and History Keystrokes and the SwapScrollKeys directive.

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. This will make the keystroke operate as a "normal" key with no special function. See NormalKey for an example.

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 (Ctrl-Up): Recalls the previous command from the command history. Also see Scrolling and History Keystrokes and the SwapScrollKeys directive.

SaveHistory = Key (Ctrl-K): Saves the command line in the command history list without executing it.

Scrollback Buffer Keys

The following keys are also part of the command line editing group. They are used to manipulate the scrollback buffer rather than to edit commands. For additional information see Scrolling and History Keystrokes and the SwapScrollKeys directive.

ScrollUp	Scroll the buffer up one line.
ScrollDown	Scroll the buffer down one line.
ScrollPgUp	Scroll the buffer up one page.
ScrollPgDn	Scroll the buffer down one page.

ScrollUp = Key: Scrolls the Take Command scrollback buffer up one line.

ScrollDown = Key: Scrolls the Take Command scrollback buffer down one line.

ScrollPgUp = Key: Scrolls the Take Command scrollback buffer up one page.

ScrollPgDn = Key: Scrolls the Take Command scrollback buffer down one page.

Popup Window Keys

The following directives apply to popup windows, including the command history window, the directory history window, the filename completion window, the extended directory search window, and the @SELECT window.

DirWinOpen	Opens the directory history window
HistWinOpen	Opens the command history window
NormalPopupKey	Deassigns a popup window key
PopupWinBegin	Moves to the first line of the popup window
PopupWinDel	Deletes a line from within the popup window
PopupWinEdit	Moves a line from the popup window to the prompt
PopupWinEnd	Moves to the last line of the popup window
PopupWinExec	Selects the current item and closes the popup window

DirWinOpen = Key (F6): Opens the directory history window while at the command line. Also see Scrolling and History Keystrokes.

HistWinOpen = Key (Ctrl-PgUp): Brings up the history window while at the command line. Also see Scrolling and History Keystrokes and the SwapScrollKeys directive.

NormalPopupKey = Key: Deassigns a popup window key in order to disable the usual meaning of the key within the popup window. This will make the keystroke operate as a "normal" key with no special function. See NormalKey for an example.

PopupWinBegin = Key (Ctrl-PgUp): Moves to the first item in the list when in the popup window.

PopupWinDel = Key (Ctrl-D): Deletes a line from within the command history or directory history window.

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

PopupWinEnd = Key (Ctrl-PgDn): Moves to the last item in the list when in the popup window.

PopupWinExec = Key (Enter): Selects the current item and closes the window.

LIST Keys

These directives are effective only inside the LIST command.

ListExit	Exits the current file
ListFind	Prompts and searches for a string
ListFindReverse	Prompts and searches backwards.
ListHex	Toggles hexadecimal display mode
ListHighBit	Toggles LIST's "strip high bit" option
ListInfo	Displays information about the current file
ListNext	Finds the next matching string
ListPrevious	Finds the previous matching string.
ListPrint	Prints the file on LPT1
ListWrap	Toggles LIST's wrap option
NormalListKey	Deassigns a LIST key

ListExit = Key (Esc): Exits from the LIST command.

ListFind = Key (F): Prompts and searches for a string.

ListFindReverse = Key (Ctrl-F): Prompts and searches backward for a string.

ListHex = Key (X): Toggles hexadecimal display mode.

ListHighBit = Key (H): Toggles LIST's "strip high bit" option, which can aid in displaying files from certain word processors.

ListInfo = Key (I): Displays information about the current file.

ListNext = Key (N): Finds the next matching string.

ListPrevious = Key (Ctrl-N): Finds the previous matching string.

ListPrint = Key (P): Prints the file on LPT1.

ListWrap = Key (W): Toggles LIST's wrap option on and off. The wrap option wraps text at the right margin.

NormalListKey = Key: Deassigns a LIST key in order to disable the usual meaning of the key within LIST. This will make the keystroke operate as a "normal" key with no special function. See NormalKey for an example.

Advanced Directives

These directives are generally used for unusual circumstances, or for diagnosing problems. Most often they are not needed in normal use. They cannot be entered via the configuration dialogs; you must enter them manually (see *TCMD.INI* for details).

ClearKeyMap	Clear default key mappings
Debug	Set debugging options
Include	Include text from another file in the current *.INI* file

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. ClearKeyMap should appear before any other key mapping directives. If you want to clear some but not all of the default mappings, use ClearKeyMap, then recreate the mappings you want to retain (*e.g.*, with "NextFile=Tab", etc.).

Debug = nnnn (0): Controls certain debugging options which can assist you in tracking down unusual problems. Use the following values for **Debug**; to select more than one option, add the values together:

1 During the startup process, display the complete command tail passed to Take Command, then wait for a keystroke.

2 Include the product name with each error message displayed by Take Command. This may be useful if you are unsure of the origin of a particular error message.

Also see the batch file debugger, a separate and unrelated facility for stepping through batch files.

Include = File: Include the text from the named file at this point in the processing of the current *.INI* file. Use this option to share a file of directives between several JP Software products. The text in the named file is processed just as if it were part of the original *.INI* file. When the include file is finished, processing resumes at the point where it left off in the original file. The included file may contain any valid directive for the current section, but may not contain a section name. Includes may be nested up to three levels deep (counting the original file as level 1). You must maintain include files manually the configuration dialogs modify the original *.INI* file only, and do not update included files.

Commands By Name

?	DRAWVLINE	LOADBTM	SETDOS
ACTIVATE	ECHO	LOG	SETLOCAL
ALIAS	ECHOERR	MD / MKDIR	SHIFT
ATTRIB	ECHOS	MEMORY	START
BEEP	ECHOSERR	MOVE	SWITCH
BREAK	ENDLOCAL	MSGBOX	TEE
CALL	ESET	ON	TEXT
CANCEL	EXCEPT	OPTION	TIME
CD / CHDIR	EXIT	PATH	TIMER
CDD	FFIND	PAUSE	TITLE
CLS	FOR	POPD	TOUCH
COLOR	FREE	PROMPT	TREE
COPY	GLOBAL	PUSHD	TRUENAME
DATE	GOSUB	QUERYBOX	TYPE
DDEEXEC	GOTO	QUIT	UNALIAS
DEL / ERASE	HELP	RD / RMDIR	UNSET
DELAY	HISTORY	REBOOT	VER
DESCRIBE	IF	REM	VERIFY
DIR	IFF	REN / RENAME	VOL
DIRHISTORY	INKEY	RETURN	VSCRPUT
DIRS	INPUT	SCREEN	WINDOW
DO	KEYBD	SCRPUT	Y
DRAWBOX	KEYSTACK	SELECT	
DRAWHLINE	LIST	SET	

Commands By Category

The best way to learn about 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

BREAK	FREE	OPTION	VER
CLS	HISTORY	PROMPT	VERIFY
COLOR	KEYBD	REBOOT	VOL
DATE	LOG	SETDOS	
DIRHISTORY	MEMORY	TIME	

File and directory management:

ATTRIB	FFIND	SELECT	TYPE
COPY	LIST	TOUCH	
DEL / ERASE	MOVE	TRUENAME	
DESCRIBE	REN / RENAME	TREE	

Subdirectory management:

CD / CHDIR	MD / MKDIR
CDD	POPD

DIR
DIRS
PUSHD
RD / RMDIR

Input and output:

DRAWBOX
DRAWHLINE
DRAWVLINE
ECHO
ECHOERR
ECHOS
ECHOSERR
INKEY
INPUT
KEYSTACK
MSGBOX
QUERYBOX
SCREEN
SCRPUT
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
ON
PAUSE
QUIT
REM
RETURN
SETLOCAL
SHIFT
SWITCH
TEXT
UNALIAS

Environment and path commands:

ESET
PATH
SET
UNSET

Window management commands:

ACTIVATE
TITLE
WINDOW

Other commands:

?
DDEEXEC
EXCEPT
EXIT
HELP
START
TEE
TIMER
Y

?

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

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

Usage

? has two functions.

When you use the **?** command by itself, it displays a list of internal commands. For help with any individual command, see the HELP command

If you have disabled a command with SETDOS /I, it will not appear in the list.

The second function of **?** is to prompt the user before executing a specific line in a batch file. If you add a ***prompt*** and a ***command***, **?** will display the prompt followed by "(Y/N)?" and wait for the user's

response. If the user presses "Y" or "y", the line will be executed. If the user presses "N" or "n", the line will be ignored.

```
c:\> ? "Load the network" call netstart.btm
```

When this command is executed, you will see the following prompt; if you answer "Y", the CALL command will be executed:

```
Load the network (Y/N)?
```

ACTIVATE

Purpose: Activate a window, set its state, or change its title.

Format: **ACTIVATE "window" [MAX | MIN | RESTORE | CLOSE | "title"]**

window: Current title of window to work with.
title: New title for window.

See also: START, TITLE, and WINDOW.

Usage

Both the current name of the ***window*** and the new name, if any, must be enclosed in double quotes. The quotes will not appear as part of the title bar text.

If no options are used, the ***window*** named in the command will become the active window and be able to receive keystrokes and mouse commands.

The MAX option expands the window to its maximum size, the MIN option reduces the window to an icon, and the RESTORE option returns the window to its default size and location on the desktop. The CLOSE option closes the window and ends the session running in the window.

This example maximizes and then renames the window called "Take Command":

```
c:\> activate "Take Command" max
c:\> activate "Take Command" "Take Command is Great!"
```

You can use wildcards, including extended wildcards, in the **window** name if you only know the first part of the title. This is useful with applications that change their window title to reflect the file currently in use.

ACTIVATE is often used before KEYSTACK to make sure the proper window receives the keystrokes.

ACTIVATE works by sending the appropriate messages to the named ***window***. If the window ignores or misinterprets the messages, ACTIVATE may not have the effect you want.

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, to create a single-letter command D to display a wide directory, instead of using the longer DIR /W, you could use the command:

```
c:\> alias d = dir /w
```

Now when you type a single **d** as a command, it will be translated into a DIR /W command.

You can also define or modify aliases with the Alias dialog. The dialog allows you to enter the alias ***name*** and ***value*** into separate fields in a dialog box, rather than using the ALIAS command. All of the information in this section also applies to aliases defined via the dialog, unless otherwise noted.

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 Microsoft Word for Windows and had the *C:\WINWORD* directory in your path, you could define the following alias:

```
c:\> alias ww = c:\winword\winword.exe
```

With this alias defined, you can probably remove *C:\WINWORD* from your path. Word for Windows will now load more quickly than it would if Take Command 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 ^ tedit`
```

This alias 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 caret [**^**] 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 different versions of Take Command, and between Take Command and 4DOS, 4OS2, and 4NT. In the text and examples below, we use the Take Command/16 characters. If you want to use the same aliases under different command processors, see Special Character Compatibility.

When an alias contains multiple commands, the commands are executed one after the other. However, if any of the commands run an external Windows application, you must be sure the alias will wait for the application to finish before continuing with the other commands. This behavior is controlled by the **Wait for completion** setting on the Options 2 page of the configuration dialogs or the ExecWait directive in the *TCMD.INI* file.

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 Word for Windows in the *E:\WINWORD* 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:\winword\winword.exe
[c:\] alias w = `pushd c:\wp ^ wp ^ popd`
```

The second alias above could have included the full path and name of *WINWORD.EXE* 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 *WINWORD.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 Take Command not to attempt to interpret that command as another (nested) alias. An asterisk used this way must be preceded by a space or the command separator and followed immediately by an internal or external command name.

By using an asterisk, you can redefine the default options for any internal or external command. For example, suppose that you always want to use the DIR command with the **/2** (two column) and **/P** (pause at the end of each page) options:

```
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 Take Command 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. The asterisk forces interpretation of the second DIR as a command, not an alias.

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 *DESCRIBE.EXE*. Normally, the internal DESCRIBE command will run anytime you type DESCRIBE. But two simple aliases will give you access to both the *DESCRIBE.EXE* program and the DESCRIBE command:

```
c:\> alias describe = c:\winutil\describe.exe
c:\> alias filedesc = *describe
```

The first line above defines DESCRIBE as an alias for the *DESCRIBE.EXE* program. If you stopped there, the external program would run every time you typed DESCRIBE and you would not have easy access to the internal DESCRIBE command. The second line renames the internal DESCRIBE command as FILEDESC. The asterisk is needed in the second command to indicate that the following word means the internal command DESCRIBE, not the DESCRIBE alias which runs your external program.

Another way to understand the asterisk is to remember that a command is always checked for an alias first, then for an internal or external command, or a batch file. The asterisk at the beginning of a command name simply skips over the usual check for aliases when processing that command, and allows Take Command to go straight to checking for an internal command, external command, or batch file.

You can also use an asterisk before a command that you enter at the command line or in a batch file. If you do, that command won't be interpreted as an alias. This can be useful when you want to be sure you are running the true, original command and not an alias with the same name, or temporarily defeat the purpose of an alias which changes the meaning or behavior of a command. For example, above we defined an alias for DIR which made directories display in 2-column paged mode by default. If you wanted to see a directory display in the normal single-column, non-paged mode, you could enter the command *DIR and the alias would be ignored during that one command.

You can also disable aliases temporarily with the SETDOS /X command.

Partial Alias Names

You can also use an asterisk in the *name* of an alias. When you do, the characters following the asterisk are optional when you invoke the alias command. (Use of an asterisk in the alias *name* is unrelated to the use of an asterisk in the alias *value* discussed above.) For example, with this alias:

```
c:\> alias wher*eis = dir /sp
```

the new command, WHEREIS, can be invoked as WHER, WHERE, WHEREI, or WHEREIS. Now if you type:

```
c:\> where myfile.txt
```

The WHEREIS alias will be expanded to the command:

```
dir /sp myfile.txt
```

Keystroke Aliases

If you want to assign an alias to a keystroke, use the keyname on the left side of the equal sign, preceded by an at sign [**@**]. For example, to assign the command DIR /W to the **F4** key, type

```
c:\> alias @F4 = dir /w
```

See Keys and Key Names for a complete listing of key names and a description of the key name format.

Windows always interprets **Alt** plus a key as a menu accelerator key. Such keystrokes are not passed to Take Command, and therefore cannot be used for keystroke aliases.

When you define keystroke aliases, the assignments will only be in effect at the command line, not inside application programs. Be careful not to assign aliases to keys that are already used at the command line (like **F1** for Help). The command-line meanings take precedence and the keystroke alias will never be invoked. If you want to use one of the command-line keys for an alias instead of its normal meaning, you must first disable its regular use with the NormalKey or NormalEditKey directives in the *TCMD.INI* file.

If you define a keystroke alias with a single at sign as shown above, then, when you press the **F4** 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 **F4** 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 [**@@**]. Take Command will execute the alias "silently," without displaying its text on the command line. For example, this command will assign an alias to the **F11** key that uses the CDD command to take you back to the previous default directory:

```
c:\> alias @@f11 = 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 and Scan 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. For example, 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 Take Command 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.

You can also save and reload your aliases using the Aliases dialog. The Export button in the dialog box is equivalent to the ALIAS > filename command shown above, and the Import button is equivalent to the ALIAS /R command.

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. Take Command interprets it to mean "the entire command line, from argument **n** to the end." If **n** is not specified, it has a default value of 1, so **%&** means "the entire command line after the alias name." The special parameter **%#** contains the number of command-line arguments.

For example, the following alias will change directories, perform a command, and return to the original directory:

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

When this alias is invoked as:

```
c:\> in c:\comm mycomm /zmodem /56K
```

the first parameter, **%1**, has the value *c:\comm*. **%2** is *mycomm*, **3** is */zmodem*, and **%4** is */56K*. The command line expands into these three separate commands:

```
pushd c:\comm
mycomm /zmodem /56K
popd
```

This next example uses the IFF command to redefine the defaults for SET. It should be entered on one line:

```
c:\> alias set = `iff %# == 0 then ^ *set /p ^ else ^ *set %& ^ endiff`
```

This modifies the SET command so that if SET is entered with no arguments, it is replaced by SET /P (pause after displaying each page), but if SET is followed by an argument, it behaves normally. Note the use of asterisks (***set**) to prevent alias loops.

If an alias uses parameters, command-line arguments will be deleted up to and including the highest referenced argument. For example, if an alias refers only to **%1** and **%4**, then the first and fourth arguments will be used, the second and third arguments will be discarded, and any additional arguments beyond the fourth will be appended to the expanded command (after the *value* portion of the alias). If an alias uses no parameters, all of the command- line arguments will be appended to the expanded

command.

Aliases also have full access to all variables in the environment, internal variables, and variable functions. For example, you can create a simple command-line calculator this way:

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

Now, if you enter:

```
c:\> calc 5 * 6
```

the alias will display:

```
The answer is: 30
```

Expanding Aliases at the Prompt

You can expand an alias on the command line and view or edit the results by pressing **Ctrl-F** after typing the alias name, but before the command is executed. This replaces the alias with its contents, and substitutes values for each alias paramter, just as if you had pressed the **Enter** key. However, the command is not executed; it is simply redisplayed on the command line for additional editing.

Ctrl-F is especially useful when you are developing and debugging a complex alias, or if you want to make sure that an alias that you may have forgotten won't change the effect of your command.

The UNKNOWN_CMD Alias

If you create an alias with the name **UNKNOWN_CMD**, it will be executed any time Take Command would normally issue an "Unknown command" error message. This allows you to define your own handler for unknown commands. When the **UNKNOWN_CMD** alias is executed, the command line which generated the error is passed to the alias for possible processing. For example, to display the command that caused the error:

```
alias unknown_cmd `echo Error in command "%&"`
```

If the **UNKNOWN_CMD** alias contains an unknown command, it will call itself repeatedly. If this occurs, Take Command will loop up to 10 times, then display an "UNKNOWN_CMD loop" error.

Options

/P (Pause) This option is only effective when ALIAS is used to display existing definitions. It pauses the display after each page and waits for a keystroke before continuing (see Page and File Prompts).

/R (Read file) This option loads an alias list from a file. The format of the file is the same as that of the ALIAS display:

```
name=value
```

where **name** is the *name* of the alias and **value** is its *value*. You can use an equal sign [**=**] or space to separate the name and value. Back quotes are not required around the value. You can add comments to the file by starting each comment line with a colon [**:**]. You can load multiple files with one ALIAS /R command by placing the names on the command line, separated by spaces:

```
c:\> alias /r alias1.lst alias2.lst
```

Each definition in an ALIAS /R file can be up to 511 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 /E /P /Q /S] [+|-[AHRS]] *files* ...**

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

/A: (Attribute select)	**/P**(ause)
/D(irectories)	**/Q**(uiet)
/E (No error messages)	**/S**(ubdirectories)

Attribute flags:

+A	Set the archive attribute
-A	Clear the archive attribute
+H	Set the hidden attribute
-H	Clear the hidden attribute
+R	Set the read-only attribute
-R	Clear the read-only attribute
+S	Set the system attribute
-S	Clear the system attribute

File Selection

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

Usage

Every file and subdirectory has 4 attributes that can be turned on (set) or turned off (cleared): **Archive**, **Hidden**, **Read- only**, and **System**.

The ATTRIB command lets you view, set, or clear attributes for any file, group of files, or subdirectory.

You can view file attributes by entering ATTRIB without specifying new attributes (*i.e.*, without the **[+|-[AHRS]]** part of the format), or with the **DIR /T** command.

The primary use of ATTRIB is to set attributes. For example, you can set the read-only and hidden attributes for the file *MEMO*:

```
c:\> attrib +rh memo
```

Attribute options apply to the file(s) that follow the options on the ATTRIB command line. The example below shows how to set different attributes on different files with a single command. It sets the archive attribute for all *.TXT* files, then sets the system attribute and clears the archive attribute for *TEST.COM*:

```
c:\> attrib +a *.txt +s -a test.com
```

To change directory attributes, use the **/D** switch. If you give ATTRIB a directory name instead of a file name, and omit **/D**, it will append "*.*" to the end of the name and act on all files in that directory, rather than acting on the directory itself.

Your operating system also supports "D" (subdirectory) and "V" (volume label) attributes. These attributes cannot be altered with ATTRIB; they are designed to be controlled only by the operating system itself.

ATTRIB will ignore underlines in the new attribute (the **[+|-[AHRS]]** part of the command). For example, ATTRIB sees these 2 commands as identical:

```
c:\> attrib +a filename
c:\> attrib +__A_ filename
```

This allows you to use a string of attributes from either the @ATTRIB variable function or from ATTRIB itself (both of which use underscores to represent attributes that are not set) and send that string back to ATTRIB to set attributes for other files. For example, to clear the attributes of *FILE2* and then set its attributes to match those of *FILE1* (enter this on one line):

```
c:\> attrib -arhs file2 & attrib +%@attrib[file1] file2
```

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

If no attributes are listed at all (*e.g.*, **/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.

This switch specifies which files to select, **not** which attributes to set. For example, to remove the archive attribute from all hidden files, you could use this command:

```
c:\> attrib /a:h -a *.*
```

/D (Directories) If you use the **/D** option, ATTRIB will modify the attributes of subdirectories in addition to files (yes, you can have a hidden subdirectory):

```
c:\> attrib /d +h c:\mydir
```

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) This option turns off ATTRIB's normal screen output. It is most useful in batch files.

/S (Subdirectories) If you use the **/S** option, the ATTRIB command will be applied to all matching files in the current or named directory and all of its subdirectories.

BEEP

Purpose: Beep the speaker or play simple music.

Format: **BEEP [*frequency duration* ...]**

frequency: The beep frequency in Hertz (cycles per second).
duration: The beep length in 1/18th second intervals.

Usage

BEEP generates a sound through your computer's speaker. It is normally used in batch files to signal that an operation has been completed, or that the computer needs attention.

Because BEEP allows you to specify the frequency and duration of the sound, you can also use it to play simple music or to create different kinds of signals for the user.

You can include as many frequency and duration pairs as you wish. No sound will be generated for frequencies less than 20 Hz, allowing you to use BEEP as a way to create short delays. The default value for *frequency* is 440 Hz; the default value for *duration* is 2.

This batch file fragment runs a program called *DEMO*, then plays a few notes and waits for you to press a key:

```
demo
beep 440 4  600 2  1040 6
pause Finished with the demo - hit a key...
```

The following table gives the *frequency* values for a five octave range (middle C is 262 Hz):

C	131	262	523	1046	2093
C# / Db	139	277	554	1108	2217
D	147	294	587	1175	2349
D# / Eb	156	311	622	1244	2489
E	165	330	659	1318	2637
F	175	349	698	1397	2794
F# / Gb	185	370	740	1480	2960
G	196	392	784	1568	3136
G# / Ab	208	415	831	1662	3322
A	220	440	880	1760	3520
A# / Bb	233	466	932	1866	3729
B	248	494	988	1973	3951

BREAK

Purpose: Display, enable, or disable Ctrl-C and Ctrl-Break checking for DOS applications.

Format: **BREAK [ON | OFF]**

Usage

The Ctrl-C and Ctrl-Break keys are used by many programs (including Take Command) as a signal to interrupt the current operation. BREAK controls how often DOS checks to see if you've entered one of these keystrokes.

Ctrl-C and Ctrl-Break checking cannot actually be enabled or disabled under Windows. The following discussion below applies only to DOS applications running under Take Command.

Normally, BREAK is turned off, and DOS only checks for Ctrl-C and Ctrl-Break keystrokes during DOS input or output operations involving the screen, keyboard, serial port, and printer. However, many programs don't use DOS for these operations, and it can be difficult to interrupt them.

When BREAK is turned on, DOS checks for Ctrl-C and Ctrl-Break every time a program calls DOS. Since most DOS applications call DOS to access files and perform other functions, turning BREAK on makes it much more likely that a Ctrl-C or Ctrl-Break will be noticed while running a DOS application.

Turning BREAK on or off does **not** affect how frequently Take Command itself checks for Ctrl-C or Ctrl-Break while an internal command is running. The BREAK setting only affects when DOS detects Ctrl-C and Ctrl-Break and notifies a DOS application. Any program can choose to ignore these signals. Also, any program can change the BREAK setting on its own.

Type BREAK plus ON or OFF to set the BREAK status, or BREAK by itself to display the current BREAK status. For example:

```
c:\> break on
c:\> break
BREAK is ON
```

BREAK is off by default. You can change the default by adding a BREAK=ON command to your *CONFIG.SYS* file.

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.

Take Command supports batch file nesting up to ten levels deep.

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

The called batch file should always either return (by executing its last line, or using the QUIT command), or terminate batch file processing with CANCEL. Do not restart or CALL the original batch file from within the called file as this may cause an infinite loop or a stack overflow.

CALL returns an exit code which matches the batch file return code. You can test this exit code with the %_? or %? environment variable, and use it with conditional commands (**&&** and **||**).

CANCEL

Purpose: Terminate batch file processing.

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

value: The numeric exit code to return to Take Command.

See also: CALL and QUIT.

Usage

The CANCEL command ends all batch file processing, regardless of the batch file nesting level. Use QUIT to end a nested batch file and return to the previous batch file.

You can CANCEL at any point in a batch file. If CANCEL is used from within an alias it will end execution of both the alias and any batch files which are running at the time.

The following batch file fragment compares an input line to "end" and terminates all batch file processing if it matches:

```
input Enter your choice:  %%option
if "%option" == "end" cancel
```

If you specify a *value*, CANCEL will set the ERRORLEVEL or exit code to that value (see the IF command, and the %? variable).

CD / CHDIR

Purpose: Display or change the current directory.

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

or

CHDIR [*path* | -]

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

See also: CDD, MD, PUSHD, RD, and Directory Navigation.

Usage

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

CD lets you navigate through a drive's subdirectory structure by changing the current working directory. If you enter CD and a directory name, the named directory becomes the new current directory. For example, to change to the subdirectory *C:\FINANCE\MYFILES*:

```
c:\> cd \finance\myfiles
c:\finance\myfiles>
```

Every disk drive on the system has its own current directory. Specifying both a drive and a directory in the CD command will change the current directory on the specified drive, but will not change the default drive. For example, to change the default directory on drive A:

```
c:\> cd a:\utility
c:\>
```

Notice that this command does not change to drive A:. Use the CDD command to change the current drive and directory at the same time.

You can change to the parent directory with **CD ..**; you can also go up one additional directory level with each additional [**.**]. For example, **CD ....** will go up three levels in the directory tree (see Extended Parent Directory Names). You can move to a sibling directory one that branches from the same parent directory as the current subdirectory with a command like **CD .\newdir .**

If you enter CD with no argument or with only a disk drive name, it will display the current directory on the default or named drive.

If CD cannot change to the directory you have specified it will attempt to search the CDPATH and the extended directory search database in order to find a matching directory and switch to it. You can also use wildcards in the ***path*** to force an extended directory search. Read the section on Directory Navigation for complete details on these and other directory navigation features.

CD saves the current directory before changing to a new directory. You can switch back to the previous directory by entering **CD -**. (There must be a space between the CD command and the hyphen.) You can switch back and forth between two directories by repeatedly entering **CD -**. The saved directory is the same for both the CD and CDD commands. Drive changes and automatic directory changes also modify the saved directory, so you can use **CD -** to return to a directory that you exited with an automatic directory change.

Directory changes made with CD are recorded in the directory history list and can be displayed in the

directory history window, which allows you to return quickly to a recently-used directory.

CD never changes the default drive. If you change directories on one drive, switch to another drive, and then enter **CD -**, the directory will be restored on the first drive but the current drive will not be changed.

Any change you make to the current directory is "global," and may affect other applications.

CDD

Purpose: Change the current disk drive and directory.

Format: **CDD [/A /S[drive …]] [*path* | -]**

path: The name of the directory (or drive and directory) to change to.
drive: A drive or list of drives to include in the extended directory search database.

/A(ll drives) **/S**(earch tree)

See also: CD, MD, PUSHD, RD, and Directory Navigation.

Usage

CDD is similar to the CD command, except that it also changes the default disk drive if one is specified. CDD will change to the directory and drive you name. To change from the root directory on drive A to the subdirectory *C:\WP:*

```
a:\> cdd c:\wp
c:\wp>
```

You can change to the parent directory with **CDD ..**; you can also go up one additional directory level with each additional [**.**]. For example, **CDD ....** will go up three levels in the directory tree.

CDD can also change to a network drive and directory specified with a UNC name (see File Systems for details).

If CDD cannot change to the directory you have specified it will attempt to search the CDPATH and the extended directory search database in order to find a matching directory and switch to it. You can also use wildcards in the ***path*** to force an extended directory search. Read the section on Directory Navigation for complete details on these and other directory navigation features.

CDD saves the current drive and directory before changing to a new directory. You can switch back to the previous drive and directory by entering **CDD -.** (There must be a space between the CDD command and the hyphen.) You can switch back and forth between two drives and directories by repeatedly entering **CDD -**. The saved directory is the same for both the CD and CDD commands. Drive changes and automatic directory changes also modify the saved directory, so you can use **CDD -** to return to a directory that you exited with a drive change or an automatic directory change.

Directory changes made with CDD are recorded in the directory history list and can be displayed directory history window, which allows you to return quickly to a recently-used directory.

Any change you make to the current directory is "global," and may affect other applications.

Options

/A (All drives) When CDD is used with this option, it displays the current directory on all drives from C: to the last drive in the system. You cannot move to a new drive and directory and use **/A** in the same command.

/S (Search tree) Builds or rebuilds the Extended Directory Search database, *JPSTREE.IDX*. You cannot move to a new drive and directory and use **/S** in the same command.

To include all local hard drives in the database, use the command:

```
cdd /s
```

To limit or add to the list of drives included in the database, list the drives and network volume names after the **/S** switch. For example, to include drives C, D, E, and the network volume \\ server\dir1 in the database, use this command:

```
cdd /s cde \\server\dir1
```

All non-hidden directories on the listed drives will be indexed; you cannot restrict the database to certain directories within a drive. Each time you use **/S**, everything in the previous directory database is replaced by the new database that is created.

CLS

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

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

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

/C(lear buffer)

Usage

CLS can be used to clear the screen without changing colors, or to clear the screen and change the screen colors simultaneously. These two examples show how to clear the screen to the default colors, and to bright white letters on a blue background:

```
c:\> cls
c:\> cls bright white on blue
```

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

See Colors and Color Names for details about colors.

Option

/C (Clear buffer) Clears the entire scrollback buffer. If **/C** is not used, only the visible portion of the window is cleared.

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 /E /H /K /M /N /P /Q /R /S /T /U /V /X /Z]** ***source*** **[+] ... [/A /B]** ***destination*** **[/A/B]**

source: A file or list of files or a device to copy ***from***.
destination: A file, directory, or device to copy ***to***.

/A(SCII)
/A: (Attribute select)
/B(inary)
/C(hanged)
/E (No error messages)
/H(idden)
/K(eep attributes)
/M(odified)
/N(othing)
/P(rompt)
/Q(uiet)
/R(eplace)
/S(ubdirectories)
/T(otals)
/U(pdate)
/V(erify)
/X (clear archive)
/Z (overwrite)

See also: ATTRIB, MOVE, and REN.

File Selection
Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, size or exclude ranges anywhere on the line apply to all *source* files.

Usage

The COPY command accepts all traditional syntax and options and adds many new features.

The simplest use of COPY is to make a copy of a file, like this example which makes a copy of a file called *FILE1.ABC*:

```
c:\> copy file1.abc file2.def
```

You can also copy a file to another drive and/or directory. The following command copies *FILE1* to the *MYDIR* directory on drive E:

```
c:\> copy file1 e:\mydir
```

Copying Files

You can copy several files at once by using wildcards:

```
c:\> copy *.txt e:\mydir
```

You can also list several ***source*** files in one command. The following command copies 3 files from the current directory to the *MYDIR* directory on drive E:

```
c:\> copy file1 file2 file3 e:\mydir
```

COPY also understands include lists, so you can specify several different kinds of files in the same command. This command copies the *.TXT*, *.DOC*, and *.BAT* files from the *E:\MYDIR* directory to the root directory of drive A:

```
c:\> copy e:\mydir\*.txt;*.doc;*.bat a:\
```

If there is only one argument on the line, COPY assumes it is the ***source***, and uses the current drive and directory as the ***destination***. For example, the following command copies all the *.DAT* files on drive A to the current directory on drive C:

```
c:\data> copy a:*.dat
```

If there are two or more arguments on the line separated by spaces, then COPY assumes that the last argument is the ***destination*** and copies all ***source*** files to this new location. If the ***destination*** is a drive, directory, or device name then the ***source*** files are copied individually to the new location. If the ***destination*** is a file name, the first ***source*** file is copied to the ***destination***, and any additional *source* files are then appended to the new ***destination*** file.

For example, the first of these commands copies the *.DAT* files from the current directory on drive A individually to *C:\MYDIR* (which must already exist as a directory); the second appends all the *.DAT* files together into one large file called *C:\DATA* (assuming *C:\DATA* is not a directory):

```
c:\> copy a:*.dat c:\mydir\
c:\> copy a:*.dat c:\data
```

When you copy to a directory, if you add a backslash [****] to the end of the name as shown in the first example above, COPY will display an error message if the name does not refer to an existing directory. You can use this feature to keep COPY from treating a mistyped ***destination*** directory name as a file name and attempting to append all your *source* files to a *destination* **file**, when you really meant to copy them individually to a *destination* **directory**.

To copy a file to a device such as the printer, use the device name as the ***destination***, for example:

```
c:\> copy schedule.txt prn
```

To copy text to or from the clipboard use **CLIP:** as the device name. Using CLIP: with non-text data will produce unpredictable results. See Redirection and Piping for more information on CLIP:.

Appending Files

A plus [**+**] tells COPY to append two or more files to a single ***destination*** file. If you list several *source* files separated with [**+**] and don't specify a ***destination***, COPY will use the name of the first ***source*** file as the destination, and append each subsequent file to the first file.

For example, the following command will append the contents of *C:\MEMO2* and *C:\MEMO3* to *C:\MEMO1* and leave the combined contents in the file named *C:\MEMO1*:

```
c:\> copy memo1+memo2+memo3
```

To append the same three files but store the result in *BIGMEMO:*

```
c:\> copy memo1+memo2+memo3 bigmemo
```

If no ***destination*** is specified, the destination file will always be created in the current directory even if the first ***source*** file is in another directory or on another drive. For example, this command will 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.

Advanced Features

If your ***destination*** has wildcards in it, COPY will attempt to match them with the ***source*** names. For example, this command copies the *.DAT* files from drive A to *C:\MYDIR* and gives the new copies the extension *.DX*:

```
c:\> copy a:*.dat c:\mydir\*.dx
```

This feature can give you unexpected results if you use it with multiple ***source*** file names. For example, suppose that drive A contains *XYZ.DAT* and *XYZ.TXT*. The command

```
c:\ copy a:\*.dat a:\*.txt c:\mydir\*.dx
```

will copy *A:XYZ.DAT* to *C:\MYDIR\XYZ.DX*. Then it will copy *A:XYZ.TXT* to *C:\MYDIR\XYZ.DX*, overwriting the first file it copied.

You can use date, time, and size ranges to further define the files that you want to copy. This example copies every file in the *E:\MYDIR* directory, which was created or modified yesterday, and which is also 10,000 bytes or smaller in size, to the root directory of drive A:

```
c:\> copy /[d-1] /[s0,10000] e:\mydir\*.* a:\
```

You can also use file exclusion ranges to restrict the list of files that would normally be selected with wildcards. This example copies every file in the *E:\MYDIR* directory except backup (*.BAK* or *.BK!*) files:

```
c:\> copy /[!*.bak;*.bk!] e:\mydir\*.* a:\
```

COPY will normally process ***source*** files which do not have the hidden or system attribute, and will ignore the read-only and archive attributes. It will always set the archive attribute and clear the read-only attribute of ***destination*** files. In addition, if the ***destination*** is an existing file with the read-only attribute, COPY will generate an "Access Denied" error and refuse to overwrite the file. You can alter some of these behaviors with switches:

/A: Forces COPY to process ***source*** files with the attributes you specify after the **:**, or to process all ***source*** files regardless of attributes (if **/A:** is used by itself).

/H Forces COPY to process hidden and system ***source*** files, as well as normal files. The hidden and system attributes from each source file will be preserved when creating the ***destination*** files.

/K Retains the read-only attribute from each ***source*** file when creating the ***destination*** file. See **/K** below for a special note if you are running under Novell Netware.

/Z Forces COPY to overwrite an existing read-only ***destination*** file.

Use caution with **/A:**, **/H**, or **/K** when both the ***source*** and ***destination*** directories contain file descriptions. If the ***source*** file specification matches the description file name (normally *DESCRIPT.ION*), and you use a switch which tells COPY to process hidden files, the *DESCRIPT.ION* file itself will be copied, overwriting any existing file descriptions in the ***destination*** directory. For example, if the *\DATA* directory contains file descriptions this command would overwrite any existing descriptions in the *\SAVE* directory:

```
c:\data> copy /h d*.* \save\
```

(If you remove the hidden attribute from the *DESCRIPT.ION* file the same caution applies even if you do not use **/A:**, **/H**, or **/K**, as *DESCRIPT.ION* is then treated like any other file.)

Options

The **/A**(SCII) and **/B**(inary) options apply to the preceding filename and to all subsequent filenames on the command line until the file name preceding the next **/A** or **/B**, if any. The other options (**/A:**, **/C**, **/E**, **/H**, **/K**, **/M**, **/N**, **/P**, **/Q**, **/R**, **/S**, **/T**, **/U**, **/V**, **/X**, **/Z**) apply to all filenames on the command line, no matter where you put them. For example, either of the following commands could be used to copy a font file to the printer in binary mode:

```
c:\> copy /b myfont.dat prn
c:\> copy myfont.dat /b prn
```

Some options do not make sense in certain contexts, in which case COPY will ignore them. For example, you cannot prompt before replacing an existing file when the ***destination*** is a device such as the printer there's no such thing as an "existing file" on the printer. If you use conflicting output options, like **/Q** and **/P**, COPY will generally take a "conservative" approach and give priority to the option which generates more prompts or more information.

/A (ASCII) If you use **/A** with a ***source*** filename, the file will be copied up to, but not including, the first Ctrl-Z (Control-Z or ASCII 26) character in the file (some application programs use the Ctrl-Z to mark the end of a file). If you use **/A** with a ***destination*** filename, a Ctrl-Z will be added to the end of the file. **/A** is the default when appending files, or when the ***destination*** is a device like NUL or PRN, rather than a disk file.

/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	**D**	Subdirectory
H	Hidden	**A**	Archive
S	System		

If no attributes are listed at all (*e.g.*, **COPY /A:**), COPY will select all files and subdirectories including hidden and system files (this is equivalent to **COPY /H**). If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set. See the cautionary note under **Advanced Features** above before using **/A:** when both ***source*** and ***destination*** directories contain file descriptions.

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.

/E (No error messages) Suppress all non-fatal error messages, such as "File not found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

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

See the cautionary note under **Advanced Features** above before using **/H** when both ***source*** and ***destination*** directories contain file descriptions.

/K (Keep attribute) To maintain compatibility with *COMMAND.COM*, *CMD.EXE*, and Netware, COPY normally maintains the hidden and system attributes, sets the archive attribute, and removes the read-only attribute on the target file. **/K** tells COPY to also maintain the read-only attribute on the ***destination*** file. However, if the ***destination*** is on a Novell Netware volume, this option will fail to maintain the read-only attribute. This is due to the way Netware handles file attributes, and is not a problem in COPY.

/M (Modified) Copy only those files with the archive attribute set, *i.e.*, those which have been modified since the last backup. The archive attribute of the ***source*** file will **not** be cleared after copying; to clear it use the **/X** switch, or use ATTRIB.

/N (Nothing) Do everything except actually perform the copy. This option is useful for testing what the result of a complex COPY command will be. **/N** does **not** prevent creation of ***destination*** subdirectories when it is used with **/S**.

/P (Prompt) Ask the user to confirm each *source* file. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames or the total number of files copied. This option is most often used in batch files. See also **/T**.

/R (Replace) Prompt the user before overwriting an existing file. Your options at the prompt are explained in detail under Page and File Prompts.

/S (Subdirectories) Copy the subdirectory tree starting with the files in the ***source*** directory plus each subdirectory below that. The ***destination*** must be a directory; if it doesn't exist, COPY will attempt to create it. COPY will also attempt to create needed subdirectories on the tree below the ***destination***, including empty ***source*** directories. If COPY /S creates one or more destination directories, they will be added automatically to the extended directory search database.

If you attempt to use COPY /S to copy a subdirectory tree into part of itself, COPY will detect the resulting infinite loop, display an error message and exit.

/T (Totals) Turns off the display of filenames, like **/Q**, but does display the total number of files copied.

/U (Update) Copy each ***source*** file only if it is newer than a matching ***destination*** file or if a matching ***destination*** file does not exist (see also **/C**). This option is useful for keeping one directory matched with another with a minimum of copying.

/V (Verify) Verify each disk write. This is the same as executing the VERIFY ON command, but is only active during the COPY. **/V** does **not** read back the file and compare its contents with what was written; it only verifies that the data written to disk is physically readable.

/X (Clear archive) Clears the archive attribute from the source file after a successful copy. This option is most useful if you are using COPY to maintain a set of backup files.

/Z (Overwrite) Overwrites read-only ***destination*** files. Without this option, COPY will fail with an "Access denied" error if the ***destination*** file has its read-only attribute set. This option allows COPY to overwrite read-only files without generating any errors.

DATE

Purpose: Display and optionally change the system date.

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

mm: The month (1 - 12).
dd: The day (1 - 31).
yy: The year (80 - 99 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
Mon  Dec 22, 1997  9:30:06
Enter new date (mm-dd-yy):
```

You can also enter a new system date by typing the DATE command plus the new date on the command line:

```
c:\> date 10-16-97
```

You can use hyphens, slashes, or periods to separate the month, day, and year entries. The year can be entered as a 2-digit or 4-digit value. Two-digit years between 80 and 99 are interpreted as 1980 - 1999; values between 00 and 79 are interpreted as 2000 - 2079.

DATE adjusts the format it expects depending on your country settings. When entering the date, use the correct format for the country setting currently in effect on your system.

DDEEXEC

Purpose: Send a DDE command to another application.

Format: **DDEEXEC *server*, *topic*, *command***

server: The DDE name of the program that will receive the command.
topic: The server's topic name for receiving the command.
command: The command string to send to the server.

Usage

Windows supports a form of communication between programs called Dynamic Data Exchange or DDE. Using DDE, one program can send a command or data to another. The receiving program is usually called the DDE server; the sending program is usually called the DDE client. When you use DDEEXEC, Take Command acts as a DDE client and the program which receives the command is the server. (Take Command can also act as a DDE server. See the DDE topic for details.)

For example, if you want to instruct the Program Manager to display its Main group, you can use this command:

```
c:\> ddeexec progman, progman, [ShowGroup(Main,1)]
```

In this example, the ***server*** name and the ***topic*** name are both **progman** and the ***command*** is **[ShowGroup(Main,1)]**

The server name, topic name, and possible commands are defined by the server application. See the documentation included with the programs to which you want to send DDE messages for details about the names and commands to use.

DEL / ERASE

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

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

or

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

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

/A: (Attribute select)	**/T**(otal)
/E (No error messages)	**/W**(ipe)
/N(othing))	**/X** (remove empty subdirectories)
/P(rompt	**/Y**(es to all prompts)
/Q(uiet)	**/Z**(ap hidden and read-only files)
/S(ubdirectories)	

File Selection

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

Usage

DEL and ERASE are synonyms, you can use either one.

Use the DEL and ERASE commands with caution; the files and subdirectories that you erase may be impossible to recover without specialized utilities and a lot of work.

To erase a single file, simply enter the file name:

```
c:\> del letters.txt
```

You can also erase multiple files in a single command. For example, to erase all the files in the current directory with a *.BAK* or *.PRN* extension:

```
c:\> del *.bak *.prn
```

To exclude files from a DEL command, use a file exclusion range. For example, to delete all files in the current directory except those whose extension is *.TXT*, use a command like this:

```
c:\> del /[!*.TXT] *.*
```

When using exclusion ranges or other more complex options you may want to use the **/N** switch first, to preview the effects of the DEL without actually deleting any files.

If you enter a subdirectory name, or a filename composed only of wildcards (* and/or ?), DEL asks for confirmation (**Y** or **N**) unless you specified the **/Y** option. If you respond with a **Y**, DEL will delete all the files in that subdirectory (hidden, system, and read-only files are only deleted if you use the **/Z** option).

DEL displays the amount of disk space recovered, unless the **/Q** option is used (see below). It does so by comparing the amount of free disk space before and after the DEL command is executed. This amount may be incorrect if you are using a deletion tracking system which stores deleted files in a hidden directory, or if, under a multitasking system, another program performs a file operation while the DEL command is executing.

Remember that DEL removes file descriptions along with files. Most deletion tracking systems will not be able to save or recover a file's description, even if they can save or recover the data in a file.

When a file is deleted, its disk space is returned to the operating system for use by other files. However, the contents of the file remain on the disk until they are overwritten by another file. If you wish to obliterate a file or wipe its contents clean, use DEL /W, which overwrites the file with zeros before deleting it. Use this option with caution once a file is obliterated, it is impossible to recover.

DEL returns a non-zero exit code if no files are deleted, or if another error occurs. You can test this exit code with the %_? environment variable, and use it with conditional commands (**&&** and **||**).

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

If no attributes are listed at all (*e.g.*, **DEL /A:**), DEL will delete 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 deletion. For example, **/A:RHS** will select only those files with all three attributes set.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/N (Nothing) Do everything except actually delete the file(s). This is useful for testing what the result of a DEL would be.

/P (Prompt) Prompt the user to confirm each erasure. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames as they are deleted, or the number of files deleted or bytes freed. See also **/T**.

/S (Subdirectories) Delete the specified files in this directory and all of its subdirectories. This 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.

/W (Wipe) Clear the file to zeros before deleting it. Use this option to completely obliterate a file's contents from your disk. Once you have used this option it is impossible to recover the file even if you are using an undelete utility, because the contents of the file are destroyed before it is deleted. **/W** overwrites the file only once; it does **not** adhere to security standards which require multiple overwrites with varying data when destroying sensitive information.

/X (Remove empty subdirectories) Remove empty subdirectories after deleting (only useful when used with **/S**). If DEL deletes one or more directories, they will be removed automatically from the extended directory search database.

/Y (Yes) The reverse of **/P** it assumes a **Y** response to everything, including deleting an entire subdirectory tree. Take Command normally prompts before deleting files when the name consists only of wildcards or a subdirectory name (see above); **/Y** overrides this protection, and should be used with extreme caution!

/Z (Zap) Delete read-only, hidden, and system files as well as normal files. Files with the read-only, hidden, or system attribute set are normally protected from deletion; **/Z** overrides this protection, and should be used with caution. Because EXCEPT works by hiding files, **/Z** will override an EXCEPT command. However, files specified in a file exclusion range will not be deleted by DEL /Z.

For example, to delete the entire subdirectory tree starting with C:\UTIL, including hidden and read- only files, without prompting (use this command with CAUTION!):

```
c:\> del /sxyz c:\util\
```

DELAY

Purpose: Pause for a specified length of time.

Format: **DELAY [*seconds*]**

seconds: The number of seconds to delay.

Usage

DELAY is useful in batch file loops while waiting for something to occur. To wait for 10 seconds:

```
delay 10
```

DELAY is most useful when you need to wait a specific amount of time for an external event, or check a system condition periodically. For example, this batch file checks the battery status (as reported by your Advanced Power Management drivers) every 15 seconds, and gives a warning when battery life falls below 30%:

```
do forever
   iff %_apmlife lt 30 then
      beep 440 4 880 4 440 4 880 4
      echo Low Battery!!
   endiff
   delay 15
enddo
```

The ***seconds*** value can be as large as 3600 (one hour).

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).

Take Command uses the minimum possible processor time during a DELAY, in order to allow other applications full use of system resources.

You can cancel a delay by pressing **Ctrl-C** or **Ctrl-Break**.

DESCRIBE

Purpose: Create, modify, or delete file and subdirectory descriptions.

Format: **DESCRIBE [/A:[[-]rhsda]] *file* [[/D]*"description"*] ...**

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

/A: (Attribute select) **/D**(escription follows)

File Selection

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

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 can also enter or modify descriptions with the Descriptions dialog. The dialog allows you to select a single file and modify its description using a dialog box, rather than using the DESCRIBE command. The information in this section also applies to descriptions created via the dialog, unless otherwise noted.

You enter a description on the command line by typing the DESCRIBE command, the filename, and the description in quotation marks, like this:

```
c:\> describe memo.txt "Memo to Bob about party"
```

If you don't put a description on the command line, DESCRIBE will prompt you for it:

```
c:\> describe memo.txt
Describe "memo.txt" : Memo to Bob about party
```

If you use wildcards or multiple filenames with the DESCRIBE command and don't include the description text, you will be prompted to enter a description for each file. If you do include the description on the command line, all matching files will be given the same description.

If you enter a quoted description on the command line, and the text matches the name of a file in the current directory, the command processor will treat the string as a quoted file name, not as description text as you intended. To resolve this problem use the **/D** switch immediately prior to the quoted description (with no intervening spaces). For example, if the current directory contains the files *DATA.TST* and *Test File*, the first of these commands will work as intended, but the second will not (in the second example the string test file will be treated as a second file name, when it is intended to be description text):

```
c:\> describe data.tst /D"test file"
c:\> describe data.tst "test file"
```

Each description can be up to 511 characters long. You can change this limit on the Options 1 page of the configuration dialogs, or with the DescriptionMax directive in the *TCMD.INI* file. In order to fit your descriptions on a single line in a standard DIR display, keep them to 40 characters or less (longer descriptions are wrapped in the DIR output). DESCRIBE can edit descriptions longer than DescriptionMax (up to a limit of 511 characters), but will not allow you to lengthen the existing text.

The descriptions are stored in each directory in a hidden file called *DESCRIPT.ION*. Use the ATTRIB command to remove the hidden attribute from this file if you need to copy or delete it. *DESCRIPT.ION* is always created as a hidden file, but will not be re-hidden by Take Command if you remove the hidden attribute.

You can change the description file name with the DescriptionName directive in the *TCMD.INI* file or the SETDOS /D command, 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 like the Windows Explorer). You can disable description processing on the Options 1 page of the configuration dialogs, with the Descriptions directive in the *TCMD.INI* file, or with SETDOS /D.

When you COPY or MOVE files between two directories, both of which have descriptions, and you use switches which enable processing of hidden files (or you have removed the hidden attribute from *DESCRIPT.ION*), you must use caution to avoid overwriting existing file descriptions in the ***destination*** directory with the *DESCRIPT.ION* file from the ***source*** directory. See the notes under the **Advanced Features** sections of COPY and MOVE for additional details.

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

If no attributes are listed at all (*e.g.*, **/A:**), DESCRIBE will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/D (Description follows) The quoted string immediately following this switch is a description, not a file name. Use **/D** to avoid any ambiguity in the meaning of quoted strings. See the **Usage** section above for details.

DIR

Purpose: Display information about files and subdirectories.

Format: **DIR [/1 /2 /4 /A[[:][-]rhsda] /B /C[HP] /D /E /F /G /H /I"text" /J /K /L /M /N /O[[:][]acdeginrsu] /P /R /S /T /U /V /W] [*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)
/C(ompression)
/D(isable color coding)
/E (use upper case)
/F(ull path)
/G (allocated size)
/H(ide dots)
/I (match descriptions)
/J(ustify names)
/K (suppress header)
/L(ower case)
/M (suppress footer)
/N(ormal)
/O(rder)
/P(ause)
/R (disable wRap)
/S(ubdirectories)
/T (aTtribute)
/U (sUmmary information)
/V(ertical sort)
/W(ide)

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.

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
```

The following sections group DIR's features together in several categories. Many of the sections move from a general discussion to more technical material. If you find some of the information in a category too detailed for your needs, feel free to skip to the beginning of the next section. The sections are:

Selecting Files

DIR can display information about a single file or about several, dozens, hundreds, or thousands of files at once. To display information about a single file, just add the name of the file to the DIR command line:

```
c:\> dir january.wks
```

The simplest way to view information about several files at once is to use wildcards. DIR can work with traditional wildcard caracters (* and ?) and the extended wildcards. For example to display all of the *.WKS* files in the current directory:

```
c:\> dir *.wks
```

To display all *.TXT* files whose names begin with A, B, or C:

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

If you don't specify a filename, DIR defaults to ***.***. This default displays all non-hidden files and subdirectories in the current directory.

If you link two or more filenames together with spaces, DIR will display all of the files that match the first name and then all of the files that match the second name. You may use a different drive and path for each filename. This example lists all of the *.WKS* and then all of the *.WK1* files in the current directory:

```
c:\> dir *.wks *.wk1
```

If you use an include list to link multiple filenames, DIR will display the matching filenames in a single listing. Only the first filename in an include list can have a path; the other files must be in the same path. This example displays the same files as the previous example, but the *.WKS* and *.WK1* files are intermixed:

```
c:\> dir *.wks;*.wk1
```

You can include files in the current or named directory plus all of its subdirectories by using the **/S** option. This example displays all of the *.WKS* and *.WK1* files in the D:\DATA directory and each of its subdirectories:

```
c:\> dir /s d:\data\*.wks;*.wk1
```

You can also select files by their attributes by using the **/A** option. For example, this command displays the names of all of the subdirectories of the current directory:

```
c:\> dir /a:d
```

Finally, with the **/I** option, DIR can select files to display based on their descriptions (see DESCRIBE for

more information on file descriptions). DIR will display a file if its description matches the text after the **/I** switch. The search is not case sensitive. You can use wildcards and extended wildcards as part of the text. For example, to display any file described as a "Test File" you can use this command:

```
c:\> dir /i"test file"
```

If you want to display files that include the words "test file" anywhere in their descriptions, use extended wild cards like this:

```
c:\> dir /i"*test file*"
```

To display only those files which do **not** have descriptions, use:

```
c:\> dir /I"[]"
```

In addition, you can use ranges to select or exclude specific sets of files. For example, to display all files modified in the last week, all files except those with a *.BAK* extension, and all files over 500 KB in size:

```
c:\> dir /[d-7]
c:\> dir /[!*.bak]
c:\> dir /[s500K]
```

You can, of course, mix any of these file selection techniques in whatever ways suit your needs.

Default DIR Output Format

The default DIR format contains 5 columns: the file name, the file size in bytes, the date of the last write, the time of the last write, and the file's description. File names are listed in lower-case; directory names in upper case:

```
 Volume in drive C is C - BOOTUP    Serial ...
 Directory of  C:\4DOS60\*.*

.            <DIR>     10-24-96  12:17
..           <DIR>     10-24-96  12:17
TEST         <DIR>     11-01-96  16:21
4dos6.pif          967  10-28-96   7:57 4DOS PIF file
4dos.com        212854  10-21-96  18:08 4DOS exe ...
4dos.ini            45  11-02-96  10:08 4DOS conf ...
```

DIR's output is normally sorted by name, with directories listed first. You can change the sort order with the **/O** option. For example, these two commands sort the output by date the first command lists the oldest file first; the second command lists the oldest file last:

```
c:\> dir /o:d
c:\> dir /o:-d
```

When displaying file descriptions, DIR wraps long lines to fit on the screen. DIR displays a maximum of 40 characters of text in each line of a description, unless your screen width allows a wider display. If you disable description wrapping with the **/R** option, the description is truncated at the right edge of the screen, and a right arrow [] is added at the end of the line to alert you to the existence of additional description text.

DIR's default output is sorted. It displays directory names first, with "<DIR>" inserted instead of a file size, and then filenames. DIR assumes that sequences of digits should be sorted numerically (for example, the file *DRAW2* is listed before *DRAW03* because 2 is numerically smaller than 03), rather than

strictly alphabetically (where *DRAW2* would come second because "2" is after "0" in alphanumeric order). You can change the sort order with the **/O** option. When DIR displays file names in a multi-column format, it sorts file names horizontally unless you use the **/V** option to display vertically sorted output.

DIR's display can be modified in many ways to meet different needs. Most of the following sections describes the various ways you can change DIR's output format.

Switching Formats

If you use the **/B** option, DIR displays just file names and omits the file size, time stamp, and description for each file, for example:

```
c:\>dir w* /b
WINDOWS
WINNT
win311
WINALIAS
WINENV.BTM
.....
```

There are several ways to modify the display produced by **/B**. The **/F** option is similar to **/B**, but displays the full path and name of each file, instead of just its name. To view the same information for a directory and its subdirectories use **/B /S** or **/F /S.**

There are several ways to modify the display produced by **/B**. The **/F** option is similar to **/B**, but displays the full path and name of each file, instead of just its name. To view the same information for a directory and its subdirectories use **/B /S** or **/F /S.** You can use **/B /X** to display the short name of each file, with no additional information. To display the short version of both the path and file name for each file, use **/F /X**. For example:

```
c:\>dir /x/f/s *.pif
C:\MACH64\INSTALL.PIF
C:\PROGRA~1\WINZIP\WZ.PIF
C:\WINDOWS\DOSPRMPT.PIF
C:\WINDOWS\STARTM~1\APPS&U~1\INFOSE~1.PIF
C:\WINDOWS\STARTM~1\PROMPTS\4DOS.PIF
C:\WINDOWS\STARTM~1\PROMPTS\TOCP.PIF
C:\WINDOWS\STARTM~1\PROMPTS\SPECIAL\4DOS(R~1.PIF
.....
```

Multiple Column Displays

DIR has three options, **/2**, **/4**, and **/W**, that create multi-column displays.

The **/2** option creates a 2-column display. The display includes the short name, file size, and time stamp for each file.

The **/4** option is similar to **/2,** but displays directory information in 4 columns. Each contains a file name and the file size in kilobytes (KB) or megabytes (MB). Directories are shown with "<D>" in the size column.

The **/W** option displays directory information in 5 or more columns, depending on your screen width. Each entry in a **DIR /W** display contains either the name of a file or the name of a directory. Directory names are placed in square brackets to distinguish them from file names.

Color-Coded Directories

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

To choose the display colors, you must either use the SET command to create an environment variable called COLORDIR, use the configuration dialogs, or use the directive in the *TCMD.INI* file. If you do not use the COLORDIR variable or the ColorDir directive, DIR will use the default screen colors for all files.

If you use both the COLORDIR variable and the ColorDir directive, the environment variable will override the settings in your *.INI* file. You may find it useful to use the COLORDIR variable for experimenting, then to set permanent directory colors with the ColorDir directive.

The format for both the COLORDIR environment variable and the ColorDir directive in the *.INI* file is:

```
ext ... :ColorName; ...
```

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

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

and "ColorName" is any valid color name (see Colors and Color Names for information on color names).

Unlike most color specifications, the background portion of the color name may be omitted for directory colors. If you don't specify a background color, DIR will use the current screen background color.

For example, to display the *.COM* and *.EXE* files in red on the current background, the *.C* and *.ASM* files in bright cyan on the current background, and the read-only files in green on white:

```
c:\> set colordir=com exe:red; c asm:bright cyan; rdonly:green on white
```

Extended wildcards can be used in directory color specifications. For example, to display *.BAK*, *.BAX*, and *.BAC* files in red:

```
c:\> set colordir=BA[KXC]:red
```

Redirected Output

The output of the DIR command, like that of most other internal commands, can be redirected to a file, printer, serial port, or other device. However, you may need to take certain DIR options into account when you redirect DIR's output.

DIR wraps both long file names and file descriptions at the width of your display. Its redirected output will also wrap at the screen width. Use the **/R** option if you wish to disable wrapping of long descriptions.

If you redirect a color-coded directory to a file, DIR will remove the color data as it sends the directory information to a file. It will usually do the same if you redirect output to a character device such as a printer or serial port. However, it is not always possible for DIR to tell whether or not a device is a character device. If you notice that non-colored lines are being sent to the output device and colored lines are appearing on your screen, you can use the **/D** option to temporarily disable color-coding when you redirect DIR's output.

To redirect DIR output to the clipboard, use **CLIP:** as the output device name, for example:

```
c:\> dir *.exe > clip:
```

Other Notes

If you have selected a specific country code for your system, DIR will display the date in the format for that country. The default date format is U.S. (mm-dd-yy). The separator character in the file time will also be affected by the country code. Thousands and decimal separators in numeric displays are affected by the country code, and by the ThousandsChar and DecimalChar settings selected with the configuration dialogs or in the *TCMD.INI* file.

If you are using a disk compression program, you can use the **/C** switch to view the amount of compression achieved for each file. When you do, the compression ratio is displayed instead of the file's description. You can also sort the display by compression ratios with the **/O:c** switch. Details for both switches are in the Options section, below.

Only the compression programs distributed with MS-DOS and Windows (DRVSPACE and DBLSPACE) are supported. **/C** and **/O:c** will be ignored for other compression programs, and on uncompressed drives.

DOS networks with large server volumes (over 2 GB) may report incorrect free disk space values at the end of the DIR display. If this occurs, it is because the network software does not report the proper values to Take Command.

Options

Options on the command line apply only to the filenames which follow the option, and options at the end of the line apply to the preceding filename only. This allows you to specify different options for different groups of files, yet retains compatibility with the traditional DIR command when a single filename is specified.

/1 Single column display display the filename, size, date, time, and the description. This is the default. If **/C** or **/O:c** is used, the compression ratio is displayed instead of the description. This option is most useful if you wish to override a default **/2**, **/4**, or **/W** setting stored in an alias.

/2 Two column display display the filename, size, date, and time.

/4 Four column display display the filename and size, in K (kilobytes) or M (megabytes). Files between 1 and 9.9 megabytes in size will be displayed in tenths (*i.e.*, "2.4M").

/A[:] (Attribute select) Display only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is optional. The attributes are:

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

If no attributes are listed at all (*e.g.*, **DIR /A**), DIR will display all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be included in the listing. For example, **/A:RHS** will display only those files with all three attributes set.

/B (Bare) Suppress the header and summary lines, and display file or subdirectory names only, in a single column. This option is most useful when you want to redirect a list of names to a file or another program. If you use **/B** with **/S**, DIR will show the full path of each file (the

same display as **/F**) instead of simply its name and extension.

/C (Compression) Display per-file and total compression ratio on compressed drives. The compression ratio is displayed instead of the file description or attributes. The ratio is left blank for directories and files with a length of 0 bytes and for files on non-compressed drives. **/C** only works in single-column mode; it is ignored if **/2**, **/4**, or **/W** is used.

Only the compression programs distributed with MS-DOS (DRVSPACE and DBLSPACE) are supported.

The numerator of the displayed compression ratio is the amount of space which would be allocated to the file if the compression utility were not in use, based on the compressed drive's cluster size (usually 8K bytes). The denominator is the space actually allocated for the compressed file. For example, if a file is allocated 6,144 bytes when compressed, and would require 8,192 bytes if uncompressed, the displayed compression ratio would be 8,192 / 6,144, or 1.3 to 1.

Using **/CH** displays compression ratios like **/C**, but bases the calculation on the host drive's cluster size. This gives a more accurate picture of the space saved through compression than is given by **/C**. This option will occasionally display compression ratios slightly less than 1.0 for files which have actually expanded when stored on the compressed drive.

If **/CP** is used instead of **/C**, the compression is displayed as a percentage (*e.g.*, 33%) instead of a ratio (*e.g.*, 3 to 1). If **/CHP** is used instead of **/CH**, the host compression is displayed as a percentage. The **/CHP** option must be entered as shown; you can **not** use **/CPH**.

/**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 upper case; also see SETDOS /U and the UpperCase directive in the *TCMD.INI* file.

/F (Full path) Display each filename with its drive letter and path in a single column, without other information.

/G (Allocated space) Display the allocated disk space instead of the actual size of each file.

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

/I Display filenames by matching text in their descriptions. The text can include wild cards and extended wildcards. The search text must be enclosed in quotation marks. You can select all filenames that have a description with **/I"[?]*"**, or all filenames that do not have a description with **/I"[]"**.

The **/I** option may be used to select files even if descriptions are not displayed (for example, if **/2** is used). However, **/I** will be ignored if **/C** or **/O:c** is used.

/J (Justify names) Justify (align) filename extensions and display them in the traditional format.

/K Suppress the header (disk and directory name) display.

/L (Lower case) Display file and directory names in lower case; also see SETDOS /U and the UpperCase directive in the *TCMD.INI* file.

/M Suppress the footer (file and byte count totals) display.

/N (Normal) Reset the DIR options to the default values. This is useful when you want to display some files in one format, and then change back to the defaults for another set of files.

/O (Order) Set the sorting order. You may use any combination of the following sorting options; if multiple options are used, the listing will be sorted with the first sort option as the primary key, the next as the secondary key, and so on:

- **-** Reverse the sort order for the next option
- **a** Sort in ASCII order, not numerically, when there are digits in the name
- **c** Sort by compression ratio (the least compressed file in the list will be displayed first). For single-column directory displays in the traditional short filename format, the compression ratios will be used as the basis of the sort and will also be displayed. For wider displays (**/2**, **/4**, and **/W**), the compression ratios will be used to determine the order but will not be displayed. If **/O:c** is used with **/CH** or **/CHP**, the sort will be based on the host-drive compression ratios. For information on supported compression systems see **/C** above.
- **d** Sort by date and time (oldest first)
- **e** Sort by extension
- **g** Group subdirectories first, then files
- **i** Sort by file description (ignored if **/C** or **/O:c** is also used).
- **n** Sort by filename (this is the default)
- **r** Reverse the sort order for all options
- **s** Sort by size
- **u** Unsorted

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/R (disable wRap) Forces long descriptions to be displayed on a single line, rather than wrapped onto two or more lines. Use **/R** when output is redirected to a character device, such as a serial port or the printer; or when you want descriptions truncated, rather than wrapped, in the on-screen display.

/S (Subdirectories) Display file information from the current directory and all of its subdirectories. DIR will only display headers and summaries for those directories which contain files that match the filename(s), ranges, and attributes that you specify 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 RHSDA, where:

/U (sUmmary information) Only display the number of files, the total file size, and the total amount of disk space used. Information on individual files is not displayed.

/V (Vertical sort) Display the filenames sorted vertically rather than horizontally (use with the **/2**, **/4** or **/W** options).

/W (Wide) Display filenames only, horizontally across the screen. **/W** displays as many columns as it can fit into the Take Command window, using 16 characters in each column.

DIRHISTORY

Purpose: Display, add to, clear, or read the directory history list.

Format: **DIRHISTORY [/A *directory* /F /P /R *filename*]**

directory: The name of a directory to be added to the directory history.
filename: The name of a file containing entries to be added to the directory history.

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

See also: HISTORY.

Usage

Every time you change to a new directory or drive, Take Command records the current directory in an internal directory history list. See the directory history window, which allows you to use the list to return to a previous directory. Also see directory navigation.

The DIRHISTORY command lets you view and manipulate the directory history list directly. If no parameters are entered, DIRHISTORY will display the current directory history list:

```
c:\> dirhistory
```

With the options explained below, you can clear the list, add new directories to the list without changing to them, save the list in a file, or read a new list from a file.

The number of directories saved in the directory history list depends on the length of each directory name. The list size can be specified at startup from 256 to 32767 characters (see the configuration dialogs or the DirHistory directive in the *TCMD.INI* file). The default size is 256 characters.

You can save the directory history list by redirecting the output of DIRHISTORY to a file. This example saves the history to a file called *DIRHIST* and reads it back again.

```
c:\> dirhistory > dirhist
     .....
c:\> dirhistory /r dirhist
```

Because the directory history stores each name only once, you don't have to delete its contents before reading back the file unless you want to delete the directories that were visited by the intervening commands.

If you need to save your directory history at the end of each day's work, you might use the first of these commands in your *TCSTART.BTM* or other startup file, and the second in *TCEXIT.BTM*:

```
if exist c:\dirhist dirhistory /r c:\dirhist
dirhistory > c:\dirhist
```

This restores the previous history list if it exists, and saves the history when Take Command exits.

DIRS

Purpose: Display the current directory stack.

Format: **DIRS**

See also: PUSHD, POPD, and Directory Navigation.

Usage

The PUSHD command adds the current default drive and directory to the directory stack, a list that Take Command 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).

For example, to change directories and then display the directory stack:

```
c:\> pushd c:\database
c:\database> pushd d:\wordp\memos
d:\wordp\memos> dirs
c:\database
c:\
```

The directory stack holds 511 characters, enough for 20 to 40 typical drive and directory entries.

DO

Purpose: Create loops in batch files.

Format: **DO [*n* | FOREVER]**

or

DO *varname* = *start* TO *end* [BY *n*]

or

DO [WHILE | UNTIL] *condition*

or

DO *varname* IN [@]*set*

commands

[ITERATE]

[LEAVE]

commands

ENDDO

varname: The environment variable that will hold the loop counter, filename, or line from a file.

n, start, end: Integers between 0 and 2,147,483,647 inclusive, or an internal variable or variable function that evaluates to such a value.

condition: A test to determine if the loop should be executed.

set: A set of values for the variable.

commands: One or more commands to execute each time through the loop. If you use multiple commands, they must be separated by command separators or be placed on separate lines.

Supports extended wildcards, ranges, and include lists.

Usage

DO can only be used in batch files.

DO can be used to create 4 different kinds of loops. The first, introduced by **DO n**, is a counted loop. The batch file lines between DO and ENDDO are repeated *n* times. For example:

```
do 5
beep
enddo
```

You can also specify "forever" for ***n*** if you wish to create an endless loop (you can use LEAVE or GOTO to exit such a loop; see below for details).

The second type of loop is similar to a "for loop" in programming languages like BASIC. DO creates an environment variable, *varname*, and sets it equal to the value *start* (if *varname* already exists in the environment, it will be overwritten). DO then begins the loop process by comparing the value of *varname*

with the value of *end.* If *varname* is less than or equal to *end*, DO executes the batch file lines up to the ENDDO. Next, DO adds 1 to the value of *varname*, or adds the value *n* if BY *n* is specified, and repeats the compare and execute process until *varname* is greater than *end*. This example displays the even numbers from 2 through 20:

```
do i = 2 to 20 by 2
echo %i
enddo
```

DO can also count down, rather than up. If *n* is negative, *varname* will decrease by *n* with each loop, and the loop will stop when *varname* is **less** than *end*. For example, to display the even numbers from 2 through 20 in reverse order, replace the first line of the example above with:

```
do i = 20 to 2 by -2
```

The third type of loop is called a "while loop" or "until loop." DO evaluates the *condition*, which can be any of the tests supported by the IF command, and executes the lines between DO and ENDDO as long as the condition is true. The loop ends when the condition becomes false.

WHILE tests the condition at the start of the loop. Therefore, if the condition is false when the loop starts, the statements within the loop will never be executed, and the batch file will continue with the statement after the ENDDO.

UNTIL tests it at the end. Therefore, if the condition is false when the loop starts, the statements within the loop will still be executed at least once.

The fourth type of loop executes the lines between DO and ENDDO once for every member of a ***set*** (this is similar to the set used in the FOR command). Normally, the set is a list of files specified with wildcards. For example:

```
do x in *.txt
```

will execute the loop once for every *.TXT* file in the current directory; each time through the loop the variable **x** will be set to the name of the next file that matches the file specification.

If, between DO and ENDDO, you create a new file that *could* be included in the list of files, it may or may not appear in an iteration of the DO loop. Whether the new file appears depends on its physical location in the directory structure, a condition over which Take Command has no control.

You can also execute the loop once for each line of text in a file by placing an [**@**] in front of the file name. If you have a file called *DRIVES.TXT* that contains a list of drives on your computer, one drive name per line, you can execute the loop once for each drive this way:

```
do x in @drives.txt
```

To execute the loop once for each line of text in the clipboard, use **CLIP:** as the file name (*e.g.* DO X IN @CLIP:). CLIP: will not return any data unless the clipboard contains text. See Redirection and Piping for more information on CLIP:.

Two special commands, ITERATE and LEAVE, can be used inside a DO / ENDDO loop. ITERATE ignores the remaining lines inside the loop and returns to the beginning of loop for another iteration (unless DO determines that the loop is finished). LEAVE exits from the current DO loop and continues with the line following ENDDO. Both ITERATE and LEAVE are most often used in an IF or IFF command:

```
do while "%var" != "%val1"
```

```
...
if "%var" == "%val2" leave
enddo
```

You can nest DO loops up to 15 levels deep.

The DO and ENDDO commands must be on separate lines, and cannot be placed within a command group, or on the same line as other commands (this is the reason DO cannot be used in aliases). However, commands within the DO loop can use command groups or the command separator in the normal way.

For example, the following command will not work properly, because the DO and ENDDO are inside a command group and are not on separate lines:

```
if "%a" == "%b" (do i = 1 to 10 ^ echo %i ^ enddo)
```

However, this batch file fragment uses multiple commands and command grouping within the DO, and will work properly:

```
do i = 1 to 10
...
if "%x1" == "%x2" (echo Done! ^ leave)
...
enddo
```

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.

DRAWBOX

Purpose: Draw a box on the screen.

Format: **DRAWBOX *ulrow ulcol lrrow lrcol style* [BRIght] *fg* ON [BRIght] *bg* [FILl [BRIght] *bgfill*] [ZOOm] [SHAdow]**

ulrow: Row for upper left corner
ulcol: Column for upper left corner
lrrow: Row for lower right corner
lrcol: Column for lower right corner
style: Box drawing style:
- **0** No lines (box is drawn with blanks)
- **1** Single line
- **2** Double line
- **3** Single line on top and bottom, double on sides
- **4** Double line on top and bottom, single on sides

fg: Foreground character color
bg: Background character color
bgfill: Background fill color (for the inside of the box)

See also: DRAWHLINE and DRAWVLINE.

Usage

DRAWBOX is useful for creating attractive screen displays in batch files.

For example, to draw a box around the edge of an 80x25 window with bright white lines on a blue background:

```
drawbox 0 0 24 79 1 bri whi on blu fill blu
```

See Colors and Color Names for details about colors.

If you use ZOOM, the box appears to grow in steps to its final size. The speed of the zoom operation depends on the speed of your computer and video system.

If you use SHADOW, a drop shadow is created by changing the characters in the row under the box and the 2 columns to the right of the box to normal intensity text with a black background (this will make characters displayed in black disappear entirely).

The row and column values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. The maximum ***row*** value is determined by the current height of the Take Command window. The maximum ***column*** value is determined by the current virtual screen width (see Resizing the Take Command Window for more information).

DRAWBOX checks for valid row and column values, and displays a "Usage" error message if any values are out of range.

Unlike DRAWHLINE and DRAWVLINE, DRAWBOX does **not** automatically connect boxes to existing lines on the screen with the proper connector characters. If you want to draw lines inside a box and have the proper connectors drawn automatically, draw the box first, then use DRAWHLINE and DRAWVLINE

to draw the lines.

DRAWBOX uses the standard line and box drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or if you use a font which does not include these line drawing characters, the box or lines may not appear on your screen as you expect. They will only appear correctly if you have configured Take Command to use a font, such as Terminal, which contains standard extended ASCII characters.

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 25 line by 80 column display, valid *rows* are 0 - 24 and valid *columns* are 0 - 79. The maximum *row* value is determined by the current height of the Take Command window. The maximum *column* value is determined by the current virtual screen width (see Resizing the Take Command Window for more information).

DRAWHLINE checks for valid row and column values, and displays a "Usage" error message if any values are out of range.

See Colors and Color Names for details about colors.

DRAWHLINE uses the standard line and box drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or if you use a font which does not include these line drawing characters, the box or lines may not appear on your screen as you expect. They will only appear correctly if you have configured Take Command to use a font, such as Terminal, which contains standard extended ASCII characters.

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 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. The maximum *row* value is determined by the current height of the Take Command window. The maximum *column* value is determined by the current virtual screen width (see Resizing the Take Command Window for more information).

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

See Colors and Color Names for details about colors.

DRAWVLINE uses the standard line and box drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or if you use a font which does not include these line drawing characters, the box or lines may not appear on your screen as you expect. They will only appear correctly if you have configured Take Command to use a font, such as Terminal, which contains standard extended ASCII characters.

ECHO and ECHOERR

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

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

ECHOERR *message*

message: Text to display.

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

Usage

Take Command has a separate echo capability for batch files and for the command line. The command-line ECHO state is independent of the batch file ECHO state; changing ECHO in a batch file has no effect on the display at the command prompt, and vice versa.

To see the current echo state, use the ECHO command with no arguments. This displays either the batch file or command-line echo state, depending on where the ECHO command is performed.

In a batch file, if you turn ECHO on, each line of the file is displayed before it is executed. If you turn ECHO off, each line is executed without being displayed. ECHO can also be used in a batch file to display a message on the screen. Regardless of the ECHO state, a batch file line that begins with the [**@**] character will not be displayed. To turn off batch file echoing, without displaying the ECHO command, use this line:

```
@echo off
```

ECHO commands in a batch file will send messages to the screen while the batch file executes, even if ECHO is set OFF. For example, this line will display a message in a batch file:

```
echo Processing your print files...
```

If you want to echo a blank line from within a batch file, enter:

```
echo.
```

You cannot use the command separator character [**^**], or the redirection symbols [**| > <**] in an ECHO message, unless you enclose them in quotes (see Argument Quoting) or precede them with the escape character.

ECHO defaults to ON in batch files. The current ECHO state is inherited by called batch files. You can change the default setting to ECHO OFF with the SETDOS /V0 command, the configuration dialogs, or the or the BatchEcho directive in the *TCMD.INI* file.

If you turn the command-line ECHO on, each command will be displayed before it is executed. This will let you see the command line after expansion of all aliases and variables. The command- line ECHO is most useful when you are learning how to use advanced features. This example will turn command-line echoing on:

```
c:\> echo on
```

ECHO defaults to OFF at the command line.

ECHOERR acts like ECHO but sends its output to the standard error device STDERR (usually the screen) instead of the standard output device. If the standard output of a batch file is redirected to a file or another device with >, ECHOERR will still generate a screen message. See Redirection and Piping for more information about the standard output and standard error devices and redirection.

ECHOS and ECHOSERR

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

Format: **ECHOS *message***

ECHOSERR *message*

message: Text to display.

See also: ECHO, SCREEN, SCRPUT, TEXT, and VSCRPUT.

Usage

ECHOS is useful for text output when you don't want to add a carriage return / linefeed pair at the end of the line. For example, you can use ECHOS when you need to redirect control sequences to your printer; this example sends the sequence **Esc P** to the printer on LPT1 (**%=** is translated to the Take Command escape character, and **%=e** to an ASCII Esc).

```
c:\> echos %=eP > lpt1:
```

You cannot use the command separator character [**^**] or the redirection symbols [**| > <**] in an ECHOS message, unless you enclose them in quotes (see Argument Quoting) or precede them with the escape character.

ECHOS does not translate or modify the message text. For example, carriage return characters are not translated to CR/LF pairs. ECHOS sends only the characters you enter (after escape character and back quote processing). The only character you cannot put into an ECHOS message is the NUL character (ASCII 0).

ECHOSERR acts like ECHOS but sends its output to the standard error device (usually the screen) instead of the standard output device. If the standard output of a batch file is redirected to a file or another device with >, ECHOSERR will still generate a screen message. See Redirection and Piping for more information about the standard output and standard error devices and redirection.

ENDLOCAL

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

Format: **ENDLOCAL**

See also: SETLOCAL.

Usage

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

For example, this batch file fragment saves everything, removes all aliases so that user aliases will not affect batch file commands, changes the disk and directory, changes the command separator, runs a program, and then restores the original values:

```
setlocal
unalias *
cdd d:\test
setdos /c~
program ~ echo Done!
endlocal
```

SETLOCAL and ENDLOCAL can only be used in batch files, not in aliases or from the command line.

ESET

Purpose: Edit environment variables and aliases.

Format: **ESET [/A]** ***variable name*****...**

variable name: The name of an environment variable or alias to edit.

/A(lias)

See also: ALIAS, UNALIAS, SET, and UNSET.

Usage

ESET allows you to edit environment variables and aliases using line editing commands (see Command-Line Editing).

For example, to edit the executable file search path:

```
c:\> eset path
path=c:\;c:\dos;c:\util
```

To create and then edit an alias:

```
c:\> alias d = dir /d/j/p
c:\> eset d
d=dir /d/j/p
```

ESET will search for environment variables first and then aliases. If you have an environment variable and an alias with the same name, ESET will edit the environment variable and ignore the alias unless you use the **/A** option.

Environment variable and alias names are normally limited to 80 characters. Their values are normally limited to 255 characters. However, if you use special techniques to create a longer environment variable, ESET will edit it provided the variable contains no more than 511 characters of text.

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 and File Exclusion Ranges.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, size, or file exclusion 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
```

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 Take Command file processing commands.

File exclusion ranges provide a faster and more flexible method of excluding files from internal commands, and do not manipulate file attributes, as EXCEPT does. However, exclusion ranges can only be used with internal commands; you must use EXCEPT for external commands.

Date, time,and size ranges can be used immediately after the word EXCEPT to further qualify which files should be excluded from the ***command***. If the ***command*** is an internal command that supports ranges, an independent range can also be used in the ***command*** itself. You can also use a file exclusion range within the EXCEPT command; however, this will select files to be **excluded** from EXCEPT, and therefore **included** in execution of the ***command***.

You can use command grouping to execute multiple *commands* with a single EXCEPT. For example, the following command copies all files in the current directory whose extensions begin with *.DA*, except the *.DAT* files, to the *D:\SAVE* directory, then changes the first two characters of the extension of the copied files to *.SA*:

```
c:\data> except (*.dat) (copy *.da* d:\save & ren *.da* *.sa*)
```

If you use filename completion (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: Exit the current Take Command session.

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

value: The numeric exit code to return.

Usage

EXIT terminates the current copy of Take Command.

To close the session, or to return to the application that started Take Command, type:

```
c:\> exit
```

If you specify a value, EXIT will return that value to the program that started Take Command. For example:

```
c:\> exit 255
```

The *value* is a number you can use to inform the program of some result, such as the success or failure of a batch file. It can range from 0 - 65,535.

FFIND

Purpose: Search for files by name or contents.

Format: **FFIND [/A[[:][-]rhsda] /B /C /D[list] /E /I /K /L /M /O[[:][-]acdeginrsu] /P /R /S /T"xx" /V /X["xx xx ...]] file...**

file: The file, directory, or list of files or directories to display.

/A(ttribute select)	**/M** (no footers)
/B(are)	**/O**(rder)
/C(ase sensitive)	**/P**(ause)
/D(rive)	**/R**(everse)
/E (upper case display)	**/S**(ubdirectories)
/E (upper case display)	**/T"xx"** (text search string)
/K (no headers)	**/V** (all matching lines)
/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.

Most of the functions provided by FFIND are also available in the Find Files / Text dialog, accessible from the Utilities menu. You can use the FFIND command, the dialog, or both, depending on your needs.

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 inthe current directory, you could use the command

```
c:\> ffind *.btm
```

The output from this command is a list of full pathnames, followed by the number of files found.

If you want to limit the output to a list of *.BTM files which contain the string *color*, you could use this command instead:

```
c:\> ffind /t"color" *.btm
```

The output from this command is a list of files that contain the string *color* along with the first line in each file that contains that string. By default, FFIND uses a case-insensitive search, so the command above will include files that contain *COLOR*, *Color*, *color*, or any other combination of upper-case and lower-case letters.

If you would rather see the **last** line of each file that contains the search string, use the **/R** option, which forces FFIND to search from the end of each file to the beginning. This option will also speed up searches somewhat if you are looking for text that will normally be at the end of a file, such as a signature line:

```
c:\> ffind /r /t"Sincerely," *.txt
```

You can use Take Command's extended wildcards in the search string to increase the flexibility of FFIND's search. For example, the following command will find .TXT files which contain either the string *June* or *July*. It will also find *Juny* and *Jule*. The **/C** option makes the search case-sensitive:

```
c:\> ffind /c/t"Ju[nl][ey]" *.txt
```

If you want to search for text that contains wildcard characters (*****, **?**, **[**, or **]**), you can use the **/I** option to force FFIND to interpret these as normal characters instead of wildcards. The following command, for example, finds all *.TXT* files that contain a question mark:

```
c:\> ffind /i/t"?" *.txt
```

At times, you may need to search for data that cannot be represented by ASCII characters. You can use FFIND's **/X** option to represent the search string in hexadecimal format (this option also changes the output to show hexadecimal offsets rather than text lines). With the /X option, the search must be represented by pairs of hexadecimal digits separated by spaces (in the example below, 41 63 65 is the hex code for "Ace**"**):

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

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

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is optional. The attributes are:

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

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 search. For example, **/A:RHS** will display only those files with all three attributes set.

/B (Bare) Display file names only and omit the text that matches the search. This option is only useful in combination with **/T** or **/X**, which normally force FFIND to display file names and matching text.

/C (Case sensitive) Perform a case-sensitive search. This option is only valid with **/T**, which defaults to a case-insensitive search. It is not needed with a **/X** hexadecimal search, which is always case-sensitive.

/D (Drive) Search all files on one or more drives. If you use **/D** without a list of drives, FFIND will search the drives specified in the list of files. If no drive letters are listed, FFIND will search all of the current drive. You can include a list of drives or a range of drives to search as part of the **/D** option. For example, to search drives C:, D:, E:, and G:, you can use either of these commands:

```
 [c:\> ffind /dcdeg ...
c:\> ffind /dc-eg ...
```

Drive letters listed after **/D** will be ignored when processing ***file*** names which also include a

drive letter. For example, this command displays all the *.BTM* files on C: and E:, but only the *.BAT* files on D:

```
c:\> ffind /s /dce *.btm d:\*.bat
```

/E (Upper case display) Display matching filenames in upper-case characters.

/I (Ignore wildcards) Only meaningful when used in conjunction with the **/T "text"** option. Suppresses the recognition of wildcard characters in the search text. This option is useful if you need to search for characters that would normally be interpreted as wildcards: *****, **?**, **[**, and **]**.

/K (No headers) Suppress the display of the header or filename for each matching text line.

/L (Line numbers) Include the line number for each text line displayed.

/M (No footers) Suppress the footer (the number of files and number of matches) at the end of FFIND's display.

/O (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
- **c** Sort by compression ratio (the least compressed file in the list will be displayed first).
- **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

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/R (Reverse) Only meaningful when used in conjunction with the **/T "text"** or **/X** options. Searches each file from the end backwards to the beginning. This option is useful if you want to display the last occurrence of the search string in each file instead of the first (the default). It may also speed up searches for information that is normally at the end of a file, such as a signature.

/S (Subdirectories) Display matches from the current directory and all of its subdirectories.

/T"text" (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 (All matching lines) Show every matching line (only valid with /T or /X). FFIND's default behavior is to show only the first matching line then and then go on to the next file. This option is only valid with **/T** or **/X**.

/X["xx xx ..."] (Hexadecimal display / search) Specify hexadecimal display and an optional hexadecimal search string.

If **/X** is followed by one or more pairs of hexadecimal digits in quotes (*e.g.*, /x"44 63 65"), FFIND will search for that exact sequence of characters or data bytes without regard to the meaning of those bytes as text. If those bytes are found, the offset is displayed (in both decimal and hexadecimal). A search of this type will always be case-sensitive.

If **/X** is **not** followed by a hexadecimal search string it must be used in conjunction with **/T**, and will change the output format to display offsets (in both decimal and hexadecimal) rather than actual text lines when the search string is found. For example, this command uses **/T** to display the first line in each BTM file containing the word "hello":

```
c:\> ffind /t"hello" *.btm
- c:\test.btm:
echo hello

    1 line in 1 file
```

If you use the same command with **/X**, the offset is displayed instead of the text:

```
c:\> ffind /t"hello" /x *.btm
- c:\test.btm:
Offset: 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] /D /F ["*options*"] /H /L /R [*path*]] *%var* IN ([@]*set* | *start, step, end*) [DO] *command ...***

options: Parsing options for a "file parsing" **FOR**.
path: The starting directory for a "recursive" **FOR**.
%var: The variable to be used in the command ("FOR variable").
set: A set of values for the variable.
start: The starting value for a "counted" **FOR**.
step: The increment value for a "counted" **FOR**.
end: The limit value for a "counted" **FOR**.
command: A command or group of commands to be executed for each value of the variable.

/A: (Attribute select)	**/H**(ide dots)
/D(isable '/' processing)	**/L** (counted loop)
/F(ile parsing)	**/R**(ecursive)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. 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. The members of the set can be a list of file names, text strings, a group of numeric values, or text read from a list of files.

When the ***set*** is made up of text or several separate file names (not an include list), the elements must be separated by spaces, tabs, commas, or the switch character (normally a slash [*/*]).

FOR includes a large number of options, some of which duplicate functions available in other commands, and / or do not follow conventions you may find in our other commands. Most of these extra options are included for compatibility with Windows NT 4.0's *CMD.EXE*. However, we make them available in all three versions of Take Command so that aliases and batch files which use them can work under all products.

The first three sections below (***Working with Files***, ***Working with Text***, and ***Retrieving Text from Files***) describe the traditional FOR command and the enhancements to it which are part of Take Command. The sections on ***Parsing Text from Files*** and ***Counted FOR Loops*** describe features added for compatibility with Windows NT 4.0. The section entitled ***Other Notes*** contains information you may need if you use any aspect of the FOR command extensively.

Working with Files

Normally, the set is a list of files specified with wildcards. For example, if you use this line in a batch file:

```
for %x in (*.txt) do list %x
```

then LIST will be executed once for each file in the current directory with the extension *.TXT*. The FOR variable %x is set equal to each of the file names in turn, then the LIST command is executed for each file. (You could do the same thing more easily with a simple LIST *.TXT. We used FOR here so you could get a feel for how it operates, using a simple example.) Many of the examples in this section are constructed in the same way.)

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

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

FOR supports wildcards and extended wildcards, as well as extended parent directory names (*e.g.*, **...*.txt** to process all of the *.TXT* files that are contained in the directory 2 levels above the current directory).

If the set includes filenames, the file list can be further refined by using date, time, size and file exclusion ranges. The range or ranges must be placed immediately after the word FOR. Ranges will be ignored if no wildcards are used inside the parentheses. For example, this set is made up of all of the *.TXT files that were created or updated on October 4, 1997:

```
for /[d10-4-97,+0] %x in (*.txt) do ...
```

If the *command* is an internal command that supports ranges, an independent range can also be used in the *command* itself.

You can also refine the list by limiting it with the **/A:** option to select only files that have specific attributes.

By default, FOR works only with files in the current directory or a specified directory. With the **/R** option, FOR will also search for files in subdirectories. For example, to work with all of the *.TXT* files in the current directory and its subdirectories:

```
for /r %x in (*.txt) do ...
```

If you specify a directory name immediately after **/R**, FOR will start in that directory and then search each of its subdirectories. This example works with all of the *.BAK* files on drive D:

```
for /r d:\ %x in (*.bak) do ...
```

When you use wildcards to specify the ***set***, FOR scans the directory and finds each file which matches the wildcard name(s) you specified. If, during the processing of the FOR command, you create a file that could be included in the ***set***, it may or may not appear in a future iteration of the same FOR command. Whether the new file appears depends on its physical location in the directory structure. For example, if you use FOR to execute a command for all *.TXT* files, and the command also creates one or more new *.TXT* files, those new files may or may not be processed during the current FOR command, depending on where they are placed in the physical structure of the directory. This is an operating system constraint over which Take Command has no control. Therefore, in order to achieve consistent results you should construct FOR commands which do not create files that could become part of the ***set*** for the current command.

Working with Text

The set can also be made up of text instead of file names. For example, to create three files named file1, file2, and file3, each containing a blank line:

```
for %suffix in (1 2 3) do echo. > file%suffix
```

You could also use the names of environment variables as the text. This example displays the name and

content of several variables from the environment (see the general discussion of the Environment for details on the use of square brackets when expanding environment variables):

```
for %var in (path prompt comspec) do echo %var=%[%var]
```

Retrieving Text from Files

FOR can extract text from files in two different ways. The first method extracts each line from each file in the ***set*** and places it in the variable. To use this method, place an [**@**] at the beginning of the ***set***, in front of the file name or names.

For example, if you have a file called *DRIVES.TXT* that contains a list of drives on your computer, one drive name per line (with a ":" after each drive letter), you can print the free space on each drive this way:

```
for %d in (@drives.txt) do free %d > prn
```

Because the [**@**] is also a valid filename character, FOR first checks to see if the file exists with the [**@**] in its name (*i.e.*, a file named @*DRIVES.TXT*). If so, the filename is treated as a normal argument. If it doesn't exist, FOR uses the filename (without the [**@**]) as the file from which to retrieve text.

If you use @CON as the filename, FOR will read from standard input (a redirected input file) or from a pipe. If you use @CLIP: as the filename, FOR will read any text available from the Windows clipboard. See Redirection and Piping for more information on these features.

Parsing Text from Files

The second method of working with text from files is to have FOR parse each line of each file for you. To begin a "file-parsing" FOR, you must use the **/F** option and then include one or more file names in the ***set***. When you use this form of FOR, the variable must be a single letter, for example, *%a*.

This method of parsing, included for compatibility with Windows NT 4.0's *CMD.EXE*, can be cumbersome and inflexible. For a more powerful method, use FOR with **@filename** as the ***set*** to retrieve each line from the file, as described in the previous section. Then use variable functions like **@INSTR**, **@LEFT**, **@RIGHT**, and **@WORD** to parse the line (see Variable Functions for information on variable functions).

By default, FOR will extract the first word or ***token*** from each line and return it in the variable. For example, to display the first word on each line in the file *FLIST.TXT*:

```
for /f %a in (flist.txt) do echo %a
```

You can control the way FOR /F parses each line by specifying one or more parsing options in a quoted string immediately after the **/F**. The available options are:

> **skip=n**: FOR /F will skip "n" lines at the beginning of each file before parsing the remainder of the file.
>
> **tokens=n, m, ...:** By default, FOR /F returns just the first word or "token" from each parsed line in the variable you named. You can have it return more than one token in the variable, or return tokens in several variables, with this option.
>
> This option is followed by a list of numbers separated by commas. The first number tells FOR /F which token to return in the first variable, the second number tells it which to return in the second variable, etc. The variables follow each other alphabetically starting with the variable you name on the FOR command line. This example returns the first word of each line in each text file in **%d**, the second in **%e**, and the third in **%f**:

```
for /f "tokens=1,2,3" %d in (*.txt) do ...
```

You can also indicate a range of tokens by separating the numbers with a hyphen [**-**]. This example returns the first word of each line in **%p**, the second through fifth words in **%q**, and the eighth word in **%r**:

```
for /f "tokens=1,2-5,8" %p in (*.txt) do ...
```

To return the rest of the line in a variable, use a range that ends with a number higher than the last token in any line, such as 999. This final example returns the first word of each line in **%a** and the remainder of each line (assuming that no line has more than 999 words!) in **%b**:

```
for /f "tokens=1,2-999" %a in (*.txt) do ...
```

eol=c: If FOR /F finds the character "c" in the line, it will assume that the character and any text following it are part of a comment and ignore the rest of the line.

delims=xxx..: By default, FOR /F sees spaces and tabs as word or token delimiters. This option replaces those delimiters with all of the characters following the equal sign to the end of the string. This option must therefore be the last one used in the quoted options string.

You can also use FOR /F to parse a single string instead of each line of a file by using the string, in quotes, as the ***set.*** For example, this command will assign variable A to the string "this", B to "is", etc., then display "this" (enter the command on one line):

```
for /f "tokens=1,2,3,4" %a in ("this is a test") do echo %a
```

"Counted" FOR Loop

The "counted FOR" loop is included only for compatibility with Windows NT 4.0's *CMD.EXE*. In most cases, you will find the DO command more useful for performing counted loops.

In a counted FOR command, the ***set*** is made up of numeric values instead of text or file names. To begin a counted FOR command, you must use the **/L** option and then include three values, separated by commas, in the ***set***. These are the ***start***, ***step***, and ***end*** values. During the first iteration of the FOR loop, the variable is set equal to the ***start*** value. Before each iteration, the variable is increased by the ***step*** value. The loop ends when the variable exceeds the ***end*** value. This example will print the numbers from 1 to 10:

```
for /l %val in (1,1,10) do echo %val
```

This example will print the odd numbers from 1 to 10 (1, 3, 5, 7, and 9):

```
for /l %val in (1,2,10) do echo %val
```

The ***step*** value can be negative. If it is, the loop will end when the variable is less than the ***end*** value.

Other Notes

You can use either *%* or *%%* in front of the variable name. Either form will work, whether the FOR command is typed from the command line or is part of an alias or batch file (some traditional command processors require a single *%* if FOR is used at the command line, but require *%%* if FOR is used in a batch file). The variable name can be up to 80 characters long. The word DO is optional.

If you use a single-character FOR variable name, that name is given priority over any environment variable which starts with the same letter, in order to maintain compatibility with the traditional FOR

command. For example, the following command tries to add a: and b: to the end of the PATH, but will not work as intended:

```
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.

The following example uses FOR with variable functions to delete the *.BAK* files for which a corresponding *.TXT* file exists in the current directory (this should be entered on one line):

```
for %file in (*.txt) do del %@name[%file].bak
```

You can use command grouping to execute multiple commands for each element in the ***set***. For example, the following command copies each *.WKQ* file in the current directory to the *D:\WKSAVE* directory, then changes the extension of each file in the current directory to *.SAV*:

```
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. For compatibility with *COMMAND.COM* and *CMD.EXE*, a single-character FOR variable is created in a special way that does not overwrite an existing environment variable with the same name. When using a multi-character variable name you must be careful not to use the name of one of your environment variables as a FOR variable. For example, a command that begins

```
c:\> for %path in ...
```

will write over your current path setting, then erase the path variable completely when FOR is done.

FOR statements can be nested.

Options

/A: (Attribute select) Process only those files that have the specified attribute(s). **/A:** will be used only when processing wildcard file names in the set. It will be ignored for filenames without wildcards or other items in the set. Preceding the attribute character with a hyphen

[**-**] will process files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

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!
```

/D (Disable '/' processing) Disable special processing of the '/' in the FOR argument group. For compatibility with certain versions of *COMMAND.COM* (in those prior to Windows 95), 4DOS and Take Command/16 normally treat a forward slash inside the ***set*** as an "escape" character, discard the slash, and return the character after the slash, followed by the remainder of the string. This behavior can be used in batch files to separate a string into individual characters (although Take Command provides a much easier method with the @INSTR and @LEFT variable functions).

The **/D** option must follow the FOR keyword and come before the variable name.

These examples show the effects of **/D**:

```
c:\> for %s in (/abcdef) do echo %s
a
bcdef

c:\> for /d %s in (/abcdef) do echo %s
/abcdef
```

/H (Hide dots) Suppress the assignment of the "." and ".." directories to the FOR variable.

/L (Counted loop) Interpret the three values in the ***set*** as the ***start***, ***step***, and ***end*** values of a counted loop. See the details under Counted FOR Loop, above.

/R (Recursive) Look in the current directory and all of its subdirectories for files in the ***set***. If the **/R** is followed by a directory name, look for files in that directory and all of its subdirectories.

FREE

Purpose: Display the total disk space, total bytes used, and total bytes free on the specified (or default) drive(s).

Format: **FREE [*drive:* ...]**

drive: One or more drives to include in the report.

See also: MEMORY.

Usage

FREE provides the same disk information as the external command CHKDSK, but without the wait, since it does not check the integrity of the file and directory structure of the disk.

A colon [**:**] is required after each drive letter. This example displays the status of drives A and C:

```
c:\> free a: c:
```

If the volume serial number is available, it will appear after the drive label or name.

DOS networks with large server disk drives (over 2 GB) may report disk space values that are too small when FREE is used. If this occurs, it is because the network software does not report the proper values to Take Command.

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 one subdirectory. Even if you use **/I**, GLOBAL will normally halt execution if Take Command receives a **Ctrl-C** or **Ctrl-Break**.

/P (Prompt) Forces GLOBAL to prompt with each directory name before it performs the command. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Do not display the directory names as each directory is processed.

GOSUB

Purpose: Execute a subroutine in the current batch file.

Format: **GOSUB *label***

label: The batch file label at the beginning of the subroutine.

See also: CALL, GOTO and RETURN.

Usage

GOSUB can only be used in batch files.

Take Command allows subroutines in batch files. A subroutine must start with a *label* (a colon [**:**] followed by a label name) which appears on a line by itself. Case differences are ignored when matching labels. Labels may be one or more words long. The subroutine must end with a RETURN statement.

The subroutine is invoked with a GOSUB command from another part of the batch file. After the RETURN, processing will continue with the command following the GOSUB command. For example, the following batch file fragment calls a subroutine which displays the directory and returns:

```
echo Calling a subroutine
gosub subr1
echo Returned from the subroutine
quit
:subr1
dir /a/w
return
```

GOSUB searches the entire batch file for the *label*, starting at the beginning of the file. If the label does not exist, the batch file is terminated with the error message "Label not found."

GOSUB saves the IFF and DO states, so IFF and DO statements inside a subroutine won't interfere with statements in the part of the batch file from which the subroutine was called.

Subroutines can be nested.

GOTO

Purpose: Branch to a specified line inside the current batch file.

Format: **GOTO [/I]** ***label***

label: The batch file label to branch to.

/I(FF and DO continue)

See also: GOSUB.

Usage

GOTO can only be used in batch files.

After a GOTO command in a batch file, the next line to be executed will be the one immediately after the *label*. The *label* must begin with a colon [**:**] and appear on a line by itself. The colon is required on the line where the label is defined, but is not required in the GOTO command itself. Case differences are ignored when matching labels. Labels may be one or more words long.

This batch file fragment checks for the existence of the file *CONFIG.SYS*. 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.sys goto C_EXISTS
echo CONFIG.NT doesn't exist - exiting.
quit
:C_EXISTS
copy *.* a:\
```

GOTO searches the entire batch file for the *label*, starting at the beginning of the file. If the label does not exist, the batch file is terminated with the error message "Label not found."

To avoid errors in the processing of nested statements and loops, GOTO cancels all active IFF statements and DO / ENDDO loops unless you use **/I**. This means that a normal GOTO (without **/I**) may not branch to any label that is between an IFF and the corresponding ENDIFF or between a DO and the corresponding ENDDO.

For compatibility with Windows NT 4.0's *CMD.EXE*, the command

```
GOTO :EOF
```

will end processing of the current batch file if the label :EOF does not exist. However, this is less efficient than using the QUIT or CANCEL command to end a batch file.

Option

/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.

See also: The Take Command Help System.

Usage

Online help is available for Take Command. This Take Command help system uses the Windows help facility.

If you type the command HELP by itself (or press **F1** when the command line is empty), the table of contents is displayed. If you type HELP plus a topic name, that topic is displayed. For example,

```
help copy
```

displays information about the COPY command and its options.

HISTORY

Purpose: Display, add to, clear, or read the history list.

Format: **HISTORY [/A *command* /F /P /R *filename*]**

command: A command to be added to the history list.
filename: The name of a file containing entries to be added to the history list.

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

See also: DIRHISTORY and LOG.

Usage

Take Command keeps a list of the commands you have entered on the command line. See Command History and Recall for additional details.

The HISTORY command lets you view and manipulate the command history list directly. If no parameters are entered, HISTORY will display the current command history list:

```
c:\> history
```

With the options explained below, you can clear the list, add new commands to the list without executing them, save the list in a file, or read a new list from a file.

The number of commands saved in the history list depends on the length of each command line. The history list size can be specified at startup from 256 to 32767 characters (see the History directive). The default size is 1024 characters.

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 configuration dialogs or the HistMin directive in the *TCMD.INI* file.

You can save the history list by redirecting the output of HISTORY to a file. This example saves the command history to a file called *HISTFILE* and reads it back again immediately. If you leave out the HISTORY /F command on the second line, the contents of the file will be appended to the current history list instead of replacing it:

```
c:\> history > histfile
c:\> history /f
c:\> history /r histfile
```

If you need to save your command history at the end of each day's work, you might use the first of these commands in your *TCSTART.BTM* or other startup file, and the second in *TCEXIT.BTM*:

```
if exist c:\histfile history /r c:\histfile
history > c:\histfile
```

This restores the previous history list if it exists, and saves the history when Take Command exits.

Options

/A (Add) Add a command to the history list. This performs the same function as the **Ctrl-K** key at the command line (see Command History and Recall).

/F (Free) Erase all entries in the command history list.

/P (Prompt) Wait for a key after displaying each page of the list. Your options at the prompt are explained in detail under Page and File Prompts.

/R (Read) Read the command history from the specified file and append it to the history list currently held in memory. Each line in the file must fit within the command-line length limit).

If you are creating a HISTORY /R file by hand, and need to create an entry that spans multiple lines in the file, you can do so by terminating each line, except the last, with an escape character. However, you cannot use this method to exceed the command-line length limit.

IF

Purpose: Execute a command if a condition or set of conditions is true.

Format: **IF [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 and @IF.

Usage

IF is normally used only in aliases and batch files. It is always followed by one or more *conditions* and then a *command*. First, the *conditions* are evaluated. If they are true, the *command* is executed. Otherwise, the *command* is ignored. If you add a NOT before a *condition*, the *command* is executed only when the *condition* is false.

You can link *conditions* with **.AND.**, **.OR.**, or **.XOR.**, and you can group conditions with parentheses (see Combining Tests below). You can also nest IF statements.

The *conditions* can test strings, numbers, the existence of a file or subdirectory, the exit code returned by the preceding external command, and the existence of aliases and internal commands.

The *command* can be an alias, an internal command, an external command, or a batch file. The entire IF statement, including all *conditions* and the *command*, must fit on one line.

Some examples of IF conditions and commands are included below; additional examples can be found in the *EXAMPLES.BTM* file which came with Take Command.

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.)

For example, if DuplicateBugs is set to Yes (the default), the following command will display nothing, because the second ECHO command is discarded along with the first when the *condition* fails. If DuplicateBugs is set to No, it will display "hello":

```
[c:\] if 1 == 2 echo Wrong! & echo hello
```

Conditions

The conditional tests listed in the following sections are available in both the IF and IFF commands. They fit into two categories: string and numeric tests, and status tests. The tests can use environment variables, internal variables and variable functions, file names, literal text, and numeric values as their arguments.

String and Numeric Tests

Six test conditions can be used to test character strings. The same conditions are available for both

numeric and normal text strings (see below for details). In each case you enter the test as:

```
string1 operator string2
```

The **operator** defines the type of test (equal, greater than or equal, and so on). You should always use spaces on both sides of the **operator**. The operators are:

EQ or **==**	*string1* equal to *string2*
NE or **!=**	*string1* not equal to *string2*
LT	*string1* less than *string2*
LE	*string1* less than or equal to *string2*
GE	*string1* greater than or equal to *string2*
GT	*string1* greater than *string2*

When IF compares two character strings, it will use either a **numeric** comparison or a **string** comparison. A numeric comparison treats the strings as numeric values and tests them arithmetically. A string comparison treats the strings as text.

The difference between numeric and string comparisons is best explained by looking at the way two values are tested. For example, consider comparing the values 2 and 19. Numerically, 2 is smaller, but as a string it is "larger" because its first digit is larger than the first digit of 19. So the first of these *conditions* 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 a string begins with a decimal separator it is not considered numeric unless the next character is a digit, and there are no more decimal separators within the string. For example, ".07" is numeric, but ".a" and ".07.01" are not.) If either value is non-numeric, a string comparison is used. If either value is non-numeric, a string comparison is used. To force a string comparison when both values are or may be numeric, use double quotes around the values you are testing, as shown above. Because the double quote is not a numeric character, IF performs a string comparison.

Case differences are ignored in string comparisons. If two strings begin with the same text but one is shorter, the shorter string is considered to be "less than" the longer one. For example, "a" is less than "abc", and "hello_there" is greater than "hello".

When you compare text strings, you should always enclose the arguments in double quotes in order to avoid syntax errors which may occur if one of the argument values is empty.

Numeric comparisons work with both integer and decimal values. The values to be compared must contain only numeric digits, decimal points, and an optional sign (**+** or **-**). The number may may contain up to 16 digits to the left of the decimal point, and 8 digits to the right.

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 command runs a program called *FASTVER* if the system CPU is a Pentium or Pentium Pro:

```
if %_cpu ge 586 fastver
```

This batch file fragment tests for a string value:

```
input "Enter your selection : " %%cmd
if "%cmd" == "WP" goto wordproc
if "%cmd" NE "GRAPHICS" goto badentry
```

This example calls *GO.BTM* if the first two characters in the file *MYFILE* are "GO" (enter this example on one line):

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

Status Tests

These conditions test the system or command processor status. You can use internal variables and variable functions to test many other parts of the system status.

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.

Do not use IF EXIST to test for existence of a directory (use IF ISDIR instead). Due to variations in operating system internals, IF EXIST will not return consistent results when used to test for the existence of a directory.

ISALIAS aliasname

If the name is defined as an alias, the condition is true.

ISDIR | DIREXIST path

If the subdirectory exists, the condition is true. For compatibility with Novell DOS / OpenDOS, DIREXIST may be used as a synonym for ISDIR.

ISINTERNAL command

If the specified command is an active internal command, the condition is true. Commands can be activated and deactivated with the SETDOS /I command.

ISLABEL label

If the specified batch file label exists, the condition is true. Labels may be one or more words long.

ISWINDOW "title"

If a window which matches the title exists, the condition is true. Double quotes must be used around the title, which may contain wildcards and extended wildcards.

The first batch file fragment below tests for the existence of *A:\JAN.DOC* before copying it to drive C (this avoids an error message if the file does not exist):

```
if exist a:\jan.doc copy a:\jan.doc c:\
```

This example tests the exit code of the previous program and stops all batch file processing if an error occurred:

```
if errorlevel == 0 goto success
echo "External Error  Batch File Ends!"
cancel
```

Combining Tests

You can negate the result of any test with **NOT**, and combine tests of any type with **.AND.**, **.OR.**, and **.XOR.**

When two tests are combined with **.AND.**, the result is true if both individual tests are true. When two tests are combined with **.OR.**, the result is true if either (or both) individual tests are true. When two tests are combined with **.XOR.**, the result is true only if one of the tests is true and the other is false.

This example runs a program called *DATALOAD* if today is Monday or Tuesday (enter this on one line):

```
if "%_dow" == "Mon" .or. "%_dow" == "Tue" dataload
```

Test conditions are always scanned from left to right there is no implied order of precedence, as there is in some programming languages. You can, however, force a specific order of testing by grouping conditions with parentheses, for example (enter this on one line):

```
if (%a == 1 .or. (%b == 2 .and. %c == 3)) echo something
```

Parentheses can only be used when the portion of the ***condition*** inside the parentheses contains at least one .and., .or., or .xor.. Parentheses on a simple condition which does not combine two or more tests will be taken as part of the string to be tested, and will probably make the test fail. For example, the first of these IF tests would fail; the second would succeed:

```
if (a == a) ...
if (a == a .and. b == b) ...
```

Parentheses can be nested.

IFF

Purpose: Perform IF / THEN / ELSE conditional execution of commands.

Format: **IFF [NOT]** ***condition*** **[.AND. | .OR. | .XOR. [NOT]** ***condition*** **...] THEN ^** ***commands***

[ELSEIFF ***condition*** **THEN ^** ***commands*** **] ...**

[ELSE ^ ***commands*** **]**

& ENDIFF

condition: A test to determine if the command(s) should be executed.

commands: One or more commands to execute if the condition(s) is true. If you use multiple commands, they must be separated by command separators or be placed on separate lines of a batch file.

See also: IF and @IF

Usage

IFF is similar to the IF command, except that it can perform one set of *commands* when a condition or set of *conditions* is true and a different set of *commands* when the *conditions* are false.

IFF can also execute multiple commands when the *conditions* are true or false; IF normally executes only one command. IFF imposes no limit on the number of commands and is generally a "cleaner" and more structured command than IF.

IFF is always followed by one or more *conditions*. If they are true, the *commands* that follow the word THEN are executed. Additional *conditions* can be tested with ELSEIFF. If none of these *conditions* are true, the *commands* that follow the word ELSE are executed. After the selected *commands* (if any) are executed, processing continues after the word ENDIFF.

If you add a NOT before the condition, the THEN *commands* are executed only when the *condition* is false and the ELSE *commands* are executed only when the *condition* is true.

The *commands* may be separated by command separators, or may be on separate lines of a batch file. You should include a command separator or a line break after a THEN, before an ELSEIFF, and before and after an ELSE.

You can link *conditions* with **.AND.**, **.OR.**, or **.XOR.**, and you can group conditions with parentheses. You can nest IFF statements up to 15 levels deep. The *conditions* can test strings or numbers, the existence of a file or subdirectory, the errorlevel returned from the preceding external command, and the existence of alias names and internal commands.

See the IF command for a list of the possible *conditions*.

The *commands* can include any internal command, alias, external command, or batch file.

The alias in this example checks to see if the argument is a subdirectory. If so, the alias deletes the subdirectory's files and removes it (enter this on one line):

```
c:\> alias prune `iff isdir %1 then ^ del /sxz %1 ^ else ^ echo Not a
   directory! ^ endiff`
```

Be sure to read the cautionary notes about GOTO and IFF under the GOTO command before using a

GOTO inside an IFF statement.

If you pipe data to an IFF, the data will be passed to the command(s) following the IFF, not to IFF itself.

INKEY

Purpose: Get a single keystroke from the user and store it in an environment variable.

Format: **INKEY [/C /D /K"keys" /P /Wn /X] [*prompt*] *%%varname***

prompt: Optional text that is displayed as a prompt.
varname: The variable that will hold the user's keystroke.

/C(lear buffer)	**/P**(assword)
/D(igits only)	**/W**(ait)
/K (valid keystrokes)	**/X** (no carriage return)

See also: INPUT.

Usage

INKEY optionally displays a prompt. Then it waits for a specified time or indefinitely for a keystroke, and places the keystroke into an environment variable. It is normally used in batch files and aliases to get a menu choice or other single-key input. Along with the INPUT command, INKEY allows great flexibility in reading input from within a batch file or alias.

If *prompt* text is included in an INKEY command, it is displayed while INKEY waits for input.

The following batch file fragment prompts for a character and stores it in the variable *NUM*:

```
inkey Enter a number from 1 to 9:  %%num
```

INKEY reads standard input for the keystroke, so it will accept keystrokes from a redirected file or from the KEYSTACK. You can supply a list of valid keystrokes with the **/K** option.

Standard keystrokes with ASCII values between 1 and 255 are stored directly in the environment variable. Extended keystrokes (for example, function keys and cursor keys) are stored as a string in decimal format, with a leading @ (for example, the **F1** key is @59). The **Enter** key is stored as an extended keystroke, with the code @28. See ASCII, Key Codes, and ANSI Codes.for lists of ASCII and extended key codes.

To test for a non-printing ASCII keystroke returned by INKEY use the **@ASCII** function to get the numeric value of the key. For example, to test for **Esc**, which has an ASCII value of 27:

```
inkey Enter a key:  %%key
if "%@ascii[%key]" == "27" echo Esc pressed
```

If you press **Ctrl-C** or **Ctrl-Break** while INKEY is waiting for a key, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job. A batch file can handle **Ctrl-C** and **Ctrl-Break** itself with the ON BREAK command.

Options

/C (Clear buffer) Clears the keyboard buffer before INKEY accepts keystrokes. If you use this option, INKEY will ignore any keystrokes which you type, either accidentally or intentionally, before it is ready to accept input.

/D (Digits only) Prevents INKEY from accepting any keystroke except a digit from 0 to 9.

/**K"keys"** Specify the permissible keystrokes. The list of valid keystrokes should be enclosed in double quotes. For alphabetic keys the validity test is not case sensitive. You can specify extended keys by enclosing their names in square brackets (within the quotes), for example:

```
inkey /k"ab[Ctrl-F9]" Enter A, B, Ctrl-F9 %%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, Take Command will echo the keystroke if possible, beep, move the cursor back one character, and wait for another keystroke.

/P (Password) Prevents INKEY from echoing the character.

/W (Wait) Time-out period, in seconds, to wait for a response. If no keystroke is entered by the end of the time-out period, INKEY returns with the variable unchanged. This allows you to continue the batch file if the user does not respond in a given period of time. You can specify **/W0** to return immediately if there are no keys waiting in the keyboard buffer.

For example, the following batch file fragment waits up to 10 seconds for a character, then tests to see if a "Y" was entered:

```
set netmon=N
inkey /K"YN" /w10 Network monitor (Y/N)?  %%net
iff "%netmon" == "Y" then
  rem Commands to load the monitor program
endiff
```

/X (No carriage return) Prevents INKEY from adding a carriage return and line feed after the user's entry.

INPUT

Purpose: Get a string from the keyboard and save it in an environment variable.

Format: **INPUT [/D /C /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 color)
/D(igits only)	**/P**(assword)
/E(dit)	**/W**(ait)
/L(ength)	**/X** (no carriage return))

See also: INKEY, KEYSTACK, MSGBOX, and QUERYBOX.

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 or from the KEYSTACK.

If you press **Ctrl-C** or **Ctrl-Break** while INPUT is waiting for input, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job. A batch file you can handle **Ctrl-C** and **Ctrl-Break** itself with the ON BREAK command.

You can pipe text to INPUT; if you do, it will set the variable to the first line it receives.

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 INPUT from accepting any keystrokes except digits 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 color) Disables the display colors set by InputColor in the *TCMD.INI* file. With this option, INPUT will use the default display colors instead.

/P (Password) Tells INPUT to echo asterisks, instead of the characters you type.

/W (Wait) Time-out period, in seconds, to wait for a response. If no keystroke is entered by the end of the time-out period, INPUT returns with the variable unchanged. This allows you to continue the batch file if the user does not respond in a given period of time. If you enter a key before the time-out period, INPUT will wait indefinitely for the remainder of the line. You can specify **/W0** to return immediately if there are no keys waiting in the keyboard buffer.

/X (No carriage return) Prevents INPUT from adding a carriage return and line feed after the user's entry.

KEYBD

Purpose: Set the state of the keyboard toggles: Caps Lock, Num Lock, and Scroll Lock.

Format: **KEYBD [/Cn /Nn /Sn]**

n: 0 to turn off the toggle, or 1 to turn on the toggle.

/C(aps lock) **/S**(croll lock)
/N(um lock)

Usage

Most keyboards have 3 toggle keys, the Caps Lock, Num Lock, and Scroll Lock. The toggle key status is usually displayed by three lights at the top right corner of the keyboard.

This command lets you turn any toggle key on or off. It is most useful in batch files and aliases if you want the keys set a particular way before collecting input from the user.

For example, to turn off the Num Lock and Caps Lock keys, you can use this command:

```
c:\> keybd /c0 /n0
```

If you use the KEYBD command with no switches, it will display the present state of the toggle keys.

The toggle key state is typically the same for all sessions, and changes made with KEYBD in one session will therefore affect all other sessions. The only exception is when running under WIN-OS/2, where KEYBD affects only the current Take Command session.

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.

KEYSTACK

Purpose: Feed keystrokes to a program or command automatically.

Format: **KEYSTACK [*!*] [/Wn] [*"abc"*] [*keyname*[n]] ...**

!: Signal to clear the Keystack and the keyboard buffer.
"abc": Literal characters to be placed in the Keystack.
keyname: Name for a key to be placed in the Keystack.
n: Number of times to repeat the named key.

/W(ait)

Usage

KEYSTACK will send the keystrokes to the current active window. If you want to send keystrokes to another program (rather than have them function with Take Command itself), you must start the program or ACTIVATE its window so it can receive the keystrokes. You must do this **before** executing the KEYSTACK command.

KEYSTACK is most often used for programs started from batch files. In order for KEYSTACK to work in a batch file, you must start the program with the START command, then use the KEYSTACK command. If you start the program directly without using START the batch file will wait for the application to complete before continuing and running the KEYSTACK command, and the keystrokes will be placed in the buffer too late.

If you use KEYSTACK in an alias executed from the prompt, the considerations are essentially the same, but depend on whether ExecWait is set. If ExecWait is **not** set, you can use KEYSTACK immediately after an application is started. However, if ExecWait **is** set, the KEYSTACK command will not be executed until the program has finished, and the keystrokes will be placed in the buffer too late.

You may not be able to use KEYSTACK effectively if you have programs running in the background which change the active window (for example, by popping up a dialog box). If a window pops up in the midst of your KEYSTACK sequence, keystrokes stored in the KEYSTACK buffer may go to that window, and not to the application you intended.

Characters entered within double quotes (*"abc"*) will be stored "as is" in the buffer. The only items allowed outside double quotes are key names, the **!** and **/W** options, and a repeat count.

See Keys and key names for a complete listing of key names and a description of the key name and numeric key code format. If you want to send the same key name or numeric code several times, you can follow it with a repeat count in square brackets. For example, to send the Enter key 4 times, you can use this command:

```
keystack enter [4]
```

The repeat count works only with individual keystrokes, or numeric keystroke or character values; it cannot be used with quoted strings.

An exclamation mark [**!**] will clear all pending keystrokes in the KEYSTACK buffer.

For example, to start Word for Windows and open the last document you worked on, you could use the command:

```
d:\doc> start winword ^ keystack /w54 F10 Right Down "1"
```

This runs Word, delays about three seconds (54 clock ticks at about 1/18 second each) for Word to get started, places the keystrokes for F10 (change to the menu bar), right arrow (go to the File menu), down arrow (display the menu), and "1" (open the most recently used file) into the buffer. Word receives these keystrokes and performs the appropriate actions.

You can store a maximum of 255 characters in the KEYSTACK buffer. A delay takes two character "slots" in the buffer. A repeated character takes one character slot per repetition.

Each time the KEYSTACK command is executed, it will clear any remaining keystrokes stored by a previous KEYSTACK command.

You may need to experiment with your programs and insert delays (see the **/W** option) to find the window activation and keystroke sequence that works for a particular program.

In Take Command/16, KEYSTACK will not work under WINOS2 (Windows running under OS/2). It can only be used under "true" Microsoft Windows.

Options

/W (Wait) Delay the next keystroke in the KEYSTACK buffer by a specified number of clock "ticks". A clock tick is approximately 1/18 second. The number of clock ticks to delay should be placed immediately after the **W**, and must be between 1 and 65,535 (65,535 ticks is about 1 hour). You can use the **/W** option as many times as desired and at any point in the string of keystrokes except within double quotes. Some programs may need the delays provided by **/W** in order to receive keystrokes properly from KEYSTACK. The only way to determine what delay is needed is to experiment.

LIST

Purpose: Display a file, with forward and backward paging and scrolling.

Format: **LIST [/A:[[-]rhsda] /H /I /R /S /T /W /X] *file*...**

file: A file or list of files to display.

/A: (Attribute select)
/H(igh bit off)
/I(gnore wildcards)
/R(everse)
/S(tandard input)
/T (search for Text)
/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 fast and flexible way to view a file, without the overhead of loading and using a text editor.

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

```
c:\> list memo.doc
```

LIST is most often used for displaying ASCII text files. It can be used for other files which contain non-alphabetic characters, but you may need to use hex mode (see below) to read these files.

LIST displays files in the Take Command window. The standard tool bar and scroll bars are replaced with the LIST tool bar and scroll bars. Use the scroll bars or cursor pad to scroll through the file. You can select the LIST commands either with the mouse (on the tool bar and scrollbars) or from the keyboard. LIST recognizes the following keys and buttons:

Key	Button	Meaning
Home		Display the first page of the file.
End		Display the last page of the file.
Esc	**Cont**	Exit the current file.
Ctrl-C	**Quit**	Quit LIST.
Ctrl-PgUp		Display previous file.
Ctrl-PgDn		Display next file.
Up Arrow		Scroll up one line.
Down Arrow		Scroll down one line.
Left Arrow		Scroll left 8 columns.
Right Arrow		Scroll right 8 columns.
Ctrl ←		Scroll left 40 columns.
Ctrl →		Scroll right 40 columns.

F1		Display online help.
B	**Back**	Go back to the previous file in the current group of files.
F	**Find**	Prompt and search for a string or a sequence of hexadecimal values.
Ctrl-F		Prompt and search for a string, searching backward from the end of the file.
G	**Goto**	Go to a specific line or, in hex mode, to a specific hexadecimal offset.
H	**High**	Toggle the "strip high bit" (**/H**) option.
I	**Info**	Display information on the current file (the full name, size, date, and time).
N	**Next**	Find next matching string.
Ctrl-N		Find previous matching string in the file.
O	**Open**	Open a new file.
P	**Print**	Print selected text, the current page, or the entire file.
W	**Wrap**	Toggle the "line wrap" (**/W**) option.
X	**heX**	Toggle the hex-mode display (**/X**) option.

Text searches performed with **F**, **N**, **Ctrl-F**, and **Ctrl-N,** or with the corresponding buttons, are not case-sensitive unless you check the "Match case" box in the search dialog. LIST remembers the search strings you have used in the current Take Command session; to select a previous string, use the drop-down arrow to the right of the string entry field (the **N key** and the **Next** button search for the top item in this drop-down list).

When the search string is found LIST displays the line containing the string at the top of the window, and highlights the string it found. Any additional occurrences of the string on the same display page are also highlighted. Highlighting is intended for use with text files; in binary files the search string will be found, but may not be highlighted properly.

If you want to search for specific hexadecimal values check the "Hex search" box, and 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"; see the online help for a complete list of standard ASCII codes). Hexadecimal searches **are** case-sensitive, and search for exactly the string you enter.

You can use extended wildcards in the search string. For example, you can search for the string "to*day" to find the next line which contains the word "to" followed by the word "day" later on the same line, or search for the numbers "101" or "401" with the search string "[14]01". If you begin the search string with a back-quote [`], or enclose it in back-quotes, wildcard characters in the string will be treated as normal text with no special wildcard meaning.

You can use the **/T** switch to specify search text for the first ***file***. When you do so, LIST begins a search as soon as the file is loaded. Use **/I** to ignore wildcards in the initial search string, and **/R** to make the initial search go backwards from the end of the file. When you LIST multiple files with a single LIST command, these switches affect only the first file; they are ignored for the second and subsequent files.

LIST normally allows long lines in the file to extend past the right edge of the screen. You can use the horizontal scrolling keys (see above) to view text that extends beyond the screen width. If you use the **W** command or **/W** switch to wrap the display, each line is wrapped when it reaches the right edge of the screen, and the horizontal scrolling keys are disabled.

To view text from the clipboard, use **CLIP:** as the file to be listed. CLIP: will not return any data unless the clipboard contains text. See Redirection and Piping for more information on CLIP:.

If you print the file which LIST is displaying, 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. Regardless of the display mode, LIST will bring up a standard print dialog which allows you to print selected text, the current page, or the entire file.

Advanced Features

If you specify a directory name instead of a filename as an argument, LIST will display each of the files in that directory.

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

You can set the colors used by LIST with the ListColors directive in the *TCMD.INI* file, or the LIST Colors selection on the Commands page of the configuration dialogs. If ListColors is not used, the LIST display will use the current default colors.

By default, LIST sets tab stops every 8 columns. You can change this behavior with the TabStops *.INI* file directive.

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

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 or the **High** button on the tool bar.

/I (Ignore wildcards) Only meaningful when used in conjunction with the **/T "text"** option. Direct LIST to interpret characters such as *****, **?**, **[**, and **]** as literal characters instead of wildcard characters. **/I** affects only the initial search started by **/T**, not subsequent searches started from within LIST.

/R (Reverse) Only meaningful when used in conjuction with the **/T "text"** option. Directs LIST to search for text from the end of the file instead of from the beginning of the file. Using this switch can speed up searches for text that is normally near the end of the file, such as a signature. **/R** affects only the initial search started by **/T**, not subsequent searches started from within LIST.

/S (Standard input) Read from standard input rather than a file. This allows you to redirect command output and view it with LIST. Normally, LIST will detect input from a redirected command and adjust automatically. However, you may find circumstances when **/S** is required. For example, to use LIST to display the output of DIR you could use either of these commands:

```
c:\> dir | list
c:\> dir | list /s
```

/T (Text) Search for text in the first ***file***. This option is the same as pressing **F**, but it allows you to specify the search text on the command line. The text must be contained in quotation marks if it contains spaces, punctuation, or wildcard characters. For example, to search for the string Take Command in the file *README.DOC*, you can use this command:

```
c:\> list /t"Take Command" readme.doc
```

The search text may include wildcards and extended wildcards. For example, to search for the words Hello and John on the same line in the file *LETTER.DAT*:

```
c:\> list /t"Hello*John" letter.dat
```

When you LIST multiple files with a single LIST command, **/T** only initiates a search in the first file. It is ignored for the second and subsequent files. Also see **/I** and **/R**.

/W (Wrap) Wrap the text at the right edge of the screen. This option is useful when displaying files that don't have a carriage return at the end of each line. The horizontal scrolling keys do not work when the display is wrapped. You can toggle this option on and off from within LIST with the **W** key or the **Wrap** button on the tool bar.

/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 or the **heX** button on the tool bar.

LOADBTM

Purpose: Switch a batch file to or from BTM mode.

Format: **LOADBTM [ON | OFF]**

Usage

Take Command recognizes two kinds of batch files: *.BAT* and *.BTM*. Batch files executing in BTM mode run two to ten times faster than in 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* 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*.BAT* file to BTM mode without changing its extension.

Using LOADBTM to repeatedly switch modes within a batch file is not efficient. In most cases the speed gained by running some parts of the file in BTM mode will be more than offset by the speed lost through repeated loading of the file each time BTM mode is invoked.

LOG

Purpose: Save a log of commands to a disk file.

Format: **LOG [/H /W *file*] [ON | OFF | *text*]**

file: The name of the file to hold the log.
text: An optional message that will be added to the log.

/H(istory log) **/W**(rite to).

See also: HISTORY.

Usage

LOG keeps a record of all internal and external commands you use, whether they are executed from the prompt or from a batch file. Each entry includes the current system date and time, along with the actual command after any alias or variable expansion. You can use the log file as a record of your daily activities.

LOG with the **/H** option keeps a similar record called a "history log". The history log records only commands entered at the prompt; it does not record batch file commands. In addition, the history log does not record the date and time for each command, and it records commands before aliases and variables are expanded.

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

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

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

To enable or disable logging, add the word "ON" or "OFF" after the LOG command:

```
c:\> log on
```

or

```
c:\> log /h on
```

Entering LOG or LOG /H with *text* writes a message to the log file, even if logging is set OFF. This allows you to enter headers in the log file:

```
c:\> log "Started work on the database system"
```

The LOG file format looks like this:

```
[date time]  command
```

where the date and time are formatted according to the country code set for your system.

The LOG /H output can be used as the basis for writing batch files. Start LOG /H, then execute the commands that you want the batch file to execute. When you are finished, turn LOG /H off. The

resulting file can be turned into a batch file that performs the same commands with little or no editing.

Options

/H (History log) This option makes the other options on the command line (after the **/H**) apply to the history log. For example, to turn on history logging and write to the file C:\LOG\HLOG:

```
c:\>  log /h /w c:\log\hlog
```

/W (Write) This switch specifies a different filename for the LOG or LOG /H output. It also automatically performs a LOG ON command. For example, to turn logging on and write the log to *C:\LOG\LOGFILE*:

```
c:\> log /w c:\log\logfile
```

Once you select a new file name with the LOG /W or LOG /H/W command, LOG will use that file until you issue another LOG /W or LOG /H/W command, or until you reboot your computer. Turning LOG or LOG /H off or on does not change the file name.

MD / MKDIR

Purpose: Create a subdirectory.

Format: **MD [/N /S]** ***path*****...**

or

MKDIR [/N /S] ***path*****...**

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

/N(o update) **/S**(ubdirectories)

See also: RD.

Usage

MD and MKDIR are synonyms. You can use either one.

MD creates a subdirectory anywhere in the directory tree. To create a subdirectory from the root, start the *path* with a backslash [****]. For example, this command creates a subdirectory called *MYDIR* in the root directory:

```
c:\> md \mydir
```

If no path is given, the new subdirectory is created in the current directory. This example creates a subdirectory called *DIRTWO* in the current directory:

```
c:\mydir> md dirtwo
```

To create a directory from the parent of the current directory (that is, to create a sibling of the current directory), start the pathname with two periods and a backslash [**..**].

If MD creates one or more directories, they will be added automatically to the extended directory search database unless the **/N** option is specified.

Option

/N (No update) Do not update the extended directory search database, *JPSTREE.IDX*. This is useful when creating a temporary directory which you do not want to appear in the extended search database.

/S (Subdirectories) MD creates one directory at a time unless you use the **/S** option. If you need to create the directory *C:\ONE\TWO\THREE* and none of the named directories exist, you can use **/S** to have MD create all of the necessary subdirectories for you in a single command:

```
c:\> md /s \one\two\three
```

MEMORY

Purpose: Display the amount and status of system RAM.

Format: **MEMORY**

Usage

MEMORY lists the total and available physical RAM, the total and free environment and alias space, and the total history space, and the total history space and the amount of memory Take Command is using for environment variable space, alias space, and history space.

MOVE

Purpose: Move files to a new directory and drive.

Format: **MOVE [/A:[[-]rhsda] /C /D /E /H /M /N /P /Q /R /S /T /U /V]** ***source... destination***

source: A file or list of files to move.
destination: The new location for the files.

/A: (Attribute select)	**/P**(rompt)
/C(hanged)	**/Q**(uiet)
/D(irectory)	**/R**(eplace)
/E (No error messages)	**/S**(ubdirectory tree)
/H(idden and system)	**/T**(otal)
/M(odified)	**/U**(pdate)
/N(othing)	**/V**(erify)

See also: COPY and RENAME.

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. Date, time, size, or file exclusion 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.)

When you move a single file, the ***destination*** can be a directory name or a file name. If it is a directory name, and you add a backslash [\] to the end of the name, MOVE will display an error message if the name does not refer to an existing directory. You can use this feature to keep MOVE from treating a mistyped ***destination*** directory name as a file name, and attempting to move the ***source*** file to that name.

If you MOVE multiple files, the ***destination*** **must** be a directory name. MOVE will move each file into the ***destination*** directory with its original name. If the ***destination*** is not a directory, MOVE will display an error message and exit. For example, if *C:\FINANCE\MYFILES* is not a directory, this command will display an error; otherwise, the files will be moved to that directory:

```
c:\> move *.wks *.txt c:\finance\myfiles
```

The **/D** option can be used for single or multiple file moves; it checks to see whether the ***destination*** is a directory, and will prompt to see if you want to create the ***destination*** directory if it doesn't exist.

If MOVE creates one or more destination directories, they will be added automatically to the extended directory search database; see Extended Directory Searches for details.

Be careful when you use MOVE with the SELECT command. If you SELECT multiple files and the ***destination*** is not a directory (for example, because of a misspelling), MOVE will assume it is a file name. In this case each file will be moved in turn to the ***destination*** file, overwriting the previous file, and then the original will be erased before the next file is moved. At the end of the command, all of the original files will have been erased and only the last file will exist as the ***destination*** file.

You can avoid this problem by using square brackets with SELECT instead of parentheses (be sure that you don't allow the command line to get too long watch the character count in the upper left corner while you're selecting files). MOVE will then receive one list of files to move instead of a series of individual filenames, and it will detect the error and halt. You can also add a backslash [\] to the end of the *destination* name to ensure that it is the name of a subdirectory (see above).

Advanced Features and Options

MOVE first attempts to rename the file(s), which is the fastest way to move files between subdirectories on the same drive. If that fails, (*e.g.*, because the ***destination*** is on a different drive or already exists), MOVE will copy the file(s) and then delete the originals.

If MOVE must physically copy the files and delete the originals, rather than renaming them (see above), then some disk space may be freed on the *source* drive. The free space may be the result of moving the files to another drive, or of overwriting a larger *destination* file with a smaller *source* file. MOVE displays the amount of disk space recovered unless the **/Q** option is used (see below). It does so by comparing the amount of free disk space before and after the MOVE command is executed. However, this amount may be incorrect if you are using a deletion tracking system which retains deleted files for later recovry, or if another program performs a file operation while the MOVE command is executed.

When physically copying files, MOVE preserves the hidden, system, and read-only attributes of the *source* files, and sets the archive attribute of the *destination* files. However, if the files can be renamed, and no copying is required, then the *source* file attributes are not changed.

Use caution with the **/A:** and **/H** switches (both of which can allow MOVE to process hidden files) when you are physically moving files, and both the ***source*** and ***destination*** directories contain file descriptions. If the ***source*** file specification matches the description file name (normally *DESCRIPT.ION*), and you tell MOVE to process hidden files, the *DESCRIPT.ION* file itself will be moved, overwriting any existing file descriptions in the ***destination*** directory. For example, if the *C:\DATA* directory contains file descriptions, this command would overwrite any existing descriptions in the *D:\SAVE* directory:

```
c:\data> move /h d*.* d:\save\
```

(If you remove the hidden attribute from the *DESCRIPT.ION* file the same caution applies even if you do not use **/A:** or **/H**, as *DESCRIPT.ION* is then treated like any other file.)

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

R	Read-only	**D**	Subdirectory
H	Hidden	**A**	Archive

S System

If no attributes are listed at all (*e.g.*, **/A:**), MOVE will select all files including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set. See the cautionary note under **Advanced Features and Options** above before using **/A:** when both ***source*** and ***destination*** directories contain file descriptions.

/C (Changed files) Move files only if the *destination* file exists and is older than the *source* (see also **/U**). This option is useful for updating the files in one directory from those in another without moving any newly-created files.

/D (Directory) Requires that the *destination* be a directory. If the *destination* does not exist, MOVE will prompt to see if you want to create it. If the *destination* exists as a file, MOVE will fail with an "Access denied" error. Use this option to avoid having MOVE accidentally interpret your *destination* name as a file name when it's really a mistyped directory name.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files and aliases.

/H (Hidden) Move all files, including hidden and system files. See the cautionary note under **Advanced Features and Options** above before using **/H** when both ***source*** and ***destination*** directories contain file descriptions.

/M (Modified) 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. **/N** does **not** prevent creation of ***destination*** subdirectories when it is used with **/S**.

/P (Prompt) Prompt the user to confirm each move. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames, the total number of files moved, or the amount of disk space recovered, if any. This option is most often used in batch files. See also **/T**.

/R (Replace) Prompt for a **Y** or **N** response before overwriting an existing *destination* file.

/S (Subdirectories) Move an entire subdirectory tree to another location. MOVE will attempt to create the *destination* directories if they don't exist, and will remove empty subdirectories after the move. When **/D** is used with **/S**, you will be prompted if the first *destination* directory does not exist, but subdirectories below that will be created automatically by MOVE. If MOVE /S creates one or more destination directories, they will be added automatically to the extended directory search database.

If you attempt to use **/S** to move a subdirectory tree into part of itself, MOVE will detect the resulting infinite loop, display an error message, and exit.

/T (Total) Don't display filenames as they are moved, but display the total number of files 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 when a file is moved from one drive to another. This is the same as executing the VERIFY ON command, but is only active during the MOVE. **/V** does **not** read back the file and compare its contents with what was written; it only verifies that the data written to disk is physically readable.

MSGBOX

Purpose: Display a message box and return the user's response.

Format: **MSGBOX OK | OKCANCEL | YESNO | YESNOCANCEL ["title"] *prompt***

title: Text for the title bar of the message box.
prompt: Text that will appear inside the message box.

See also: INKEY, INPUT, and QUERYBOX.

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. Take Command 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 Take Command variable **%_?**. Be sure to save the return value in another variable or test it immediately; because the value of **%_?** changes with every internal command.

The following list shows the value returned for each possible selection:

Yes	10	**No**	11
OK	10	**Cancel**	12

If you exit the message box without selecting one of these options (for example, some message boxes allow you to exit by pressing **Esc** or double-clicking the close button), MSGBOX will set **%_?** to 0. If there is an error in the MSGBOX command itself, **%_?** will be set to 1 for a syntax error or 2 for any other error.

For example, to display a Yes or No message box and take action depending on the result, you could use commands like this:

```
msgbox yesno "Copy" Copy all files to A:?
iff %_? == 10 then
   copy *.* a:
endiff
```

MSGBOX creates a popup dialog box. If you prefer to retrieve input from inside the command line window, see the INKEY and INPUT commands.

ON

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

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

or

ON ERROR [*command*]

or

ON ERRORMSG [*command*]

Usage

ON can only by used in batch files.

ON sets a "watchdog" that remains in effect for the duration of the current batch file. Whenever a BREAK or ERROR condition occurs after ON has been executed, the corresponding *command* is automatically executed.

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

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

ON ERROR executes the ***command*** immediately after the error occurs, without displaying any command processor error message (operating system errors may still be displayed in some cases). ON ERRORMSG displays the appropriate error message, then executes the ***command***. If both are specified, ON ERROR will take precedence, and the ON ERRORMSG setting will be ignored. The remainder of this section discusses both settings together, using the term "ON ERROR[MSG]".

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

Each time ON BREAK or ON ERROR[MSG] is used, it defines a new *command* to be executed for a break or error, and any old *command* is discarded. If you use ON BREAK or ON ERROR[MSG] with no following *command*, that type of error handling is disabled. Error handling is also automatically disabled when the batch file exits.

ON BREAK and ON ERROR[MSG] only affect the current batch file. If you CALL another batch file, the first batch file's error handling is suspended, and the CALLed file must define its own error handling. When control returns to the first batch file, its error handling is reactivated.

The *command* can be any command that can be used on a batch file line by itself. Frequently, it is a GOTO or GOSUB command. For example, the following fragment traps any user attempt to end the batch file by pressing **Ctrl-C** or **Ctrl-Break**. It scolds the user for trying to end the batch file and then continues displaying the numbers from 1 to 1000:

```
on break gosub gotabreak
do i = 1 to 1000
echo %i
enddo
quit
```

```
:gotabreak
echo Hey!  Stop that!!
return
```

You can use a command group as the *command* if you want to execute multiple commands, for example:

```
on break (echo Oops, got a break! & quit)
```

ON BREAK and ON ERROR[MSG] always assume that you want to continue executing the batch file. After the *command* is executed, control automatically returns to the next command in the batch file (the command after the one that was interrupted by the break or error). The only way to avoid continuing the batch file after a break or error is for the *command* to transfer control to another point with GOTO, end the batch file with QUIT or CANCEL, or start another batch file (without CALLing it).

When handling an error condition with ON ERROR, you may find it useful to use internal variables, particularly %_? and %_SYSERR, to help determine the cause of the error.

The ON ERROR[MSG] command will **not** be invoked if an error occurs while reading or writing a redirected input file, output file, or pipe.

Caution: If a break or error occurs while the *command* specified in ON BREAK or ON ERROR[MSG] is executing, the *command* will be restarted. This means you must use caution to avoid or handle any possible errors in the commands invoked by ON ERROR[MSG], since such errors can cause an infinite loop.

OPTION

Purpose: Modify Take Command configuration.

Format: **OPTION [//optname=value ...]**

optname: An INI file directive to set or modify.
value: A new value for that directive.

See also: The *TCMD.INI* file.

Usage:

OPTION displays a set of dialogs which allows you to modify many of the configuration options stored in the *TCMD.INI* file.

When you exit from the dialogs, you can select **Save** to save your changes in the *.INI* file for use in the current session and all future sessions, select **OK** to use your changes in the current session only, or select **Cancel** to discard the changes. See Configuration Dialogs for more information.

OPTION does not preserve comments when saving modified settings in the *.INI* file. To be sure *.INI* file comments are preserved, put them on separate lines in the file (see *TCMD.INI* for details).

Setting Individual Options

If you follow the OPTION command with one or more sequences of a double slash mark [*//*] followed by an **option=value** setting, the OPTION dialogs will not appear. Instead, the new settings will take effect immediately, and will be in effect for the current session only. This example turns off batch file echo and changes the input colors to bright cyan on black:

```
c:\> option //BatchEcho = No //InputColors = bri cya on bla
```

Option values may contain whitespace. However, you cannot enter an option value which contains the "//" string.

This feature is most useful for testing settings quickly, and in aliases or batch files which depend on certain options being in effect.

Changes made with *//* are temporary. They will not be saved in the *.INI* file, even if you subsequently load the option dialogs and select **Save**.

PATH

Purpose: Display or alter the list of directories that Take Command 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 Take Command is asked to execute an external command (a *.COM*, *.EXE*, *.BTM*, or *.BAT* file, or an executable extension), it first looks for the file in the current directory. If it fails to find an executable file there, it then searches the \WINDOWS directory, the \WINDOWS\SYSTEM directory, and the directory which contains *TCMD.EXE*. If it still hasn't found an executable file, Take Command will then search each of the *directories* specified in the PATH setting.

For example, after the following PATH command, Take Command will search for an executable file in seven directories: the current directory, the two Windows directories, the *TCMD.EXE* directory, the root directory on drive C, then the *DOS* subdirectory on C, and then the *UTIL* subdirectory on C:

```
c:\> path c:\;c:\dos;c:\util
```

The list of *directories* to search is stored as an environment string, and can also be set or viewed with SET, and edited with ESET.

Directory names in the path must be separated by semicolons [**;**]. Each directory name is shifted to upper case to maintain compatibility with programs which can only recognize upper case directory names in the path. If you modify your path with the SET or ESET command, you may include directory names in lower case. These may cause trouble with some programs, which assume that all path entries have been shifted to upper case.

If you enter PATH with no parameters, the current path is displayed:

```
c:\> path
PATH=C:\;C:\DOS;C:\UTIL
```

Entering PATH and a semicolon clears the search path so that only the current directory is searched for executable files (this is the default at system startup).

Some applications also use the PATH to search for their data files.

If you include an explicit file extension on a command name (for example, *WP.EXE*), the search will find files with that name and extension in the current directory and every directory in the path. It will not locate other executable files with the same base name (*e.g.*, *WP.COM*).

If you have an entry in the path which consists of a single period [**.**], the current directory will **not** be searched first, but instead will be searched when Take Command 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.

In normal use, Take Command can create a path as long as 250 characters (the command-line limit is

255 characters, and "PATH " takes five). However, some DOS applications expect a path no longer than the traditional limit of 123 characters. If you extend your path beyond this limit and experience problems with DOS application programs started from Take Command, you may need to reduce the length of the PATH.

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 506 characters (the internal buffer limit, with room for "PATH "). It is usually more efficient to use aliases to load application programs than to create a long PATH. See ALIAS for details.

If you specify an invalid directory in the path, it will be skipped and the search will continue with the next directory in the path.

PAUSE

Purpose: Suspend batch file or alias execution.

Format: **PAUSE [*text*]**

text: The message to be displayed as a user prompt.

Usage

A PAUSE command will suspend execution of a batch file or alias, giving you the opportunity to change disks, turn on the printer, etc.

PAUSE waits for any key to be pressed and then continues execution. You can specify the *text* that PAUSE displays while it waits for a keystroke, or let it use the default message:

```
Press any key when ready...
```

For example, the following batch file fragment prompts the user before erasing files:

```
pause Press Ctrl-C to abort, any other key to erase all .LST files
erase *.lst
```

If you press **Ctrl-C** or **Ctrl-Break** while PAUSE is waiting for a key, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job. A batch file can handle **Ctrl-C** and **Ctrl-Break** itself with the ON BREAK command.

POPD

Purpose: Return to the disk drive and directory at the top of the directory stack..

Format: **POPD [*]**

See also: DIRS, PUSHD, and Directory Navigation.

Usage

Each time you use the PUSHD command, it saves the current disk drive and directory on the internal directory stack. POPD restores the last drive and directory that was saved with PUSHD and removes that entry from the stack. You can use these commands together to change directories, perform some work, and return to the starting drive and directory.

Directory changes made with POPD are recorded in the directory history list and can be displayed in the directory history window. Read the section on Directory Navigation for complete details on this and other directory navigation features.

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> pushd c:\wordp\memos
c:\wordp\memos> pushd a:\123
a:\123> popd
c:\wordp\memos> popd
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.

Under Windows 3.x, any change you make to the current directory is "global," and may affect other applications.

PROMPT

Purpose: Change the command-line prompt.

Format: **PROMPT [*text*]**

text: Text to be used as the new command-line prompt.

Usage

You can change and customize the command-line prompt at any time. The prompt can include normal text and system information such as the current drive and directory, the time and date, and the amount of memory available. You can create an informal "Hello, Bob!" prompt or an official-looking prompt full of impressive information.

The prompt *text* can contain special commands in the form **$?**, where **?** is one of the characters listed below:

- **b** The vertical bar character [**|**].
- **c** The open parenthesis [**(**].
- **d** Current date, in the format: *Fri 12-12-97* (the month, day, and year are formatted according to your current country settings).
- **D** Current date, in the format: *Fri Dec 12, 1997*.
- **e** The ASCII ESC character (decimal 27).
- **f** The close parenthesis [**)**].
- **g** The **>** character.
- **h** Backspace over the previous character.
- **l** The **<** character.
- **m** The time (hours and minutes) in 24-hour format.
- **M** The time (hours and minutes) in the default country format.
- **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 *3.10*.
- **xd:** Current directory on drive *d:*, in lower case.
- **Xd:** Current directory on drive *d:*, in upper case, including the drive letter.
- **z** Current shell nesting level. The first copy of Take Command is shell level 0, and each subsequent copy increments the level by 1.

+ Display one **+** character for each directory on the PUSHD directory stack.

$** The **$ character.

_ CR/LF (go to beginning of a new line).

For example, to set the prompt to the current date and time, with a ">" at the end:

```
c:\> prompt $d $t $g
Fri  Dec 12, 1997 10:29:19 >
```

To set the prompt to the current date and time, followed by the current drive and directory in upper case on the next line, with a ">" at the end:

```
c:\> prompt $d $t$_$P$g
Fri  Dec 12, 1997 10:29:19
[c:\]
```

The Take Command prompt can be set in TCSTART, or in any batch file that runs when Take Command starts. The Take Command default prompt is **ng** (drive name followed by ">") on floppy drives, and **pg** (current drive and directory followed by ">") 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
(3104K) c:\data>
```

Notice that the @DOSMEM function is shown with two leading percent signs [**%**]. If you used only one percent sign, the @DOSMEM function would be expanded once when the PROMPT command was executed, instead of every time the prompt is displayed. As a result, the amount of memory would never change from the value it had when you entered the PROMPT command. You can also use back quotes to delay expanding the variable function until the prompt is displayed:

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

You can use this feature along with the @EXEC variable function to create a complex prompt which not only displays information but executes commands. For example, to execute an alias which checks battery status each time the prompt is displayed (enter the alias on one line):

```
c:\> alias cbatt `if %_apmlife lt 30 beep 440 4 880 4 440 4 880 4`
c:\> prompt `%@exec[@cbatt]$p$g`
```

You can include ANSI escape sequences in the PROMPT ***text*** using Take Command's built-in ANSI support. This example uses ANSI sequences to set a prompt that displays the shell level, date, time and path in color on the top line of the screen (enter the command as one line):

```
c:\> prompt $e[s$e[1;1f$e[41;1;37m$e[K[$z] $d
     Time: $t$h$h$h  Path: $p$e[u$e[0;32m$n$g
```

PUSHD

Purpose: Save the current disk drive and directory, optionally changing to a new drive and directory.

Format: **PUSHD [*path*]**

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

See also: DIRS, POPD and Directory Navigation.

Usage

PUSHD saves the current drive and directory on a "last in, first out" directory stack. The POPD command returns to the last drive and directory that was saved by PUSHD. You can use these commands together to change directories, perform some work, and return to the starting drive and directory. The DIRS command displays the contents of the directory stack.

To save the current drive and directory, without changing directories, use the PUSHD command by itself, with no ***path***.

If a ***path*** is specified as part of the PUSHD command, the current drive and directory are saved and PUSHD changes to the specified drive and directory. If the ***path*** includes a drive letter, PUSHD changes to the specified directory on the new drive without changing the current directory on the original drive.

This example saves the current directory and changes to *C:\WORDP\MEMOS*, then returns to the original directory:

```
c:\> pushd \wordp\memos
c:\wordp\memos> popd
c:\>
```

PUSHD can also change to a network drive and directory specified with a UNC name (see File Systems for details).

If PUSHD cannot change to the directory you have specified it will attempt to search the CDPATH and the extended directory search database. You can also use wildcards in the ***path*** to force an extended directory search. Read the section on Directory Navigation for complete details on these and other directory navigation features.

Directory changes made with PUSHD are also recorded in the directory history list and can be displayed in the directory history window.

The directory stack can hold up to 511 characters, or between 20 and 40 typical entries (depending on the length of the names). If you exceed this limit, the oldest entry is removed before adding a new entry.

Under Windows 3.x, any change you make to the current directory is "global," and may affect other applications.

QUERYBOX

Purpose: Use a dialog box to get an input string from the user and save it in an environment variable.

Format: **QUERYBOX /E /Ln ["title"] *prompt %%varname***

title: Text for the title bar of the message box.
prompt: Text that will appear inside the message box.
varname : Variable name where the input will be saved.

/E(dit existing value) **/L** (maximum length)

See also: INKEY, INPUT, and MSGBOX.

Usage

QUERYBOX displays a dialog box with a prompt, an optional title, and a string input field. Then it waits for your entry, and places any characters you type into an environment variable. QUERYBOX is normally used in batch files and aliases to get string input.

QUERYBOX is similar to INPUT, except it appears as a popup dialog box. If you prefer to work within the command line window, see the INKEY and INPUT commands.

Standard command-line editing keys may be used to edit the input string as it is entered. All characters entered up to, but not including, the carriage return are stored in the variable.

For example, to prompt for a string and store it in the variable NAME:

```
querybox "File Name" Enter a name:  %%name
```

If you press **Ctrl-C** or **Ctrl-Break** while QUERYBOX is waiting for input, execution of an alias will be terminated, and execution of a batch file will be suspended while you are asked whether to cancel the batch job. A batch file can handle **Ctrl-C** and **Ctrl-Break** itself with ON BREAK.

Options:

/E (Edit) Allows you to edit an existing value. If there is no existing value for ***varname***, QUERYBOX allows you to enter a new value.

/Ln (Length) Sets the maximum number of characters which QUERYBOX will accept to "n".

QUIT

Purpose: Terminate the current batch file.

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

value: The numeric exit code to return to Take Command or to the previous batch file.

See also: CANCEL and EXIT.

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.

This example batch file fragment checks to see if the user entered "quit" and exits if true.

```
input  Enter your choice :  %%option
if  "%option" == "quit"  quit
```

QUIT only ends the current batch file. To end all batch file processing, use the CANCEL command.

If you specify a *value*, QUIT will set the ERRORLEVEL or exit code to that value. For information on exit codes see the IF command, and the %? variable.

You can also use QUIT to terminate an alias. If you QUIT an alias while inside a batch file, QUIT will end both the alias and the batch file and return you to the command prompt or to the calling batch file.

RD / RMDIR

Purpose: Remove one or more subdirectories.

Format: **RD *path* ...**

or

RMDIR *path* ...

path: The name of one or more subdirectories to remove.

See also: MD.

File Selection

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

Usage

RD and RMDIR are synonyms. You can use either one.

RD removes directories from the directory tree. For example, to remove the subdirectory *MEMOS* from the subdirectory *WP*, you can use this command:

```
c:\> rd \wp\memos
```

Before using RD, you must delete all files and subdirectories (and their files) in the ***path*** you want to remove. Remember to remove hidden and read-only files as well as normal files (you can use DEL /Z to delete hidden and read-only files).

You can use wildcards in the ***path***.

If RD deletes one or more directories, they will be deleted automatically from the extended directory search database.

You cannot remove the root directory, the current directory (**.**), any directory above the current directory in the directory tree, or any directory in use by another process in a multitasking system.

REBOOT

Purpose: Do a warm or cold system reboot.

Format: **REBOOT [/R /V]**

/R(estart) **/V**(erify)

Usage

REBOOT will log off or restart Windows or 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
```

Take Command issues the standard commands to shut down other applications and the operating system before rebooting.

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.

Options

/R (Restart) Restart Windows, but do not reboot the computer.

/V (Verify) Prompt for confirmation (**Y** or **N**) before rebooting.

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. For example:

```
rem This batch file provides a
rem menu-based system for accessing
rem word processing utilities.
rem
rem Clear the screen and get selection
cls
```

REM must be followed by a space or tab character and then your comment. Comment lines can be up to 255 characters long. Take Command will normally ignore everything on the line after the REM command, including quote characters, redirection symbols, and other commands (see below for the exception to this rule).

If ECHO is ON, the comment is displayed. Otherwise, it is ignored. If ECHO is ON and you don't want to display the line, preface the REM command with an at sign [**@**].

You can also place a comment in a batch file by starting the comment line with two colons [**::**]. In essence this creates a batch file "label" without a valid label name. Such comments are processed slightly faster than those entered with REM, because they do not require the command processor to handle a command.

When debugging a batch file, you may find it convenient to use REM to temporarily disable certain commands. Simply add "REM " at the start of any command to convert it temporarily to a comment.

You can use REM to create a zero-byte file if you use a redirection symbol **immediately** after the REM command. For example, to create the zero-byte file *C:\FOO*:

```
c:\> rem>foo
```

(This capability is included for compatibility with traditional command processors. A simpler method for creating a zero-byte file with Take Command is to use **>filename** as a command, with no actual command before the [**>**] redirection character.)

REN / RENAME

Purpose: Rename files or subdirectories.

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

or

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

old_name: Original name of the file(s) or subdirectory.
new_name: New name to use, or new path on the same drive.

/A: (Attribute select)
/E (No error messages)
/N(othing)
/P(rompt)
/Q(uiet)
/S(ubdirectory)
/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 give REN the ***old_name*** of an existing file or subdirectory and then a ***new_name***. The ***new_name*** must not already exist you can't give two files the same name (unless they are in different directories). The first example renames the file *MEMO.TXT* to *MEM.TXT*. The second example changes the name of the *\WORD* directory to *\WP*:

```
c:\> rename memo.txt mem.txt
c:\> rename \word \wp
```

If you use REN to rename a directory, the extended directory search database will be automatically updated to reflect the change.

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 tcstart.btm *.sav
c:\> ren *.txt *.sav
```

REN can move files to a different subdirectory on the same drive. When it is used for this purpose, REN requires one or more filenames for the ***old_name*** and a directory name for the ***new_name***:

```
c:\> ren memo.txt \wp\memos\
c:\> ren oct.dat nov.dat \data\save\
```

The final backslash in the last two examples is optional. If you use it, you force REN to recognize the

last argument as the name of a directory, not a file. The advantage of this approach is that if you accidentally mistype the directory name, REN will report an error instead of renaming your files in a way that you didn't intend.

Finally, REN can move files to a new directory and change their name at the same time if you specify both a path and file name for *new_name*. In this example, the files are renamed with an extension of *.SAV* as they are moved to a new directory:

```
c:\> ren *.dat \data\save\*.sav
```

You cannot rename a subdirectory to a new location on the directory tree.

REN does not change a file's attributes. The ***new_name*** file(s) will have the same attributes as ***old_name***.

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

If no attributes are listed at all (*e.g.*, **/A:**), REN will rename all files including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/E (No error messages) Suppress all non-fatal error messages, such as "File Not Found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files.

/N (Nothing) Do everything except actually rename the file(s). This option is useful for testing what a REN command will actually do.

/P (Prompt) Prompt the user to confirm each rename operation. Your options at the prompt are explained in detail under Page and File Prompts.

/Q (Quiet) Don't display filenames or the number of files renamed. This option is most often used in batch files. See also **/T**.

/S (Subdirectory) Normally, you can rename a subdirectory only if you do not use any wildcards in the *new_name*. This prevents subdirectories from being renamed inadvertently when a group of files is being renamed with wildcards. **/S** will let you rename a subdirectory even when you use wildcards. **/S** does **not** cause REN to process files in the current directory and all subdirectories as it does in some other file processing commands. To rename files throughout a directory tree, use a GLOBAL REN.

/T (Total) Don't display filenames as they are renamed, but report the number of files renamed. See also **/Q**.

RETURN

Purpose: Return from a GOSUB (subroutine) in a batch file.

Format: **RETURN [*value*]**

value: The numeric exit code to return to the command processor or to the previous batch file.

See also: GOSUB.

Usage

Take Command allows subroutines in batch files.

A subroutine begins with a label (a colon followed by one or more words) and ends with a RETURN command.

The subroutine is invoked with a GOSUB command from another part of the batch file. When a RETURN command is encountered the subroutine terminates, and execution of the batch file continues on the line following the original GOSUB. If RETURN is encountered without a GOSUB, Take Command will display a "Missing GOSUB" error.

The following batch file fragment calls a subroutine which displays the files in the current directory:

```
echo Calling a subroutine
gosub subr1
echo Returned from the subroutine
quit
:subr1
dir /a/w
return
```

If you specify a ***value***, RETURN will set the ERRORLEVEL or exit code to that value. For information on exit codes see the IF command, and the **%?** variable.

SCREEN

Purpose: Position the cursor on the screen and optionally display a message.

Format: **SCREEN *row column* [*text*]**

row: The new row location for the cursor.
column: The new column location for the cursor.
text: Optional text to display at the new cursor location.

See also: ECHO, SCRPUT, TEXT, and VSCRPUT.

Usage

SCREEN allows you to create attractive screen displays in batch files. You use it to specify where a message will appear on the screen. You can use SCREEN to create menus and other similar displays. The following batch file fragment displays a menu:

```
@echo off
cls
screen 3 10 Select a number from 1 to 4:
screen 6 20  1 - Word Processing
screen 7 20  2 - Spreadsheet
screen 8 20  3 - Telecommunications
screen 9 20  4 - Quit
```

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 25 line by 80 column display, valid *rows* are 0 - 24 and valid *columns* are 0 - 79. The maximum ***row*** value is determined by the current height of the Take Command window. The maximum ***column*** value is determined by the virtual screen width (see Resizing the Take Command Window for more information). SCREEN checks for a valid ***row*** and ***column***, and displays a "Usage" error message if either value is out of range.

You can also specify the *row* and *column* as offsets from the current cursor position. Begin the value with a plus sign [**+**] to move the cursor down the specified number of rows or to the right the specified number of columns, or with a minus sign [**-**] to move the cursor up or to the left. This example prints a string 3 lines above the current position, in absolute column 10:

```
screen -3 10 Hello, World!
```

If you specify 999 for the ***row***, SCREEN will center the text vertically on the display. If you specify 999 for the ***column***, SCREEN will center the text horizontally. This example prints a message at the center of the Take Command window:

```
screen 999 999 Hello, World
```

SCRPUT

Purpose: Position text on the screen and display it in color.

Format: **SCRPUT *row col* [BRIght] *fg* ON BRIght] *bg text***

row: Starting row
col: Starting column
fg: Foreground character color
bg: Background character color
text: The text to display

See also: ECHO, SCREEN, TEXT, and VSCRPUT.

Usage

SCRPUT allows you to create attractive screen displays in batch files. You use it to specify where a message will appear on the screen and what colors will be used to display the message text. You can use SCRPUT to create menu displays, logos, etc.

SCRPUT works like SCREEN, but requires you to specify the display colors. See Colors and Color Names for details.

The ***row*** and ***column*** are zero-based, so on a 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79. The maximum ***row*** value is determined by the current height of the Take Command window. The maximum ***column*** value is determined by the current virtual screen width (see Resizing the Take Command Window for more information). SCRPUT checks for a valid ***row*** and ***column***, and displays a "Usage" error message if either value is out of range.

You can also specify the *row* and *column* as offsets from the current cursor position. Begin the value with a plus sign [**+**] to move down the specified number of rows or to the right the specified number of columns, or with a minus sign [**-**] to move up or to the left.

If you specify 999 for the ***row***, SCRPUT will center the text vertically in the Take Command window. If you specify 999 for the ***column***, SCRPUT will center the text horizontally.

SCRPUT does not move the cursor when it displays the ***text***.

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

```
cls white on blue
scrput 3 10 bri whi on blu Select an option:
scrput 6 20 bri red on blu 1 - Word Processing
scrput 7 20 bri yel on blu 2 - Spreadsheet
scrput 8 20 bri gre on blu 3 - Communications
scrput 9 20 bri mag on blu 4 - Quit
```

SELECT

Purpose: Interactively select files for a command.

Format: **SELECT [/A[[:][-]rhsda] /C[HP] /E /H /I"text" /J /L /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)	**/I** (match descriptions)
/C[HP] (Compression)	**/J**(ustify names)
/E (use upper case)	**/L**(ower case)
/H(ide dots)	**/O**(rder)

File Selection

Supports extended wildcards, ranges, multiple file names, and include lists. 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 part of its command-line arguments.

When you execute the SELECT command, the file list is displayed in a full-window format which includes a top-line status bar and shows the command to be executed, the number of files marked, and the number of Kbytes in those files.

SELECT uses the cursor up, cursor down, PgUp, and PgDn keys to scroll through the file list. You can also use character matching to find specific files, just as you can in any popup window. While the file list is displayed you can enter any of the following keys to select or unselect files, display files, execute the command, or exit:

+ or **space**	Select a file, or unselect a marked file.
-	Unselect a marked file.
*****	Reverse all of the current marks (except those on subdirectories). If no files have been marked you can use * to mark all of the files.
/	Unselect all files.
Enter	Execute the command with the marked files, or with the currently highlighted file if no files have been marked.
Esc	Skip the files in the current display and go on to the next file specification inside the parentheses or brackets (if any).

Ctrl-C or **Ctrl-Break**	Cancel the current SELECT command entirely.

The file list is shown in standard directory format, with names at the left and descriptions at the right.

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 description area of the screen horizontally and view the additional text.

You can set the default colors used by SELECT on the Commands page of the configuration dialogs, or with the SelectColors directive in the *TCMD.INI* file. If SelectColors is not used, the SELECT display will use the current default colors.

Creating SELECT Commands

In the simplest form of SELECT, you merely specify the command and then the list of files from which you will make your selection(s). For example:

```
c:\> select copy (*.com *.exe) a:\
```

will let you select from among the *.COM* files on the current drive, and will then invoke the COPY command to copy each file you select to drive A:. After the *.COM* files are done, the operations will be repeated for the *.EXE* files.

If you want to select from a list of all the *.COM* and *.EXE* files mixed together, create an include list inside the parentheses by inserting a semicolon:

```
c:\> select copy (*.com;*.exe) a:\
```

Finally, if you want the SELECT command to send a single list of files to COPY, instead of invoking COPY once for each file you select, put the file names in square brackets instead of parentheses:

```
c:\> select copy [*.com;*.exe] a:\
```

If you use brackets, you have to be sure that the resulting command (the word COPY, the list of files, and the destination drive in this example) does not exceed the command line length limit of 255 characters for internal commands or 126 characters for external commands. The current line length is displayed by SELECT while you are marking files to help you to conform to these limits.

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, size and file exclusion ranges. The range(s) must be placed immediately after the word SELECT. If the ***command*** is an internal command that supports ranges, an independent range can also be used in the ***command*** itself.

You cannot use command grouping to make SELECT execute several commands, because SELECT will assume that the parentheses are marking the list of files from which to select, and will display an error message or give incorrect results if you try to use parentheses for command grouping instead. (You **can** use a SELECT command **inside** command grouping parentheses, you just can't use command grouping to specify a group of commands for SELECT to execute.)

Advanced Topics

If you don't specify a command, the selected filename(s) will become the command. For example, this command defines an alias called UTILS that selects from the executable files in the directory *C:\UTIL*, and then executes them in the order marked:

```
c:\> alias utils select (c:\util\*.com;*.exe;*.btm;*.bat)
```

If you want to use filename completion to enter the filenames inside the parentheses, type a space after the opening parenthesis. Otherwise the command-line editor will treat the open parenthesis as the first character of the filename.

With the **/I** option, you can select files based on their descriptions. SELECT will display files if their description matches the text after the **/I** switch. The search is not case sensitive. You can use wildcards and extended wild cards as part of the text.

When sorting file names and extensions for the SELECT display, Take Command normally assumes that sequences of digits should be sorted numerically (for example, the file DRAW2 would come before DRAW03 because 2 is numerically smaller than 03), rather than strictly alphabetically (where DRAW2 would come second because "2" comes after "0"). You can defeat this behavior and force a strict alphabetic sort with the **/O:a** option.

Options

/A[:] (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is optional. The attributes are:

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

If no attributes are listed at all (*e.g.*, SELECT **/A** ...), SELECT will display all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be included in the listing. For example, **/A:RHS** will display only those files with all three attributes set.

/C (Compression) Display per-file and total compression ratio on compressed drives. The compression ratio is displayed instead of the file description. The ratio is left blank for directories and files with a length of 0 bytes, and for files on non-compressed drives.

Only the compression programs distributed with MS-DOS and Windows 95 (DRVSPACE and DBLSPACE) are supported.

Using **/CH** displays compression ratios like **/C**, but bases the calculation on the host drive's cluster size. This gives a more accurate picture of the space saved through compression than is given by **/C**.

If **/CP** is used instead of **/C**, the compression is displayed as a percentage (*e.g.*, 33%) instead of a ratio. If **/CHP** is used instead of **/CH**, the host compression is displayed as a percentage. The **/CHP** option must be entered as shown; you can **not** use **/CPH**.

See **DIR /C** for more details on how compression ratios are calculated.

/E Display filenames in upper case; also see SETDOS /U and the UpperCase directive in the *TCMD.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. You can select all filenames that have a description with **/I"[?]*"**, or all filenames that do not have a description with **/I"[]"**. **/I** will be ignored if **/C** or **/O:c** is also used.

/J (Justify names) Justify (align) filename extensions and display them in the traditional format.

/L (Lower case) Display file and directory names in lower case; also see SETDOS /U and the UpperCase directive in the *TCMD.INI* file.

/O (Order) Set the sort order for the files. The order can be any combination of the following options:

- **-** Reverse the sort order for the next option.
- **a** Sort in ASCII order, not numerically, when there are digits in the name.
- **c** Sort by compression ratio (the least compressed file in the list will be displayed first). For information on supported compression systems see **/C** above. If **/O:c** is used with **/CH** or **/CHP** the sort will be based on the host drive compression ratios.
- **c** Sort by compression ratio (the least compressed file in the list will be displayed first). For information on supported compression systems see **/C** above.
- **d** Sort by date and time (oldest first).
- **e** Sort by extension.
- **g** Group subdirectories first, then files.
- **i** Sort by file description.
- **n** Sort by filename (this is the default).
- **r** Reverse the sort order for all options.
- **s** Sort by size.
- **u** Unsorted.

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. Take Command itself uses several environment variables.

You can also create or modify environment variables with the Environment dialog. The dialog allows you to enter the variable ***name*** and ***value*** into separate fields in a dialog box, rather than using the SET command. All of the information in this section also applies to variables defined via the dialog, unless otherwise noted.

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 PATH, an entry called CMDLINE, and whatever other environment variables you and your programs have established:

```
c:\> set
PATH=C:\;C:\UTIL
CMDLINE=C:\TCMD201\TCSTART.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 Take Command. 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 Take Command 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 255 characters.

The size of the environment is set automatically, and increased as necessary as you add variables.

Take Command normally passes its environment to DOS applications, but **not** to Windows applications. See Starting Windows Applications for additional details.

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 255-byte command-line length limit for Take Command: The file is in the same format as the SET display (*i.e.*, ***name=value***), so SET /R can accept as input a file generated by redirecting SET output. For example, the following commands will save the environment variables to a file, and then reload them from that file:

```
c:\> set > varlist
c:\> 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 Take Command configuration.

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

/A(NSI)
/C(ompound)
/D(escriptions)
/E(scape character)
/F(ormat for @EVAL)
/G (numeric separators)
/I(nternal commands)
/M(ode for editing)
/N(o clobber)
/P(arameter character)
/S(hape of cursor)
/U(pper case)
/V(erbose)
/X (expansion, special characters)
/Y (debug batch file)

Usage

SETDOS allows you to customize certain aspects of Take Command 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 Take Command executes the configuration directives in the *TCMD.INI* file, and can also be changed from the configuration dialogs.. The name of the corresponding directive is listed with each option below; if none is listed, that option cannot be set from the *TCMD.INI* file. You can also define the SETDOS options in your *TCSTART* or other startup file (see Automatic Batch Files), in aliases, or at the command line.

Options

/A (ANSI) This option determines whether Take Command's ANSI support is enabled. **/A1** enables ANSI string processing in the Take Command window. The default of **/A0** disables ANSI strings. See the ANSI Codes Reference for a list of the ANSI sequences supported by Take Command. Also see the _ANSI internal variable.

/C (Compound character) This option sets the character used for separating multiple commands on the same line. The default is the caret [^]. You cannot use any of the redirection characters (| > <), or the blank, tab, comma, or equal sign as the command separator. The command separator is saved by SETLOCAL and restored by ENDLOCAL. The following example changes the separator character to a tilde [**~**]:

```
c:\> setdos /c~
```

If you want to share batch files or aliases among Take Command and our products for OS/2 and Windows NT, see the %+ variable, which retrieves the current command separator, and the section on Special Character Compatibility for details on using compatible command separators for all the products you use. Also see the CommandSep directive.

/D (Descriptions) This option controls whether file processing commands like COPY, DEL, MOVE, and REN process file descriptions along with the files they belong to. **/D1** turns description processing on, which is the default. **/D0** turns description processing off. Also see the Descriptions directive.

You can also use **/D** to set the name of the hidden file in each directory that contains file

descriptions. To do so, follow **/D** with the filename in quotes:

```
c:\> setdos /d"files.bbs"
```

Use this option with caution because changing the name of the description file will make it difficult to transfer file descriptions to another system. This option is provided for bulletin board system operators and others who have special needs.

/E (Escape character) This option sets the character used to suppress the normal meaning of the following character. Any character following the escape character will be passed unmodified to the command. The default escape character for Take Command is a Ctrl-X; for Take Command 32 it is a caret [**^**]. You cannot use any of the redirection characters (| > <) or the blank, tab, comma, or equal sign as the escape character. The escape character is saved by SETLOCAL and restored by ENDLOCAL. Certain characters (**b**, **c**, **e**, **f**, **k**, **n**, **q**, **r**, **s**, and **t**) have special meanings when immediately preceded by the escape character.

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

/F (Format for @EVAL) This option lets you set default decimal precision for the @EVAL variable function. The maximum precision is 16 digits to the left of the decimal point and up to 8 digits to the right of the decimal point.

The general form of this option is **/Fx.y**, where the x value sets the minimum number of digits to the right of the decimal place and the y value sets the maximum number of digits. You can use **=x,y** instead of **=x.y** if the comma is your decimal separator. Both values can range from 0 to 8; if x is greater than y, it is ignored. You can specify either or both values: /F2.5, /F2, and /F.5 are all valid entries. See the EvalMax and EvalMin directives to set the precision when Take Command starts; see the @EVAL function if you want to set the precision for a single computation.

/G (Numeric separators) This option sets the decimal and thousands separator characters. The format is **/Gxy** where "x" is the new decimal separator and "y" is the new thousands separator. Both characters must be included. The only valid settings are **/G.,** (period is the decimal separator, comma is the thousands separator); **/G,.** (the reverse); or **/G0** to remove any custom setting and use the default separators associated with your current country code (this is the default).

The decimal separator is used for @EVAL, numeric IF and IFF tests, version numbers, and other similar uses. The thousands separator is used for numeric output, and is skipped when performing calculations in @EVAL.

/I (Internal) This option allows you to disable or enable internal commands. To disable a command, precede the command name with a minus [**-**]. To re-enable a command, precede it with a plus [**+**]. For example, to disable the internal LIST command to force Take Command to use an external command:

```
c:\> setdos /i-list
```

/M (Mode) This option controls the initial line editing mode. To start in overstrike mode at the beginning of each command line, use **/M0**. To start in insert mode, use **/M1** (the default). Also see the EditMode directive.

/N (No clobber) This option controls output redirection. **/N0** means existing files will be

overwritten by output redirection (with **>**) and that appending (with **>>**) does not require the file to exist already. This is the default. **/N1** means existing files may not be overwritten by output redirection, and that when appending the output file must exist. A **/N1** setting can be overridden with the [**!**] character. 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. The default is the ampersand [**&**]. The parameter character is saved by SETLOCAL and restored by ENDLOCAL.

If you want to share batch files or aliases among 4DOS, 4OS2, 4NT, and Take Command, see Special Character Compatibility for details on selecting compatible parameter characters for all the products you use. Also see the ParameterChar directive.

/S (Shape) This option sets the cursor width. The format is **/So:i** where **o** is the width for overstrike mode, and **i** is the width for insert mode. The width is entered as a percentage of the total character width. The default values are 100:15 (a 100% or block cursor for overstrike mode, and a 15% or thin line 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%. Also see the CursorOver and CursorIns directives.

/U (Upper) This option controls the default case (upper or lower) for filenames displayed by internal commands like COPY and DIR. **/U0** displays file names in lower case (the default). **/U1** displays file names in the traditional upper case. Also see the UpperCase directive.

/V (Verbose) This option controls the default for command echoing in batch files. **/V0** disables echoing of batch file commands unless ECHO is explicitly set ON. **/V1**, the default setting, enables echoing of batch file commands unless ECHO is explicitly set OFF. Also see the BatchEcho directive.

/V2 forces echoing of all batch file commands, even if ECHO is set OFF or the line begins with an "@". This allows you to turn echoing on for a batch file without editing the batch file and removing the ECHO OFF command(s) within it. **/V2** is intended for debugging, and can be set with SETDOS, but not with the configuration dialogs or the BatchEcho directive in the *TCMD.INI* file. For more information on batch file debugging, see **/Y** below.

/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 Take Command 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 the built-in batch file debugger. The debuggger allows you to "single-step" through a batch file line by line, with the file displayed in a popup window as it executes. For complete details on using the debugger see Debugging Batch Files (this topic also covers additional debugging techniques which do not require stepping through each line individually).

To start the debugger, insert a SETDOS /Y1 command at the beginning of the portion of the batch file you want to debug, and a SETDOS /Y0 command at the end.

You cannot use the batch debugger with REXX files. It can only be used with normal Take Command batch files.

You can also invoke SETDOS /Y1 from the prompt, but because the debugger is automatically turned off whenever the command processor returns to the prompt, you must enter the SETDOS command and the batch file name on the same line, for example:

```
c:\> setdos /y1 ^ mybatch.btm
```

SETLOCAL

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

Format: **SETLOCAL**

See also: ENDLOCAL.

Usage

SETLOCAL is used in batch files to save the default disk drive and directory, the environment, the alias list, and the command separator, escape character, parameter character, decimal separator, and thousands separator. You can then change their values and later restore the original values with ENDLOCAL.

For example, this batch file fragment saves everything, removes all aliases so that user aliases will not affect batch file commands, changes the disk and directory, changes the command separator, runs a program, and then restores the original values:

```
setlocal
unalias *
cdd d:\test
setdos /c~
program ~ echo Done!
endlocal
```

SETLOCAL and ENDLOCAL are not nestable within a batch file. However, you can have multiple, separate SETLOCAL / ENDLOCAL pairs within a batch file, and nested batch files can each have their own SETLOCAL / ENDLOCAL. You cannot use SETLOCAL in an alias or at the command line.

An ENDLOCAL is performed automatically at the end of a batch file if you forget to do so. If you invoke one batch file from another without using CALL, the first batch file is terminated, and an automatic ENDLOCAL is performed; the second batch file inherits the settings as they were prior to any SETLOCAL.

SHIFT

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

Format: **SHIFT [*n* | /n]**

n: Number of positions to shift.

Usage

SHIFT is provided for compatibility with older batch files, where it was used to access more than 10 parameters. Take Command 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).

For example, create a batch file called *TEST.BAT*:

```
echo %1 %2 %3 %4
shift
echo %1 %2 %3 %4
shift 2
echo %1 %2 %3 %4
shift -1
echo %1 %2 %3 %4
```

Executing *TEST.BAT* produces the following results:

```
c:\> test one two three four five six seven

one two three four
two three four five
four five six seven
three four five six
```

If you add a slash before the value ***n***, the value determines the postion at which to begin the shift. For example,

```
shift /2
```

leaves parameters %0 and %1 unchanged, and moves the value of %3 to postion %2, %4 to %3, etc. The value after the slash cannot be negative, and shifts performed with the slash cannot be undone later in the batch file.

START

Purpose: Start a program in another session or window.

Format: **START [/C /CM /D*path* /E /EXIT /INV /K /MAX /MIN POS=row,col,width,height /WAIT] [*command*]**

path: Startup directory.

command: Command to be executed.

/C(lose when done)	**/INV**(isible)
/CM (Caveman)	**/K**(eep when done)
/D(irectory)	**/MAX**(imized)
/E(nvironment)	**/MIN**(imized)
/EXIT (exit Windows)	**/WAIT** (for session to finish)

Usage

START is used to begin a new Windows session, and optionally run a program in that session. If you use START with no parameters, it will begin a new Take Command session. If you add a ***command***, START will begin a new session or window and execute that command.

START will return to the Take Command prompt immediately (or continue a batch file), without waiting for the progrqam to complete, unless you use **/CM** or **/WAIT**.

/MAX and **/MIN** allow you to start a character-mode windowed session in a maximized window or a minimized window. 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 sessions); **/K** allows you to keep the session open and go to a prompt (the default for character mode sessions).

If the ***progname*** is the name of a directory instead of an executable program, Take Command will start Windows Explorer in the specified directory if you are using Windows 95 or Windows NT 4.0 or later. (Explorer must be in the PATH, the *\WINDOWS* directory, or the *\WINDOWS\SYSTEM* directory for this feature to work correctly.)

Options

/C (Close) The session or window is closed when the application ends.

/CM (Caveman) Run a DOS application under Caveman. Use this option to force a DOS application to run under Caveman even if Caveman is not the default method for starting DOS programs. For more details see Take Command and DOS Applications. **/CM** will be ignored if *CAVEMAN.386* is not loaded. Take Command always waits for the application to finish when **/CM** is used, even if you do not use **/WAIT**.

/D (Directory) Specifies the startup directory. Include the directory name immediately after the **/D**, with no intervening spaces or punctuation. Due to limitations in the way Windows starts DOS programs, **/D** is ignored when starting DOS applications.

/E (Environment) Specifies that a Windows application should receive Take Command's version of the environment, not the environment that was active when Windows was started. By default, all applications receive a copy of the Windows startup environment and not any changes that were made by Take Command. DOS applications that are run

under Caveman always receive Take Command's version of the environment; those that are run with the START command always receive the Windows startup environment regardless of whether you use the **/E** switch.

A bug in Windows sometimes prevents an application from starting when you use the **/E** option. You can sometimes work around this bug by changing the length of the total environment by a few bytes. To do so, either add a new entry to the environment or remove an entry. You can modify the environment with the SET command

/EXIT Exit Windows to run a DOS program, then restart Windows.

/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.

/MAX (Maximized) Start the session or window maximized.

/MIN (Minimized) Start the session or window minimized.

/WAIT Wait for the new session or window to finish before continuing.

This switch is ignored when starting DOS programs under WIN-OS/2, because there is no way for Take Command to determine when a DOS program run under WIN-OS/2 has finished.

SWITCH

Purpose: Select commands to execute based on a value.

Format: **SWITCH *expression***

CASE value1 [.OR. value2] …

commands

CASE value3

commands

[DEFAULT

commands]

ENDSWITCH

expression: An environment variable, internal variable, variable function, text string, or a combination of these elements, that is used to select a group of commands.

value1, ***value2***, *etc.*: A value to test or multiple values connected with **.OR.**

commands: One or more commands to execute if the expression matches the value. If you use multiple commands, they must be separated by command separators or placed on separate lines of a batch file.

See also: IF, and IFF.

Usage:

SWITCH can only be used in batch files. It allows you to select a command or group of commands to execute based on the possible values of a variable or a combination of variables and text.

The SWITCH command is always followed by an ***expression*** created from environment variables, internal variables, variable functions, and text strings, and then by a sequence of CASE statements matching the possible ***values*** of that ***expression***. If one of the ***values*** in a CASE statement matches the ***expression***, the commands following that CASE statement are executed, and all subsequent CASE statements and the commands which follow them are ignored. If no matches are found, the commands following the optional DEFAULT statement are executed. If there are no matches and there is no DEFAULT statement, no commands are executed by SWITCH.

After all of the commands following the CASE or DEFAULT statement are executed, the batch file continues with the commands that follow ENDSWITCH.

You must include a command separator or new line after the ***expression***, before each CASE or DEFAULT statement, before each command, and before ENDSWITCH. You can link values in a CASE statement only with **.OR.** (but not with **.AND.** or **.XOR.**).

For example, the following batch file fragment displays one message if the user presses **A**, another if user presses **B** or **C**, and a third if the user presses any other key:

```
inkey Enter a keystroke: %%key
switch %key
```

```
case A
   echo It's an A
case B .or. C
   echo It's either B or C
default
   echo It's not A, B, or C
endswitch
```

In the example above, the value of a single environment variable was used for the ***expression***. You will probably find that this is the best method to use in most situations. However, you can use other kinds of expressions if necessary. The first example below selects a command to execute based on the length of a variable, and the second bases the action on a quoted text string stored in an environment variable:

```
switch %@len[%var1]
case 0
   echo Missing var1
case 1
   echo Single character
...
endswitch

switch "%string1"
case "This is a test"
   echo Test string
case "The quick brown fox"
   echo It's the fox
...
endswitch
```

The SWITCH and ENDSWITCH commands must be on separate lines, and cannot be placed within a command group, or on the same line as other commands (this is the reason SWITCH cannot be used in aliases). However, commands within the SWITCH block can use command groups or the command separator in the normal way.

SWITCH commands can be nested.

You can exit from all SWITCH / ENDSWITCH processing by using GOTO to a line past the last ENDSWITCH.

TEE

Purpose: Copy standard input to both standard output and a file.

Format: **TEE [/A] *file*...**

file: One or more files that will receive the "tee-d" output.

/A(ppend)

See also: Y and the redirection options.

Usage

TEE is normally used to "split" the output of a program so that you can see it on the display and also save it in a file. It can also be used to capture intermediate output before the data is altered by another program or command.

TEE gets its input from standard input (usually the piped output of another command or program), and sends out two copies: one goes to standard output, the other to the *file* or *files* that you specify. TEE is not likely to be useful with programs which do not use standard output, because these programs cannot send output through a pipe.

For example, to search the file *DOC* for any lines containing the string "Take Command", make a copy of the matching lines in *TC.DAT*, sort the lines, and write them to the output file *TC.DAT*:

```
c:\> find "Take Command" doc | tee tc.dat | sort > tcs.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, the previous command writes its output to a temporary file. When that command finishes, TEE reads the temporary file, display the output, and writes it to the file(s) named in the TEE command.

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
4 - Exit
endtext
inkey /k"1234" Enter your selection:  %%key
```

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
Mon  Dec 22, 1997  9:30:06
New time (hh:mm:ss):
```

TIME defaults to 24-hour format, but you can optionally enter the time in 12-hour format by appending "a", "am", "p", or "pm" to the time you enter.

For example, to enter the time as 9:30 am:

```
c:\> time 9:30 am
```

Windows 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)
/2 (stopwatch #2)
/3 (stopwatch #3)
/S(plit)

Usage

The TIMER command turns a system stopwatch on and off. When you first run TIMER, the stopwatch starts:

```
c:\> timer
Timer 1 on:  12:21:46
```

When you run TIMER again, the stopwatch stops and the elapsed time is displayed:

```
c:\> timer
Timer 1 off:  12:21:58
Elapsed time: 0:00:12.06
```

There are three stopwatches available (1, 2, and 3) so you can time multiple overlapping events. By default, TIMER uses stopwatch #1.

TIMER is particularly useful for timing events in batch files. For example, to time both an entire batch file, and an intermediate section of the same file, you could use commands like this:

```
rem Turn on timer 1
timer
rem Do some work here
rem Turn timer 2 on to time the next section
timer /2
rem Do some more work
echo Intermediate section completed
rem Display time taken in intermediate section
timer /2
rem Do some more work
rem Now display the total time
timer
```

The smallest interval TIMER can measure depends on the operating system you are using, your hardware, and the interaction between the two. However, it should never be greater than .06 second. The largest interval is 23 hours, 59 minutes, 59.99 seconds.

Options

/1 Use timer #1 (the default).

/2 Use timer #2.

/3 Use timer #3.

/S (Split) Display a split time without stopping the timer. To display the current elapsed time but leave the timer running:

```
c:\> timer /s
Timer 1 elapsed: 0:06:40.63
```

ON Start the timer regardless of its previous state (on or off). Otherwise the TIMER command toggles the timer state (unless **/S** is used).

TITLE

Purpose: Change the window title.

Format: **TITLE *title***

title: The new window title.

See also: ACTIVATE and WINDOW.

Usage

TITLE changes the text that appears in the caption bar at the top of the Take Command window. It is included for compatibility with traditional character-mode command processors (like WIndows NT's *CMD.EXE*). You can also change the window title with the WINDOW command or the ACTIVATE command.

The title text should not be enclosed in quotes unless you want the quotes to appear as part of the actual title.

To change the title of the current window to "Take Command Test":

```
c:\> title Take Command Test
```

TOUCH

Purpose: Change a file's date and time stamps.

Format: **TOUCH [/C /D[mm-dd-yy] /E /F /Q /T[hh:mm]] *file...***

file: One or more files whose date and/or time stamps are to be changed.

/C(reate file)	**/F**(orce read-only files)
/D(ate)	**/Q**(uiet)
/E (No error messages)	**/T**(ime)

File Selection

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

Usage:

TOUCH is used to change the date and / or time of a file. You can use it to be sure that particular files are included or excluded from an internal command, backup program, compiler MAKE utility , or other program that selects files based on their time and date stamps, or to set a group of files to the same date and time for consistency.

TOUCH should be used with caution, and in most cases should only be used on files you create. Many programs depend on file dates and times to perform their work properly. In addition, many software manufacturers use file dates and times to signify version numbers. Indiscriminate changes to date and time stamps can lead to confusion or incorrect behavior of other software.

TOUCH normally works with existing files, and will display an error if the ***file*** you specify does not exist, or has the read-only attribute set. To create the ***file*** if it does not already exist, use the **/C** switch. To force a date and time change for read-only files, use the **/F** switch.

TOUCH displays the date, time, and full name of each file whose timestamp is modified. To disable this output, use **/Q**.

If you don't specify a date or a time, TOUCH will default to the current date and time from your system clock. For example, to set the time stamp of all *.C* files in the current directory to the current date and time:

```
d:\source> touch *.c
 6-12-97 11:13:58  D:\SOURCE\MAIN.C
 6-12-97 11:13:58  D:\SOURCE\INIT.C
 ...
```

If you specify a date but not a time, the time will default to the current time from your system clock. Similarly, if you specify a time but not a date, the date will be obtained from the system clock.

Options:

/C (Create file) Create the ***file*** (as a zero-byte file) if it does not already exist. You cannot use wildcards with **/C**, but you can create multiple ***files*** by listing them individually on the command line.

/D (Date) Specify the date that will be set for the selected files. If the date is not specified, TOUCH will use the current date. The date must be entered using the proper format for your

current country settings.

/E (No error messages) Suppress all non-fatal error messages, such as "File not found." Fatal error messages, such as "Drive not ready," will still be displayed. This option is most useful in batch files.

/F (Force read-only files) Remove the read-only attribute from each file before changing the date and time, and restore it afterwards. Without **/F**, attempting to change the date and time on a read-only file will usually cause an error.

/Q (Quiet) Do not display the new date and time and the full name for each file.

/T (Time) Specify the time that will be set for the selected files in hh:mm format. If the time is not specified, TOUCH will use the current time.

TREE

Purpose: Display a graphical directory tree.

Format: **TREE [/A /B /F /H /P /S /T[:acw]] *dir...***

dir: The directory to use as the start of the tree. If more than one directory is specified, TREE will display a directory tree for each.

/A(SCII)
/B(are)
/F(iles)
/H(idden directories)
/P(ause)
/S (file size)
/T(ime and date)

File Selection

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

Usage:

The TREE command displays a graphical representation of the directory tree using standard or extended ASCII characters. For example, to display the directory structure on drive C:

```
c:\> tree c:\
```

TREE uses the standard line drawing characters in the U.S. English extended ASCII character set. If your system is configured for a different country or language, or if you use a font which does not include these line drawing characters, the connecting lines in the tree display may not appear correctly on your screen. To correct the problem, use **/A**, or configure Take Command to use a font such as Terminal, which contains standard extended ASCII characters.

You can print the display, save it in a file, or view it with LIST by using standard redirection symbols (see Redirection and Piping). Be sure to review the **/A** option before attempting to print the TREE output. The options, discussed below, specify the amount of information included in the display.

Options:

/A (ASCII) Display the tree using standard ASCII characters. You can use this option if you want to save the directory tree in a file for further processing or print the tree on a printer which does not support the graphical symbols that TREE normally uses.

/B (Bare) Display the full pathname of each directory, without any of the line-drawing characters.

/F (Files) Display files as well as directories. If you use this option, the name of each file is displayed beneath the name of the directory in which it resides.

/H (Hidden) Display hidden as well as normal directories. If you combine **/H** and **/F**, hidden files are also displayed.

/P (Pause) Wait for a key to be pressed after each screen page before continuing the display. Your options at the prompt are explained in detail under Page and File Prompts.

/S (Size) Display the size of each file. This option is only useful when combined with **/F**.

/T (Time and date) Display the time and date for each directory. If you combine **/T** and **/F**, the time and date for each file will also be displayed.

TRUENAME

Purpose: Find the full, true path and file name for a file

Format: **TRUENAME *file***

file: The file whose name TRUENAME will report.

See also: The @TRUENAME variable function.

Usage:

Default directories, as well as the JOIN and SUBST external commands, can obscure the true name of a file. TRUENAME "sees through" these obstacles and reports the fully qualified name of a file.

The following example uses TRUENAME to get the true pathname for a file:

```
c:\> subst d: c:\util\test
c:\> truename d:\test.exe
c:\util\test\test.exe
```

TRUENAME can handle simple drive substitutions such as those created by JOIN, SUBST, or most network drive mappings. However, it may not be able to correctly determine the true name if you use "nested" JOIN or SUBST commands, or a network which does not report true names properly.

TYPE

Purpose: Display the contents of the specified file(s).

Format: **TYPE [/A:[[-]rhsda] /L /P] *file*...**

file: The file or list of files that you want to display.

/A: (Attribute select) **/P**(ause)
/L(ine numbers)

See also: LIST.

File Selection

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

Usage

The TYPE command displays a file. It is normally only useful for displaying ASCII text files. Executable files (*.COM* and *.EXE*) and many data files may be unreadable when displayed with TYPE because they include non-alphanumeric characters.

To display the files *MEMO1* and *MEMO2*:

```
c:\> type /p memo1 memo2
```

You can press **Ctrl-S** to pause TYPE's display and then any key to continue.

To display text from the clipboard use **CLIP:** as the file name. CLIP: will not return any data if the clipboard does not contain text. See Redirection and Piping for more information on CLIP:.

You will probably find LIST to be more useful for displaying files. However, the TYPE /L command used with redirection is useful if you want to add line numbers to a file, for example:

```
c:\> type /l myfile > myfile.num
```

Options

/A: (Attribute select) Select only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [**-**] will select files that do **not** have that attribute set. The colon [**:**] after **/A** is required. The attributes are:

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

If no attributes are listed at all (*e.g.*, **TYPE /A:**), TYPE will select all files and subdirectories including hidden and system files. If attributes are combined, all the specified attributes must match for a file to be selected. For example, **/A:RHS** will select only those files with all three attributes set.

/L (Line numbers) Display a line number preceding each line of text.

/P (Pause) Prompt after displaying each page. Your options at the prompt are explained in

detail under Page and File Prompts.

UNALIAS

Purpose: Remove aliases from the alias list.

Format: **UNALIAS [/Q /R *file*...] *alias*...**

or

UNALIAS *

alias: One or more aliases to remove from memory.
file: One or more files containing variable definitions.

/Q(uiet) **/R**(ead file)

See also: ALIAS and ESET.

Usage

Take Command maintains a list of the aliases that you have defined. The UNALIAS command will remove aliases from that list. You can remove one or more aliases by name, or you can delete the entire alias list by using the command **UNALIAS ***.

For example, to remove the alias DDIR:

```
c:\> unalias ddir
```

To remove all the aliases:

```
c:\> unalias *
```

If you keep aliases in a file that can be loaded with the ALIAS /R command, you can remove the aliases by using the UNALIAS /R command with the same file name:

```
c:\> unalias /r alias.lst
```

This is much faster than removing each alias individually in a batch file, and can be more selective than using UNALIAS *.

You can also remove individual aliases with the Alias dialog.

Options

/Q (Quiet) Prevents UNALIAS from displaying an error message if one or more of the aliases does not exist. This option is most useful in batch files, for removing a group of aliases when some of the aliases may not have been defined.

/R (Read) Read the list of aliases to remove from a file. The file format should be the same format as that used by the ALIAS /R command. You can use multiple files with one UNALIAS /R command by placing the names on the command line, separated by spaces:

```
c:\> unalias /r alias1.lst alias2.lst
```

UNSET

Purpose: Remove variables from the environment or disable file associations inherited from Windows..

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

or

UNSET *

name: One or more variables to remove or file types to disable.
file: One or more files containing variable definitions.

/Q(uiet) **/R**(ead from file)

See also: ESET and SET.

Usage

UNSET removes one or more variables from the environment, or disables file associations "inherited" from Windows.

For example, to remove the environment variable CMDLINE:

```
c:\> unset cmdline
```

If you use the command **UNSET ***, all of the environment variables will be deleted:

```
c:\> unset *
```

You can also remove individual variables from the environment with the Environment dialog.

UNSET can be used in a batch file, in conjunction with the SETLOCAL and ENDLOCAL commands, to clear the environment of variables that may cause problems for applications run from that batch file.

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

You can also use UNSET to disable direct file associations that Take Command has inherited from Windows (see Windows File Associations for additional details). If the first character of the variable name is a period [**.**], UNSET will look first for a matching environment variable to remove. If it doesn't find one, it will next look in the list of direct file associations it has loaded from Windows. UNSET will not modify Windows' file associations, just Take Command's copy of the associations.

Use caution when removing environment variables, and especially when using UNSET *. Many programs will not work properly without certain environment variables; for example, Take Command uses PATH and CDPATH.

Options

/Q (Quiet) Prevents UNSET from displaying an error message if one or more of the variables or associations does not exist. This option is most useful in batch files, for removing a group of variables when some of the variables may not have been defined.

/R (Read) Read environment variables to UNSET from a file. This much faster than using

multiple UNSET commands in a batch file, and can be more selective than UNSET *. The file format should be the same format as that used by the SET /R command (see SET for more details).

VER

Purpose: Display the Take Command and operating system versions.

Format: **VER [/R]**

/R(evision level)

Usage

Version numbers consist of a one-digit major version number, a separator, and a one- or two-digit minor version number. VER uses the default decimal separator defined by the current country information. The VER command displays both version numbers:

```
c:\> ver
Take Command/16 2.01A   Windows Version is 3.1
```

Option

/R (Revision level) Display the Take Command and Windows internal revision levels, plus your Take Command serial number and registered name.

/R also displays whether DOS is loaded into the high memory area (HMA), is resident in ROM, or is in normal base memory. This output is only meaningful in MS-DOS or PC-DOS version 5.0 or above, and in Win-OS/2.

VERIFY

Purpose: Enable or disable disk write verification or display the verification state.

Format: **VERIFY [ON | OFF]**

Usage

DOS maintains an internal verify flag. When the flag is on, DOS attempts to verify each disk write by making sure that the data written to the disk can be read back successfully into the computer. It does **not** compare the data in memory with the data actually placed on disk to fully verify the disk write process.

If used without any parameters, VERIFY will display the state of the verify flag:

```
c:\> verify
VERIFY is ON
```

VERIFY is off when the system boots up. Once it is turned on with the VERIFY ON command, it stays on until you use the VERIFY OFF command or until you reboot.

Verification will slow your disk write operations slightly (the effect is not usually noticeable).

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, with OS/2, or with Windows has a volume serial number.

The VOL command will display the volume label and, if available, the volume serial number of a disk volume. If the disk doesn't have a volume label, VOL will report that it is "unlabeled." If you don't specify a drive, VOL displays information about the current drive:

```
c:\> vol
Volume in drive C: is MYHARDDISK
```

If available, the volume serial number will appear after the drive label or name.

To display the disk labels for drives A and B:

```
c:\> vol a: b:
Volume in drive A: is unlabeled
Volume in drive B: is BACKUP_2
```

VSCRPUT

Purpose: Display text vertically in the specified color.

Format: **VSCRPUT *row col* [BRIght] *fg* ON [BRIght] *bg text***

row: Starting row number.
col: Starting column number.
fg: Foreground text color.
bg: Background text color.
text: The text to display.

See also: SCRPUT.

Usage

VSCRPUT writes text vertically on the screen rather than horizontally. Like the SCRPUT command, it uses the colors you specify to write the text. VSCRPUT can be used for simple graphs and charts generated by batch files.

The ***row*** and ***column*** are zero-based, so on a 25 line by 80 column display valid rows are 0 - 24 and valid columns are 0 - 79. The maximum ***row*** value is determined by the current height of the Take Command window. The maximum ***column*** value is determined by the current virtual screen width (see Resizing the Take Command Window for more information). VSCRPUT checks for a valid *row* and *column*, and displays a "Usage" error message if either value is out of range.

You can also specify the *row* and *column* as offsets from the current cursor position. Begin the value with a plus sign [**+**] to move down the specified number of rows or to the right the specified number of columns before displaying text, or with a minus sign [**-**] to move up or to the left.

If you specify 999 for the ***row***, VSCRPUT will center the text vertically in the Take Command window. If you specify 999 for the ***column***, VSCRPUT will center the text horizontally.

VSCRPUT does not move the cursor when it displays the ***text***.

The following batch file fragment displays an X and Y axis and labels them:

```
cls bright white on blue
drawhline 20 10 40 1 bright white on blue
drawvline 2 10 19 1 bright white on blue
scrput 21 20 bright red on blue X axis
vscrput 8 9 bright red on blue Y axis
```

WINDOW

Purpose: Minimize or maximize the current window, restore the default window size, or change the window title.

Format: **WINDOW [MIN | MAX | RESTORE | /POS=x,y,width, height] ["*title* "]**

title: A new title for the window.

/POS(ition)

See also: ACTIVATE and TITLE.

Usage

The WINDOW command is used to control the appearance and title of the current window. WINDOW can only be used to specify one change to the current window at a time; to perform more than one operation, you must use multiple WINDOW commands (see examples below).

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.

For example, to maximize the current window and change it's title, you must perform two WINDOW commands:

```
c:\> window max
c:\> window "JP Software / Take Command/16"
```

Option

/POS(ition) Set the window screen position and size. The syntax is **/POS=row, col, width, height**, where the values are specified in pixels. **Row** and **col** refer to the position of the top left corner of the window relative to the bottom left corner of the screen.

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.

See Piping for more information on pipes.

Starting Take Command

You will typically start Take Command from an item in one of the Program Manager groups. The installation software will create a Take Command group, and a desktop object within it which starts Take Command. Usually this item is sufficient, but if you prefer you can create multiple desktop objects or items to start Take Command with different startup commands or options, or to run different batch files or other commands; see Creating Desktop Objects for details. You can use these items to run commonly-used commands and batch files directly from the desktop.

Each item or icon represents a different Take Command window. You can set any necessary command line parameters for Take Command such as a command to be executed, any desired switches, and the name and path for the *TCMD.INI* file. See Command Line Options for more information on startup command line switches and options.

When you configure a Take Command item, place the full path and name for the *TCMD.EXE* file in the Command Line field, and put any startup options that you want passed to Take Command (*e.g.*, the name of a startup batch file) after the *TCMD.EXE* file name. For example:

```
Command Line:        C:\TCMD20\TCMD.EXE C:\GO.BAT
Working directory:   C:\
```

You do not need to use the Change Icon button, because *TCMD.EXE* already contains an icon.

When Take Command starts it automatically runs the optional *TCSTART* batch file. You can use this file to load aliases and environment variables and otherwise initialize Take Command.

You can also place the name of a batch file, internal or external command, or alias at the end of the Command Line field for any item (as shown in the example above). The batch file, command, or alias will be executed after *TCSTART* but before the first prompt is displayed.

Like DOS programs, each Windows program has a command line which can be used to pass information to the program when it starts. The command line is entered in the Command Line field for each item in a Program Manager group (or each item defined under another Windows shell), and consists of the name of the program to execute, followed by any startup options.

The Take Command startup command line does not need to contain any information. When invoked with an empty command line, Take Command will configure itself from the *TCMD.INI* file, run *TCSTART*, and then display a prompt and wait for you to type a command. However, you may add information to the startup command line that will affect the way Take Command operates.

Creating Desktop Objects

This section explains how to create new objects or icons to run Take Command on your Program Manager desktop. It assumes you are running Windows 3.1 or above or Windows for Workgroups, with Program Manager as the shell, and with a standard Windows desktop. If you are using an alternate shell, or you have altered your Windows desktop configuration substantially, you may need to take those changes into account as you read the instructions below.

When Take Command is installed it normally creates a program group which appears in the Program Manager, and includes items to run Take Command and its online help.

If you want to create a new item for Take Command/16 in any group, use the Program Manager's File / New menu selection. Set the command line to *d:\path\TCMD.EXE* (use the appropriate drive and path for your system). Set the Description and Working Directory fields to your desired values, then click OK to create the item.

No additional settings are required; the only required item is the drive and path for *TCMD.EXE*. However, you can put command-line switches, a command, or the name of a batch file at the end of the command line for any Take Command/16 item. This allows you to run specific commands or set configuration options when you start Take Command from that item.. See Command Line Options for details.

For more information on creating and configuring Program Manager items see your Windows documentation.

Command Line Options

Some of the options that Take Command recognizes are required in certain circumstances; others are available if you want finer control over the way the program starts.

The line that starts Take Command will typically include the program name with drive and path then include any switches for the program, for example:

```
c:\tcmd300\tcmd.exe @c:\tcmd300\tcmd.ini
```

Although the startup command line is usually very simple, you can add several options. The complete syntax for the startup command line is:

```
d:\path\program [d:\path] [@d:\path\inifile] [//iniline] [[/C] command]
```

Do not include the square brackets shown in the command line above. They are there to indicate that the items within the brackets are optional. Not all options are available in all products; see below for details.

If you include any of the options below, you should use them in the order that they are described. If you do not do so, you may find that they do not operate properly.

The following items can be included on the command line:

d:\path\program: The path and name of the executable program file (*TCMD.EXE*). It is required to start Take Command.

d:\path: This is the second **d:\path** in the command line above. It sets the drive and directory where the program is stored, called the **COMSPEC path**. This option is included for compatibility with character-mode command processors, but is not needed in normal use. Take Command can find its own directory without a COMSPEC path, and usually the COMSPEC variable should be left pointing to the default character mode command processor in use on your system, not changed to point to Take Command.

@d:\path\inifile: This option sets the path and name of the *.INI* file. You don't need this option if your *.INI* file is named *TCMD.INI* **and** it is either in the same directory as the executable program or in the Windows directory. This option is most useful if you want to start the program with a specific and unique *.INI* file.

//iniline: This option tells Take Command to treat the text appearing between the *//* and the next space or tab as an *.INI* directive. The directive should be in the same format as a line in the *.INI* file, but may not contain spaces, tabs, or comments. Directives on the command line override any corresponding directive in the *.INI* file. This option may be repeated. It is a convenient way to place a few simple directives on the startup line without having to modify or create a new *.INI* file.

[/C] command: This option tells Take Command to run a specific command after starting. The command will be run after *TCSTART*, and before the prompt is displayed. The command can be any valid alias, internal or external command, or batch file. All other startup options must be placed before the command, because Take Command 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**, Take Command will execute the command and then exit and return to the parent program or the desktop without displaying a prompt.

For example, this command line will start Take Command, execute any TCSTART file you have created, execute the file *START.BTM*, and then display the prompt:

```
c:\tcmd\tcmd.exe c:\tcmd\start.btm
```

This command line will start Take Command, execute any TCSTART file you have created, execute the file *PROCESS.BTM*, and then exit when *PROCESS.BTM* is done. The prompt will not be displayed by this session:

```
c:\tcmd\tcmd.exe /c c:\tcmd\process.btm
```

What's New?

This section provides a comprehensive list of what's changed since our previous release, version 1.02. Maintenance changes made between versions 2.00 and 2.01 are indicated by **2.01** in the left margin.

This topic does not explain how to use each new feature. Instead, where appropriate we have provided links below to the detailed help topics containing additional usage information or other documentation.

Some of the descriptions here may be more detailed than you need; if you aren't using a feature, feel free to skip to the next item. If you are new to Take Command with version 2.0, you can skip this topic entirely.

This topic is divided into the following subtopics:

The major new features in this release include:

» **Extended Directory Searches** allow you to change to a directory anywhere on your system by entering only part of its name. They must be explicitly enabled before you can use them. See Directory Navigation for complete details.

» You can **directly execute files with Windows file associations** from the Take Command prompt.

» **New commands** include:

ECHOERR and **ECHOSERR**: Display output on the standard error device (rather than the usual standard output device).

OPTION: Offers complete configuration adjustment, either through interactive dialogs or on the command line.

SWITCH: Provides for "case" statements in batch files.

TOUCH: Adjusts file dates and times.

TREE: Displays the directory tree, with or without file names, in a variety of formats.

» New **file exclusion ranges** provide a convenient way to exclude files from any internal command -- faster and more flexible than using EXCEPT.

» The new **batch file debugger** can execute each line step by step, process or trace into additional batch files, and display variables, aliases, and expanded commands at each step.

There over 100 additional new features beyond those mentioned here. See the individual subtopics listed above for details.

What's New: General Features and Enhancements

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

» Added a complete batch file debugger. The debugger displays the batch file in a window and allows you to execute each line step by step, process or trace into additional batch files and subroutines, and display variables and aliases at each step. See Batch File Debugging for complete details.

» Added a new type of range called a "file exclusion range". The syntax is "/[!filename ...]" which excludes filenames that match those inside the brackets. For example, to display everything but *.TXT* and *.BAK* files:

```
dir /[!*.txt *.bak] *.*
```

Exclusion ranges are faster, more flexible, and more reliable than the similar EXCEPT command when excluding files from processing by internal commands. However they do not work with external commands.

» Popup windows (for filename completion, command history recall, etc.) now allow you to search for a line within the window contents by typing the first few characters of the line.

» You can now redirect to and from the clipboard by using the pseudo-device name CLIP:. (In 4DOS, this feature will only work if you're in a DOS session in Windows -- it does not support transfers to/from the OS/2 clipboard). For example, to redirect DIR to the clipboard:

```
dir *.doc > clip:
```

» The online helphas been reorganized to make it easier to navigate through the main topics, and includes additional reference information, reference tables, and a glossary.

» The default maximum file description length is now 511 bytes in all products.

» Two new characters can now follow the escape character: An escape followed by a 'q' will substitute a double quote; an escape followed by a 'k' will substitute a back quote.

» The decimal and thousands characters used in @EVAL and in displayed version numbers and other similar locations are now controllable with the DecimalChar and ThousandsChar directives in the *.INI* file, the corresponding options in the configuration or OPTION dialogs, and the SETDOS /G command. These characters are saed by SETLOCAL and restored by ENDLOCAL. This is intended as an aid to those writing batch files which perform arithmetic operations and which may be used in countries with differing separator characters.

» The directory stack size used by PUSHD and POPD has been increased from 255 to 511 bytes to leave adequate room for long directory names.

What's New: Command Line Editing

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

- » Extended directory searches can be used directly from the command line for quick directory navigation; see Automatic Directory Changes or Directory Navigation for details.

- » Made several enhancements to Filename completion. including:

  - º The Ctrl-A key, which toggles between long and short filenames for filename completion, can now be hit at any point during command line entry -- not just during filename completion. For example, if you hit Ctrl-A at the beginning of the command line, all filenames subsequently returned for that line will be short names (until you hit Ctrl-A again).

  - º Filename completion can now be customized for individual commands via the new FileCompletion *.INI* directive (or environment variable). For example, you can configure 4DOS to complete only the names of *.TXT* files when the command line starts with the name of your text editor, or to display only directory names when you are entering a CD command.

  - º The F7 filename completion popup window now sorts the filename list alphabetically.

- » You can now expand aliases immediately while still on the command line with the Ctrl-F key.

- » Command line history recall will now stop at the beginning and end of the history list rather than wrapping around, if you set HistWrap to No in the *.INI* file or through the configuration dialogs.

What's New: GUI-Related Changes

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

» Take Command now supports ANSI screen commands.

» In the Find Files dialog, double-clicking on the filename, then clicking on LIST in the Info dialog will now start LIST (in a separate copy of Take Command) with the pointer at the first matching string in the file.

» When selecting colors in the configuration dialogs, if either the foreground or background is set to (Default) then the default colors will be used for that option.

» A double (left) click in the Take Command window now selects the word under the mouse pointer.

What's New: Command Changes

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

» ATTRIB: Added the /E switch to disable display of non-fatal errors. Also, ATTRIB now allows underscores in the attribute string, so that you can get a result from the @ATTRIB variable function and feed it directly to the ATTRIB command.

» CD and CDD: Now support extended directory searches, which allow you to change to a directory anywhere on your system by entering only part of its name. The CDD /S switch builds the extended directory search database. Extended directory searches must be explicitly enabled before you can use them. See Directory Navigation for complete details.

» CDD: Added the /A switch to display the current directory for all existing and ready drives from C: to Z:.

2.01 » CLS: Fixed a problem with display updates being turned off after an invalid switch was entered.

» COPY: Added several switches:

/E Disable display of non-fatal errors.

/K Preserve read-only attributes during a COPY.

/X Clear the archive bit from the source file after a successful copy.

/Z Overwrite read-only target files.

» DEL: Added two switches:

/E Disable display of non-fatal errors.

/W Clear the file to 0's before deleting it.

2.01 Fixed a problem with /W and 0-byte files.

2.01 » DESCRIBE: Worked around a Novell Netware bug which caused trouble with descriptions with trailing drive specs (i.e., "file from drive D:").

» DIR: Added / modified two DIR switches:

/4 Now displays files between 1 and 9.9 Mb in tenths (i.e., "2.4M").

/G (New) Displays the allocated size instead of the file size.

2.01 Fixed several problems, including:

/4/Z did not display file sizes ending in ".9M" (1.9M, 2.9M, etc.) correctly.

/J did not display the descriptions.

/OGU was ignoring the 'U'; it will now display the files unsorted after the directory names.

"*.*" was incorrectly being appended to file specifications that ended in a question mark.

» DIRHISTORY: This new command has the same syntax as HISTORY, but it modifies the

directory history.

» DO: Added two new DO loop types:

º "DO x IN filename" retrieves each matching filename from a wildcard spec and inserts the value into the variable.

º "DO x IN @filename" retrieves each line in the file and inserts it into the variable.

2.01 Fixed a problem with "LEAVE" not closing the file handle on a "do var in @filename", and a similar problem with exiting the batch file with QUIT or CANCEL from inside a DO loop which had a file open.

2.01 » DRAWVLINE: Fixed a problem with connecting to a horizontal line on the right side.

» ECHOERR and ECHOSERR: These new commands are like ECHO and ECHOS, but output goes to the standard error device instead of standard output.

2.01 » ECHOS: Fixed a problem with aborting an ECHOS with a ^C while in a DO or FOR loop.

» ENDLOCAL: To aid in making batch files portable, SETLOCAL and ENDLOCAL now save and restore the command separator, escape character, parameter character, and decimal and thousands separators.

» FFIND: Added two new switches:

/I Do a literal match even if the text search string contains wildcard characters.

/R Start searching for text from the end backwards.

Also, the /X switch will now display the offset in both hex and decimal.

2.01 Added support for piping into FFIND. You can either specify CON for the filename, or if no filename is specified FFIND will detect whether STDIN is a pipe and use that.

2.01 Fixed a problem when an "access denied" error is received during a text search.

» FOR: Added several new switches for compatibility with Windows NT 4.0's *CMD.EXE*; see the command reference information for complete details. Also, added the ability to read lines from the clipboard.

2.01 Fixed a problem with combining /A:xx and /R.

2.01 Fixed a problem with combining /H and /R.

» GOTO: Added support for Windows NT 4.0's "GOTO :EOF" -- If there is no ":EOF" label, GOTO ends the current batch file (equivalent to a QUIT).

» IF / IFF: These commands have several changes, including:

º Support for nested conditional tests, with parentheses, *e.g.*:

```
if (%a == 1 .or. %b == 2) .and. %c == 3 echo something
```

See the command reference information for complete syntax rules.

º A new "IF DEFINED varname" test, which succeeds if the specified variable exists in

the environment. This is included for compatibility with Windows NT 4.0's *CMD.EXE*, and is the same as a test like:

```
if "%varname" ne "" ...
```

- The comparison tests now accept a leading decimal separator as a numeric character, provided the remainder of the string is numeric and does not contain additional decimal characters.

» KEYSTACK: You can now repeat special characters by following the key name or numeric code with the count enclosed in square brackets. For example, to repeat a carraige return (ASCII value 13) eight times:

```
keystack 13 [8]
```

» LIST: Add a range of enhancements, including:

- Added three new switches:

  /I Ignore case in a /T search.

  /R The search initiated by /T goes backwards from the end of the file.

  /T Search for text when LIST starts.

- Ctrl-PgUp and Ctrl-PgDn will go to the previous and next file in the current group, respectively.

- Ctrl-F searches backwards for a text string; Ctrl-N repeats the last search, searching backwards.

- Matching strings on the first page are now highlighted after a search.

- When piping output to LIST in most cases you no longer need the /S switch; for example, to view DIR's output in LIST you can now use:

```
dir | list
```

» MD: Added the /N switch to create a directory without updating the extended directory search database (useful for temporary directories).

» MOVE: Added the /E switch to disable display of non-fatal errors.

» OPTION: This new command can be used for two purposes. When invoked without parameters, it loads configuration dialogs which adjust most commonly-used settings in the *.INI* file. The dialogs provide a convenient method of adjusting configuration without manually editing the *.INI* file. OPTION can also be used to change specific settings on an individual basis with the OPTION Name=value ... syntax; see the command for complete details.

» PROMPT: Added the $+ metacharacter, which displays one + for each PUSHD level.

» REN / RENAME: Added the /E switch to disable display of non-fatal errors.

» RETURN: Now accepts an optional argument for the errorlevel to return. The errorlevel can be tested with %? or IF ERRORLEVEL.

» SCREEN, SCRPUT, and VSCRPUT: If you specify 999 for the row, the text will be centered

vertically; if you specify 999 for the column, the text will be centered horizontally.

» SELECT: You can now type characters from the start of a filename and the selection bar will jump to the first matching name. Due to this change, the key to popup LIST on the currently selected file has been changed from L to ^L. Also, added the /T:acw switch to select the date and time to use for display and sorting on LFN and NTFS drives.

» SETLOCAL: To aid in making batch files portable, SETLOCAL now saves the command separator, escape character, parameter character, and decimal and thousands separators; ENDLOCAL restores them.

» SHIFT: The new "/n" argument will start the shift at the specified argument -- i.e., "shift /2" moves %3 to %2, %4 to %3, etc.

» SWITCH: This new command provides a C-like switch construct for batch files. SWITCH scans each CASE statement looking for a matching value; if it finds one it executes the block of code inside that CASE statement, and then jumps to the end of the switch block (ENDSWITCH). If no CASE statement matches, SWITCH will execute the code in the (optional) DEFAULT block.

2.01 Fixed occasional problems with nested SWITCHes.

» TOUCH: This new command changes the date and/or time for a file or files, and can also create a new file with a specified date and time. You can specify a date and time or use the current system clock, and you can optionally change the last access / creation date and time fields on LFN and NTFS drives.

2.01 /T[acw] and /D[acw] now default to the current date and time. Previously when the "a", "c", or "w" was specified the date or time had to be specified also.

» TREE: This new command displays a graphical directory tree using either line-drawing or ASCII characters. It can also optionally display file names, dates, times, and sizes.

» UNALIAS: Added the /R switch to read a file of aliases to remove.

» UNSET: Added the /R switch to read a file of variables to remove.

What's New: Startup and Configuration

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

Added or modified the following *.INI* directives (all are new unless otherwise noted):

- CDDWinLeft, CDDWinTop, CDDWinWidth, CDDWinHeight: These directives set the position, size, and color of the popup window used for extended directory searches.

- CUA = YES | No: If set to "Yes" (the default), Take Command will use the standard Common User Access keys for cut, copy, and paste (Shift-Del, Ctrl-Ins, and Shift-Ins). If set to "No", Take Command will use the Windows non-CUA keys (Ctrl-X, Ctrl-C, and Ctrl-V).

- FileCompletion = cmd1:ext1 ext2;cmd2 ...: Sets up command-specific filename completion.

- HistMove = Yes | NO: If set to Yes, a recalled line from the command history is moved to the end of the history list, and removed from its original location.

- Include = filename: Includes the contents of the named file as if they had appeared at the location of the Include= directive in the current *.INI* file.

- LoadAssociations = YES | No: Determines whether the command processor will load Windows file associations at startup.

- TreePath = Path: Specifies the location of *JPSTREE.IDX* (the extended directory search database; defaults to C:\).

What's New: Variables and Functions

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

Added or updated the following internal variables (all variables listed are new unless otherwise noted):

2.01 » _CMDPROC: Returns the name of the current command processor.

» _CPU: Now returns "686" for Pentium Pro.

2.01 » _DOS: Now reports if Take Command is running under Windows 98.

» _DOWI: Returns the current day of week as an integer (Sun = 1, Mon = 2, etc.).

2.01 » _PIPE: Fixed a problem with this variable not always returning 1 when inside a pipe.

» _SELECTED: Returns the selected (highlighted) text.

» _XPIXELS: Returns the physical screen horizontal size in pixels.

» _YPIXELS: Returns the physical screen vertical size in pixels.

Added or updated the following variable functions (all functions listed are new unless otherwise noted):

» @CLIP[n]: Returns line **n** from the clipboard (base 0).

» @CONVERT[input,output,value]: Converts a number from one base to another.

» @DAY[date]: Returns the day for the specified date.

» @DOW[date]: Returns the day of week for the specified date, as a string (Sun, Mon, etc.)

» @DOWI[date]: Returns the day of week for the specified date, as an integer (Sun = 1, Mon = 2, etc.).

» @DOY[date]: Returns the day of year for the specified date (1366).

» @EVAL[expression]: Now supports user-definable decimal and thousands characters; see DecimalChar and and ThousandsChar, or , or SETDOS /G for details.

2.01 Fixed a bug with maximum-length argument strings.

» @EXEC[command]: This function has been modified; if you preface the command with an '@', @EXEC will return an empty string rather than the result code of the command.

» @EXECSTR[command]: Returns the first line written to STDOUT by the specified command. (This is intended to provide functionality similar to UNIX back-quoting.)

» @EXPAND[filename[,attributes]]: Expands a wildcard filename and returns all of the matching filenames / directory names on a single line.

» @FILEDATE / @FILETIME[filename[,acw]]: Added the optional second argument to determine which date / time field to return on LFN and NTFS drives.

» @FILESIZE[filename[,bkm[,a]]]: Added the optional third argument ***a***(llocated); if specified, the function returns the size actually used on disk, not the amount of data in the file.

» @FILEWRITE[n,text]: Can now write to standard output and standard error.

2.01 » @FILEWRITEB[n,length,string]: No longer truncates on a write if the file was opened in binary mode.

2.01 » @GETDIR[d:\path]: Fixed a problem with returning the root directory.

» @INSERT[n,string1,string2]: Inserts **string1** into **string2** starting at offset **n**.

» @LEFT[n,string]: Returns the leftmost **n** characters of **string**.

» @MONTH[date]: Return the month for the specified date.

» @NUMERIC[string]: Now considers a leading decimal separator as a numeric character, provided the remainder of the string is numeric and does not contain additional decimal characters.

» @REPLACE[string1,string2,text]: Replaces all occurrences of **string1** in the **text** with **string2**.

» @RIGHT[n,string]: Returns the rightmost **n** characters of **string**.

» @SEARCH[filename[,path]]: Now accepts an optional second argument for the path to search.

» @SELECT[filename,top,left,bottom,right,title[,1]]: Has a new optional argument following the title. If it's set to 1, @SELECT will sort the list alphabetically.

» @STRIP[chars,string]: Return **string** with the characters in **chars** removed.

» @WILD[string1,string2]: Does a wildcard comparison on the two strings and returns 1 if they match; 0 if they don't.

2.01 » @WORDS[["xxx"],string]: Fixed a problem if the line began with a -.

» @YEAR[date]: Return the year for the specified date.

What's New: Technical and Compatibility Enhancements

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

» If Take Command is the shell, it will now look for *STARTUP.GRP* first, then for *AUTOSTART.GRP* (which some international versions of Windows use instead).

» TYPE NUL now "works" (i.e. it generates no output), for compatibility with batch files which use TYPE NUL > file to generate a 0-byte file.

» Added debugging options which allow you to view the command "tail" passed to Take Command, and to "tag" error messages with the product name. See the Debug directive in *TCMD.INI* for additional details.

What's New: Bugs Fixed

[This subtopic covers some of the new features in this version of Take Command. For additional new features use the << and >> "browse buttons" at the top of the window, or see the main What's New topic.]

» DESCRIBE: Fixed a problem with quoted long filenames with paths.

» INPUT: Fixed a problem with /P not displaying *'s.

» REN / RENAME: Now works properly when renaming quoted long filenames with embedded wildcards.

» Worked around some operating system bugs which caused trouble with the Restart choice on the File Menu, and with the REBOOT command.

» The file find dialog now adds quotes to long filenames when doing a file-only search. This should allow you to double-click the name successfully.

» @FILESEEKL now always returns to the start of the file before seeking.

» Added some protection to description handling to avoid destroying descriptions for files with long names when updating the description file in a non-LFN environment.

» Improved support for large directories with large description (*DESCRIPT.ION*) files.

» Worked around most instances of a Windows bug which causes problems with displaying dialogs properly when Take Command is the Windows shell.

2.01 » Enabled the NormalPopupKey directive in the .INI file. Previously this directive was documented but was only available under its old name (NormalHWinKey).

2.01 » Fixed a problem which caused spurious "nesting level" errors when the Include directive was used more than three times in the .INI file.

2.01 » Fixed a problem with calling the FFIND dialog while inside LIST.

2.01 » Fixed a problem with expanding duplicate variables (i.e., "echo %foo% %foo%").

2.01 » Fixed a problem with printing marked blocks of text in LIST and the scrollback buffer.

2.01 » Modified popup windows to avoid the situation where the bottom half of the window is empty when the initially selected line is at the end of the list.

The Take Command Help System

This online help system for Take Command covers all Take Command features and internal commands. It includes reference information to assist you in using Take Command and developing batch files, and it includes most but not all of the details which are included in the printed Take Command manuals.

You can start the help system with the HELP command or the **F1** key. If you use the HELP command by itself you will see the help Table of Contents; if you follow the command with a topic name you will see help on that topic (if available).

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 Table of Contents for this help file, and you can pick the topic 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.

Take Command uses the Windows help system to display help text. Once you've started the help system with **F1** or the HELP command, you can use standard Windows keystrokes to navigate. For more information, click on the Help menu at the top of this window.

Finally, if you use a command incorrectly, omit a required parameter, or use an unrecognized option, Take Command will display a syntax summary of the command.

Error Messages

This section lists error messages generated by Take Command, 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. You may find some of these messages (for example, "Access denied") vague enough that they are not always helpful. Take Command 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.

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.

Already excluded files: You used more than one exclude range in a command. Combine the exclusions into a single range.

Application requires Windows NT: This program requires Windows NT features not available in Windows 3.x. Obtain a 16-bit version of the program, or run it under Windows NT.

Application requires 32-bit Windows: This program requires 32-bit features not available in Windows 3.x. Obtain a 16-bit version of the program, or run it under Windows NT or Windows 95.

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

Batch file missing: Take Command can't find the batch (*.BTM* or *.BAT*) 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 or MOVE file to itself: You cannot COPY or MOVE a file to itself. Take Command attempts to perform full path and filename expansion before copying to help ensure that files aren't inadvertently destroyed.

Can't create: Take Command 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: Take Command can't delete the specified file or directory. The disk is probably write protected.

Can't get directory: Take Command can't read the directory. The disk drive is probably not ready.

Can't load compressed EXE file: Windows cannot read the EXE file you tried to run. The file may be corrupted. Reinstall the file or restore it from a backup.

Can't load real-mode application: Windows cannot load the application. The file may be corrupted.

Reinstall the file or restore it from a backup.

Can't load second instance: Windows cannot load a second copy of the application. Use the first copy, or close it before opening the second.

Can't make directory entry: Take Command 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: Take Command 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 does not allow. Change to the parent directory and try again.

CAVEMAN.386 a.b.c installed Take Command requires x.y.z: You are running an older version of *CAVEMAN.386* with a newer version of Take Command. If you just upgraded Take Command, check the DEVICE statement in your *SYSTEM.INI* file to make sure it points to the correct directory for the new version. If you cannot resolve the problem contact JP Software or your dealer for assistance. See the Take Command/16 *Introduction and Installation Guide* for details on installing Caveman.

CD-ROM door open or **CD-ROM not ready**: The CD-ROM drive door is open, the power is off, or the drive is disconnected. Correct the problem and try again.

CD-ROM not High Sierra or ISO-9660: The CD-ROM is not recognized as a data CD (it may be a music CD). Put the correct CD in the drive and try again.

Clipboard is empty or not text format: You tried to retrieve some text from the Windows clipboard, but there is no text available. Correct the contents of the clipboard and try again.

Clipboard is in use by another program: Take Command could not access the Windows clipboard because another program was using it. Wait until the clipboard is available, or complete any pending action in the other program, then try again.

Command line too long: A single command exceeded 255 characters, or the entire command line exceeded 511 characters, during alias and variable expansion. Reduce the complexity of the command or use a batch file. Also check for an alias which refers back to itself either directly or indirectly.

Command only valid in batch file: You have tried to use a batch file command, like DO or GOSUB, from the command line or in an alias. A few commands can only be used in batch files (see the individual commands for details).

Contents lost before copy: COPY was appending files, and found one of the source files is the same as the destination. That source file is skipped, and appending continues with the next file.

Data error: Windows 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.

DDE [error message]: A DDE transaction could not be completed by the DDEEXEC command. The error message explains the reason. Consult the documentation for the DDE server you are using for additional details if necessary.

DLL is corrupt: A Dynamic Link Library associated with the program you tried to run is corrupt. Reinstall the *.DLL* file, or restore it from a backup.

Directory stack empty: POPD or DIRS can't find any entries in the directory stack.

Disk is write protected: The disk cannot be written to. Check the disk and remove the write-protect tab or close the write- protect window if necessary.

Drive not ready close door: The removable disk drive door is open. Close the door and try again.

Duplicate redirection: You tried to redirect standard input, standard output, or stand error more than once in the same command. Correct the command and try again.

Environment already saved: You have already saved the environment with a previous SETLOCAL command. You cannot nest SETLOCAL / ENDLOCAL pairs.

Error in command-line directive: You used the **//iniline** option to place an *.INI* directive on the startup command line, but the directive is in error. Usually a more specific error message follows, and can be looked up in this list.

Error on line [nnnn] of [filename]: There is an error in your *TCMD.INI* file. The following message explains the error in more detail. Correct the line in error and restart Take Command for your change to take effect.

Error reading: Windows 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 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 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 Take Command won't overwrite it.

File not found: Take Command couldn't find the specified file. Check the spelling and path name.

File type not found: The FTYPE command could not find the specified file type in the Windows 95 or Windows NT registry.

General failure: This is usually a hardware problem, particularly a disk drive failure or a device not properly connected to a serial or parallel port. Try to correct the problem, or reboot and try again. Also see **Data error** above.

Include file not found: You used the Include directive in the *TCMD.INI* file, but the file you specified was not found or could not be opened.

Include files nested too deep: You used the Include directive in the *TCMD.INI* file, and attempted to nest include files more than three levels deep.

Incorrect operating system: The application you tried to run requires a different operating system. Contact the manufacturer of the application for support.

Incorrect Windows version: The application you tried to run requires a different version of Windows than the one you are using. Contact the manufacturer of the application for support.

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 *TCMD.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 *TCMD.INI* file.

Invalid color: You gave an invalid value for a color directive in the *TCMD.INI* file.

Invalid count: The character repeat count for KEYSTACK is incorrect.

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

Invalid directive name: Take Command can't recognize the name of a directive in the *TCMD.INI* file.

Invalid drive: A bad or non-existent disk drive was specified.

Invalid EXE file: Windows cannot execute the program. The EXE file may be corrupted. Reinstall the file or restore it from a backup.

Invalid key name: You tried to make an invalid key substitution in the *TCMD.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 *TCMD.INI* file.

Invalid parameter: Take Command 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 *TCMD.INI* file.

Invalid time: An invalid time was entered. Check the syntax and reenter.

Keystroke substitution table full: Take Command ran out of room to store keystroke substitutions entered in the *TCMD.INI* file. Reduce the number of key substitutions or contact JP Software or your dealer for assistance.

Label not found: A GOTO or GOSUB referred to a non-existent label. Check your batch file.

Listbox is full: There is no more room in the Find Files / Text dialog's results box. Use a more selective search, or use the FFIND command rather than the dialog.

Missing ENDTEXT: A TEXT command is missing a matching ENDTEXT. Check the batch file.

Missing GOSUB: Take Command 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: Take Command 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.

Not an alias: The specified alias is not in the alias list.

Not an exexcutable file: Windows cannot run the file. Either you specified a file which is not executable, or the file has been corrupted. Correct the command, reinstall the file, or restore it from a backup.

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: Take Command or Windows 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 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.

Path not found: Windows cannot find the file you tried to run. Re-enter the command with the correct path.

Quit the active applications before exiting Windows: The EXIT command failed when exiting with Take Command running as the Windows shell, because one or more other applications would not close. Close the other application(s) manually and try again.

Read error: Windows encountered a disk read error; usually caused by a bad or unformatted disk. Also see **Data error** above.

Sector not found: Disk error, usually caused by a bad or unformatted disk. Also see **Data error** above.

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

Sharing violation: You tried to access a file in use by another program in a multitasking system or on a network. Wait for the file to become available, or change your method of operation so that another program does not have the file open while you are trying to use it.

Startup failed, contact JP Software: Take Command could not initialize and start operation. Contact JP Software or your dealer for assistance.

String area overflow: Take Command ran out of room to store the text from string directives in the *TCMD.INI* file. Reduce the complexity of the *TCMD.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 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_CMD loop: The UNKNOWN_CMD alias called itself more than ten times. The alias probably contains an unknown command itself, and is stuck in an infinite loop. Correct the alias.

Unknown command: A command was entered that Take Command didn't recognize and couldn't find in the current search path. Check the spelling or PATH specification. You can handle unknown commands with the UNKNOWN_CMD alias (see ALIAS).

Unknown EXE type: Windows cannot execute the program. The EXE file may be corrupted. Reinstall the file or restore it from a backup.

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 ACTIVATE command could not find a window with the specified title. Correct the command or open the appropriate window and try again.

Write error: Windows encountered a disk write error; usually caused by a bad or unformatted disk. Also see **Data error** above.

Troubleshooting, Service, and Support

If you need help with Take Command, we encourage you to review our documentation and then contact us for assistance if required.

If you need help with **sales**, **ordering**, or **shipments** (including defective disks or other materials which were shipped to you), or with **brand codes**, please contact our **Sales and Customer Service** department. See Contacting JP Software for our email address, mail address, and telephone numbers. (The sales and customer service staff cannot assist you with technical problems. However, if you have multiple questions or are unsure of the nature of the problem, feel free to contact us for customer service; the staff will have a support technician contact you if your question turns out to require technical expertise.)

If you need **technical support** for Take Command, review the Technical Support information section, which tells you what we need to know to provide you with accurate and timely support. Then contact us via one of the methods described there. (The technical support staff cannot assist you with sales, ordering, replacement brand cards, or other administrative matters.)

Technical Support

Before You Contact Us

Before contacting us for support, please check this help file, the Reference Manual and other documentation for answers to your question. If you can't find what you need, try the Index. If you're having trouble getting Take Command to run properly, review the information on Error Messages, and look through the *README.DOC* file for any last-minute information.

If you do need to contact us for support, we can do a much better job of assisting you if you can give us some basic information, separate from your interpretations of or conclusions about the problem. The first four items listed below are essential for us to be able to understand and assist you with your problem:

- **What environment are you working in?** This includes the operating system version are you using, the version of the JP Software product involved, and related information such as network connections and the name and version number of any other software which appears to be involved in the problem. Use the VER /R command to determine the Take Command version and operating system version.
- **What exactly did you do?** A concise description of what steps you must take to make the problem appear is much more useful than a long analysis of what might be happening.
- **What did you expect to happen?** Tell us the result you expected from the command or operation in question, so that we understand what you are trying to do.
- **What actually happened?** At what point did the failure occur? If you saw an error message or other important or unusual information on the screen, what **exactly** did it say?
- **Briefly, what techniques did you use to try to resolve the problem?** What results did you get?
- **If the problem seems related to startup and configuration issues,** what are the contents of any startup files you use (such as *CONFIG.SYS*, *AUTOEXEC.BAT*, *TCSTART*, *TCEXIT*, and the *TCMD.INI* file), any batch files they call, and any alias or environment variable files they load?
- **Can you repeat the problem or does it occur randomly?** If it's random, does it seem related to the programs you're using when the problem occurs?

Electronic Support

Usually the best way to contact us for support is via CompuServe or the Internet. The most efficient method is to use our CompuServe support conference; if you do not have CompuServe access, contact us via Internet email. See Contacting JP Software for our addresses.

Whenever possible, we also read messages posted on the Usenet *comp.os.msdos.4dos* newsgroup, and in 4DOS conferences on the RIME, Ilink, and FidoNet BBS networks (these conferences are named for 4DOS, but carry messages related to all JP Software products). These areas offer valuable information and discussions with other users, but are not managed by JP Software, and are not official support channels. To be certain of a direct answer from our support staff use our CompuServe forum or Internet email, or contact us by telephone, fax, or mail.

A number of support resources are available from our web site listed above, including error message listings, documentation files, product histories, technical tips and discussions, other technical information, and links to other companies' sites. We update this information regularly, and we encourage you to check the Technical Support area of the web site to see if the information there will address any questions you have.

Technical support messages should be sent as standard ASCII text. Please do not transmit attached files, binary files, screen images, or any file over 10K bytes in size to any of our electronic technical support addresses unless asked to do so by our support staff.

Telephone Support

Technical support by telephone within the US and Canada is handled on a callback basis. To contact our support staff, call the US / Canada Support Line at any time and leave a short voice mail message describing your technical problem (this line can **not** be used for sales / customer service issues such as pricing, ordering, upgrades, or shipping problems). We check these messages regularly throughout the day and will return your call as quickly as possible. See Contacting JP Software for our phone numbers.

We generally return all technical support calls within 24 hours (weekends and holidays excluded), and most are returned much more quickly, usually on the same business day. If your problem is urgent and requires a faster response, please let us know and we will try to accommodate you. If you contact us by telephone and don't receive a reply within 24 hours, please try again. We probably tried to return your call and were unable to reach you.

If you are calling from outside the US and Canada, are not sure if your question requires technical support, need other assistance in addition to your technical questions, or find yourself playing "telephone tag" with our support staff, please call our main number listed above. Our office staff will assist you with all of your concerns, and have a technical support representative call you back if necessary.

If you have a problem with a batch file or complex alias, please contact us electronically if possible. Include a copy of the batch file or alias in question, preferably as part of the text of your message (not as an attachment). If you do not have electronic access, contact us by fax if possible. Problems of this type are usually very difficult to diagnose over the telephone because we cannot see the material you are working with. For longer batch files (over about 25 lines), do your best to reproduce the problem in a smaller test file.

If you need more in-depth assistance with the development of complex batch files or other procedures, please contact us for information on consulting services.

Contacting JP Software

You can contact JP Software at the following addresses and numbers. Our normal business hours are 8:30 AM to 5:00 PM weekdays, eastern US time (except holidays).

Address:	JP Software Inc. P.O. Box 1470 East Arlington, MA 02174 USA
Main number:	(781) 646-3975
Fax:	(781) 646-0904
Order Line:	(781) 368-8777 (US / Canada, orders only)
Support Line:	(781) 646-0798 (US / Canada only; **see Telephone Support before using this number**)
Internet:	Sales / Customer Service: **sales@jpsoft.com** Technical Support*: **support@jpsoft.com** World Wide Web: **http://www.jpsoft.com/** File downloads via FTP: For the simplest access to JP Software files use our web site. For direct FTP access connect to **ftp.std.com** and look in the **/vendors/jpsoft** directory.
CompuServe:	Sales / Customer Service: **75020,244** Technical Support and File Downloads*: **GO JPSOFT** or **GO PCVENB**, section / library 10, User ID 75300,1215.
BBS Downloads:	Channel 1 BBS, Boston, 617-349-1300 at 2,400 28,800 baud, no parity, 8 data bits, 1 stop bit.

** Technical support messages should be sent as standard ASCII text. Please do not transmit attached files, binary files, screen images, or any file over 10K bytes in size to any of our electronic technical support addresses unless asked to do so by our support staff.*

File Systems and File Name Conventions

You may have dozens, hundreds, or thousands of files stored on your computer's disks. Your operating system is responsible for managing all of these files. In order to do so, it uses a unique name to locate each file in much the same way that the post office assigns a unique address to every residence.

The unique name of any file is composed of a **drive letter**, a directory **path**, and a **filename**. Each of these parts of the file's name is case insensitive; you can mix upper and lower case letters in any way you wish.

The topics below are roughly divided according to the different parts of a file name, and cover the file system structure and naming conventions:

Drives and Volumes

A **drive letter** designates which drive contains the file. In a file's full name, the drive letter is followed by a colon. Drive letters **A:** and **B:** are normally reserved for the floppy disk drives.

Normally, drive **C:** is the first (or only) hard disk drive. Most current operating systems can divide a large hard disk into multiple logical drives or **volumes** that are usually called **C:**, **D:**, **E:**, etc. Network systems (LANs) give additional drive letters to sections of the network file server drives.

Most recent systems also include a CD-ROM drive. The CD-ROM is also assigned a drive letter (or several letters, for CD-ROM changers), typically using letters beyond that used by the last hard disk in the system, but before any network drives.Some systems may have "RAM disks" (sometimes called "virtual disks"), which are areas of memory set aside by software (a "RAM disk driver") for use as fast but temporary storage. Like CD-ROM drives, RAM disks are usually assigned drive letters beyond the last hard disk in the system, but before network drives.

For example, on a system with a large hard disk you might have **A:** and **B:** as floppy drives, **C:**, **D:**, and **E:** as parts of the hard disk, **F:** as a CD-ROM drive, **G:** as a RAM disk, and **H:** and **I:** as network drives.

Additional information about disk files and directories is available under File Systems, Directories and Subdirectories, File Names, and File Attributes and Time Stamps.

File Systems

Each disk volume is organized according to a **file system**. The file system determines how files are named and how they are organized on the disk.

DOS and Windows use the **FAT File System**. Its name comes from the **F**ile **A**llocation **T**able DOS uses to keep track of the space allocated to each file. Take Command supports the FAT file system as well.

A **network file system** allows you to access files stored on another computer on a network, rather than on your own system. Take Command supports all network file systems which are compatible with DOS and Windows.

File and directory names for network file systems depend on both the "server" software running on the system that has the files on it, and the "client" software running on your computer to connect it to the network. However, they usually follow the rules described under File Names.

Most network software "maps" unused drive letters on your system to specific locations on the network, and you can then treat the drive as if it were physically part of your local computer.

Some networks also support the **Universal Naming Convention**, which provides a common method for accessing files on a network drive without using a "mapped" drive letter. Names specified this way are called UNC names. They typically appear as **\\server\volume\path\filename**, where **server** is the name of the network server where the files reside, **volume** is the name of a disk volume on that server, and the **path\filename** portion is a directory name and file name which follow the conventions described under **Directories** below. Take Command supports UNC filenames, and also allows you to use UNC directory names when changing directories (see Directory Navigation for more details).

In rare cases, Take Command may not be able to report correct statistics on network drives (such as the number of bytes free on a drive). This is usually because the network file system does not provide complete or accurate information.

Additional information about disk files and directories is available under Drives and Volumes, Directories and Subdirectories, File Names, and File Attributes and Time Stamps.

Directories and Subdirectories

A file system is a method of organizing all of the files on an entire disk or hard disk volume. **Directories** are used to divide the files on a disk into logical groups that are easy to work with. Their purpose is similar to the use of file drawers to contain groups of hanging folders, hanging folders to contain smaller manila folders, and so on. **Directories** are also sometimes referred to as **folders**.

Every drive has a **root** or base directory, and many have one or more **subdirectories**. Subdirectories can also have subdirectories, extending in a branching **tree** structure from the root directory. The collection of all directories on a drive is often called the **directory tree**, and a portion of the tree is sometimes called a **subtree**. The terms **directory** and **subdirectory** are typically used interchangeably to mean a single subdirectory within this tree structure.

Subdirectory names follow the same rules as file names (see File Names). However, it is usually best to use a name without an extension, as some application programs do not properly handle subdirectory names that have an extension.

The drive and subdirectory portion of a file's name are collectively called the file's **path**. For example, the file name *C:\DIR1\DIR2\MYFILE.DAT* says to look for the file *MYFILE.DAT* in the subdirectory *DIR2* which is part of the subdirectory *DIR1* which is on drive C. The path for *MYFILE.DAT* is *C:\DIR1\DIR2*. The backslashes between subdirectory names are required. The total length of a file's path may not exceed 64 characters (this limit excludes the file name and extension, but includes the drive letter and colon).

The operating system and command processor remember both a **current or default drive** for your system as a whole, and a **current or default directory** for every drive in your system. Whenever a program tries to create or access a file without specifying the file's path, the operating system uses the current drive (if no other drive is specified) and the current directory (if no other directory path is specified).

The root directory is named using the drive letter and a single backslash. For example, *D:* refers to the root directory of drive *D:*. Using a drive letter with no directory name at all refers to the current directory on the specified drive. For example, *E:TCMD.DOC* refers to the file *TCMD.DOC* in the current directory on drive *E:*, whereas *E:\TCMD.DOC* refers to the file *TCMD.DOC* in the root directory on drive *E:*.

There are also two special subdirectory names that are useful in many situations: a single period by itself [**.**] means "the current default directory." Two periods together [**..**] means "the directory which contains the current default directory" (often referred to as the **parent directory**). These special names can be used wherever a full directory name can be used. Take Command allows you to use additional periods to specify directories further "up" the tree (see Extended Parent Directory Names).

Additional information about disk files and file systems is available under Drives and Volumes, File Systems, File Names, and File Attributes and Time Stamps.

File Names

Under the FAT file system, the filename consists of a **base name** of 1 to 8 characters plus an optional **extension** composed of a period plus 1 to 3 more characters. Traditional FAT filenames with an 8-character name and a 3-character extension are sometimes referred to as short filenames (SFNs) to distinguish them from the long filenames (LFNs) supported under Windows 95 and Windows NT.

You can use alphabetic and numeric characters plus the punctuation marks **! # $ % & ' ( ) - @ ^ _ ` { }** and **~** in both the base name and the extension of a FAT filename. Because the exclamation point [**!**], percent sign [**%**], caret [**^**], at sign [**@**], parentheses [**()**], and back-quote [`] also have other meanings to Take Command, it is best to avoid using them in filenames.

You may occasionally encounter filenames which are not displayed the way you expect if you have used characters from outside the U.S. English character set in the name. These are generally due to problems in the way Windows translates characters between the OEM and ANSI character sets. Correcting the problem may require experimentation with fonts and character sets, and occasionally some such problems may not be readily correctable within Take Command. For more information on underlying issues related to fonts and character sets see ASCII, Key Codes, and ANSI Codes.

Additional information about disk files and file systems is available under Drives and Volumes, File Systems, Directories and Subdirectories, and File Attributes and Time Stamps.

File Attributes and Time Stamps

Each file also has **attributes**, and one or more **time stamps**. Attributes define characteristics of the file which may be useful to the operating system, to you, or to an application program. Time stamps can record when the file was created, last modified, or last accessed. Most Take Command file processing commands allow you to select files for processing based on their attributes and/or time stamp(s).

Each file on your system has four standard attributes. Every time a program modifies a file, the operating system sets the **Archive** attribute, which signals that the file has been modified since it was last backed up. This attribute can be used by Take Command to determine which files to COPY, and by backup programs to determine which files to back up. When the **Read-only** attribute is set, the file can't be changed or erased accidentally; this can be used to help protect important files from damage. The **Hidden** and **System** attributes prevent the file from appearing in normal directory listings. (Two additional attributes, **Directory** and **Volume label**, are also available. These attributes are controlled by the operating system, and are not modified directly by Take Command.)

Attributes can be set and viewed with the ATTRIB command. The DIR command also has options to select filenames to view based on their attributes, to view the attributes themselves, and to view information about normally "invisible" hidden and system files.

When a file is created, and every time it is modified, the operating system records the system time and date in a **time stamp** in the file's directory entry. Several Take Command commands and variable functions, and many backup and utility programs, use this time stamp to determine the relative ages of files.

Additional information about disk files and file systems is available under Drives and Volumes, File Systems, Directories and Subdirectories, and File Names.

Miscellaneous Reference Information

Colors and Color Names

You can use color names in several of the directives in the *TCMD.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 directives like ColorDIR, where using color names may take too much space. Take Command recognizes these color numbers:

```
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.

The blinking text and border colors that are available with 4DOS can not be used with Take Command. This restriction is due to limitations inherent in Windows and in graphical displays.

Keys and Key Names

Key names are used to define keystroke aliases, in several *TCMD.INI* directives, and with the KEYSTACK command. The format of a key name is the same in all three 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-F10     This is okay
Alt F10     The space will cause an error
```

If you prefer, you can use a numeric value instead of a key name. Use the ASCII code for an ASCII, extended ASCII, or control character. Use the scan code preceded by an at sign [**@**] for extended key codes like **F1** or the cursor keys. For example, use 13 for **Enter**, or @59 for **F1**. In general, you will find it easier to use the names described above rather than key numbers.

Some keys are intercepted by Windows and are not passed on to Take Command. For example, Alt-Esc and Ctrl-Esc typically pop up a tasklist or are used in switching among multiple tasks. Keys which are intercepted by the operating system (including menu accelerators, *i.e.* **Alt** plus another key) generally cannot be assigned to aliases or with key mapping directives, because Take Command never receives these keystrokes and therefore cannot act on them.

Popup Windows

Several features of Take Command display popup windows. A popup window may be used to display filenames, recently-executed commands, recently-used directories, the results of an extended directory search, or a list created by the SELECT command or the @SELECT internal function.

Popup windows always display a list of choices and a cursor bar. You can move the cursor bar inside the window until you find the choice that you wish to make, then press the **Enter** key to select that item.

Navigation inside any popup window follows the conventions described below. Additional information on each specific type of popup window is provided when that window is introduced, later in the manual.

You can control the position and size of most popup windows with the configuration dialogs, or with the PopupWinLeft, PopupWinTop, PopupWinWidth, and PopupWinHeight directives in the *.INI* file. A few popup windows (*e.g.*, the extended directory search window) have their own specific *.INI* directives, and corresponding separate choices in the configuration dialogs. You can also change the keys used in most popup windows with key mapping directives in the *.INI* file.

Once a window is open, you can use these navigation keys to find the selection you wish to make:

Up Arrow	Move the selection bar up one line.
Down Arrow	Move the selection bar one line.
Left Arrow	Scroll the display left 1 column, if it is a scrolling display (*i.e.* if it has a horizontal scrollbar).
Right Arrow	Scroll the display right 1 column, if it is a scrolling display (*i.e.* if it has a horizontal scrollbar).
PgUp	Scroll the display up one page.
PgDn	Scroll the display down one page.
Ctrl-PgUp or **Home**	Go to the beginning of the list.
Ctrl-PgDn or **End**	Go to the end of the list.
Esc	Close the window without making a selection.
Enter	Select the current item and close the window.

In addition to scrolling through a popup window, you can search the list using character matching. If you press a character, the cursor bar will move to the next entry that begins with that character. If you type multiple characters, the cursor will move to the entry that begins with the search string entered to that point (you can enter a search string up to 32 characters long). If no entry matches the character or string that you have typed, Take Command beeps and does not move the cursor bar. To reset the search string, press Backspace.

Executable Files and File Searches

When Take Command knows that it is supposed to run an external command, it tries to find an executable file whose name matches the command name. (Executable files are typically those with a *.COM* or *.EXE* extension, or with a *.PIF* extension under Windows.) It runs the executable file if it finds one.

If Take Command cannot find an executable program to run, it next looks for a batch file (a file with one or more commands in it) whose name matches the command name. Take Command looks first for a *.BTM* file, then for a *.BAT* file, and finally for a *.REX* file See *.BAT, .CMD*, and *.BTM* Files for more information on these different types of batch files. If it finds such a file, it then reads each line in the file as a new command.

If the search for a batch file fails, Take Command checks to see if the command name matches the name of a file with an extension that is associated with a specific application (for example, if you have associated *.DOC* with your editor or word processor, and type the name of a *.DOC* file). If a match is found, Take Command runs the program you specified when the association was defined.

In searching for the application associated with a file, Take Command will first examine any executable extensions you have defined to associate a file extension with a specific program to process that type of file. Take Command then checks file associations defined in *WIN.INI* and / or the Windows registry; see Windows File Associations for complete details.

Take Command first searches for an executable program, a batch file, and a file with an executable extension or Windows file association in the current directory. If that search fails, it repeats its search in every directory in your **search path**.

The search path is a list of directories that Take Command (and some applications) search for executable files. For example, if you wanted Take Command to search the root directory of the C: drive, the \ WINUTIL subdirectory on the C: drive, and the \UTIL directory on the D: drive for executable files, your search path would look like this:

```
PATH=C:\;C:\WINUTIL;D:\UTIL
```

Notice that the directory names in the search path are separated by semicolons.

You can create or view the search path with the PATH command. You can use the ESET command to edit the path. Many programs also use the search path to find their own files. The search path is stored in the environment with the name PATH.

Take Command also searches the \WINDOWS and \WINDOWS\SYSTEM directories, in that order, after the current directory and before any directories listed in your search path. This part of the search procedure conforms with the traditional search sequences used by Windows.

Remember, Take Command always looks for an executable file or a file with an executable extension or Windows file association in the current subdirectory, then in the Windows directories if appropriate (see above), and then in each directory in the search path. (You can change the search order so the current directory is not searched first; see the PATH command for details.)

If you include an extension as part of the command name, Take Command only searches for a file with that extension. Similarly, if you include a path as part of the command name, the command processor will look only in the directory you specified, and ignore the usual search of the current directory and the PATH.

The following table sums up the possible search options (the term "standard search" refers to the search of the current directory, the Windows directories in Take Command/16 and Take Command/32, and each

directory in the search path):

Command	Take Command Search Sequence
WP	Standard search for any executable file whose base name is *WP*.
WP.EXE	Standard search for *WP.EXE*; will not find files with other extensions.
C:\WP7\WP	Looks in the *C:\WP7* directory for any executable file whose base name is *WP*. Does not check the standard search directories.
C:\WP7\WP.EXE	Looks only for the file *C:\WP7\WP.EXE*.
LAB.DOC	Standard search for *LAB.DOC*, if *.DOC* is defined as a Take Command executable extension or Windows file association. Runs the associated application if the file is found.
C:\LI\LAB.DOC	Looks only for the file *C:\LI\LAB.DOC*, and only if *.DOC* is defined as a Take Command executable extension or Windows file association. Runs the associated application if the file is found.

If Take Command cannot find an executable file, batch program, or a file with an executable extension or Windows file association in the current directory or any directory in the search path, it looks for an alias called UNKNOWN_CMD (see the ALIAS command for details). If you have defined an alias with that name, it is executed (this allows you to control error handling for unknown commands). Otherwise, Take Command displays an "Unknown command" error message and waits for your next instruction.

ASCII, Key Codes, and ANSI Codes

For ASCII, key code, and ANSI code reference tables, see:

ASCII Table
Key Codes and Scan Codes Table
ANSI Codes Reference

The remainder of this section gives a detailed explanation of character sets, ASCII, and key codes (for more information on Take Commands ANSI string support see the separate ANSI Support topic). If you are troubleshooting a keyboard or character display problem be sure to read all of the explanation below before referring to the tables.

The translation of a key you type on the keyboard to a displayed character on the screen depends on several related aspects of character handling. A complete discussion of these topics is well beyond the scope of this document. However, a basic picture of the steps in the keystroke and character translation process will help you understand how characters are processed in your system, and why they occasionally may not come out the way you expect.

Internally, computers use numbers to represent the keys you press and the characters displayed on the screen. To display the text that you type, your computer and operating system require five pieces of information:

- The numeric **key code** for the physical key you pressed;
- The specific character that key code represents based on your current keyboard layout or **country setting**;
- The **character set** currently in use on your system (see below);
- The international **code page** in use for that character set; and
- The **display font** used to display the character.

The numeric **key code** is determined by your physical hardware including the language that your keyboard is produced for. The **character set** is usually determined by the operating system. These items typically are not under your control. However, most systems do allow you to control the keyboard **country setting,** the **code page, and** the **display font**.

For an explanation of how key codes work, see the Key Codes and Scan Codes Explanation. For a list of key codes and scan codes for keys on the standard U.S. keyboard see the Key Codes and Scan Codes Table.

If the key codes produced by your keyboard, the code page, and the font you choose are not fully compatible with each other, the characters displayed on the screen will not match what you type. The differences are likely to appear in line-drawing characters, "international" (non-English) characters, and special symbols, not in commonly-used U.S. English alphabetic, numeric, or punctuation characters.

Most systems use a "single-byte" character set for keyboard and screen display. These sets define 256 characters or symbols, with a numeric representation for each. ("Double-byte" character sets, with up to 65,536 characters each, are used for languages with more than 256 symbols, and for some multi-lingual systems.) Most PC single-byte character sets are based on a code called ASCII, the **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. For a complete list of ASCII codes see the ASCII Table.

The original ASCII code was defined over 30 years ago for use in mainframe and minicomputer systems, and has 128 character values. These include the upper and lower case letters, numerals, and

punctuation marks used in U.S. English, plus non-printing control codes (which can be entered on most keyboards by pressing the **Ctrl** key plus another character, or by pressing certain special keys like **Tab**, **Enter**, **Backspace**, and **Esc**). However, ASCII is not a complete character set, because it defines only 128 of the required 256 symbols.

IBM, in its original PC, created a complete 256-character set (called the Original Equipment Manufacturer or "OEM" character set) by defining an additional 128 **extended ASCII** codes for math symbols, "international" characters, the characters used to draw boxes and lines, and some miscellaneous symbols.

Some operating systems support other character sets; in particular, Windows uses the ANSI character set internally to store and display text, even though other parts of the system (*e.g.* the file system which stores file names on disk) use IBM's OEM character set. The ANSI character set is identical to the OEM character set for U.S. English printed characters, but may vary for "international" characters not used in U.S. English. In most cases, Windows automatically translates characters from one set to another as needed, but problems can sometimes result in errors in displayed text (*e.g.*, differences between the appearance of accented characters in filenames in Windows and DOS applications).

See your operating system documentation for more information about character sets, code pages, and country and language support. Refer to your operating system and/or font documentation for details on the full character set available in any particular font.

The tables in this 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.

ASCII Tables

These tables shows the 256-character ASCII character set for U.S. English systems. Most of the first 128 characters will be the same on non-U.S. systems. Characters 1 through 31 and 128 through 255 are likely to vary in appearance between fonts, and may look different if your system is not configured for U.S. English.

For more details on ASCII, character sets, and key codes, see the general information topic on ASCII, Key Codes, and ANSI Codes.

The tables below are primarily in the MS Line Draw font, as this font gives the best representation of standard ASCII characters. The Control Characters portion of the table also uses the Symbol, and WingDings fonts to display the best possible approximation of the standard screen characters shown in common ASCII charts. If you do not have all of these fonts on your system the characters may not appear correctly.

Certain characters are shown as blank in the tables below: Characters 000 and 255 are blank because they are normally shown that way in ASCII charts; characters 002, 008, 010 - 014, and 023 are blank because the standard symbols for these characters have no representation in typical Windows fonts, so they cannot be shown here.

Control Characters (0 - 31)

Dec	Hex	Char	Name	Ctrl	Dec	Hex	Char	Name	Ctrl
000	00		NUL	^@	016	10	Ø	DLE	^P
001	01	J	SOH	^A	017	11	×	DC1	^Q
002	02		STX	^B	018	12	ô	DC2	^R
003	03		ETX	^C	019	13	!	DC3	^S
004	04	◆	EOT	^D	020	14	¶	DC4	^T
005	05	♣	ENQ	^E	021	15	§	NAK	^U
006	06	♠	ACK	^F	022	16	n	SYN	^V
007	07		BEL	^G	023	17		ETB	^W
008	08		BS	^H	024	18		CAN	^X
009	09	m	HT	^I	025	19	↓	EM	^Y
010	0A		LF	^J	026	1A	→	SUB	^Z
011	0B		VT	^K	027	1B	←	ESC	^[
012	0C		FF	^L	028	1C	_	FS	^\
013	0D		CR	^M	029	1D	↔	GS	^]
014	0E		SO	^N	030	1E	Ù	RS	^^
015	0F	R	SI	^O	031	1F	Ú	US	^_

Standard ASCII Printing Characters (32 - 127)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
032	20	Space	064	40	@	096	60	`
033	21	!	065	41	A	097	61	a
034	22	"	066	42	B	098	62	b
035	23	#	067	43	C	099	63	c
036	24	$	068	44	D	100	64	d
037	25	%	069	45	E	101	65	e
038	26	&	070	46	F	102	66	f

039	27	'	071	47	G	103	67	g
040	28	(	072	48	H	104	68	h
041	29	)	073	49	I	105	69	i
042	2A	*	074	4A	J	106	6A	j
043	2B	+	075	4B	K	107	6B	k
044	2C	,	076	4C	L	108	6C	l
045	2D	-	077	4D	M	109	6D	m
046	2E	.	078	4E	N	110	6E	n
047	2F	/	079	4F	O	111	6F	o
048	30	0	080	50	P	112	70	p
049	31	1	081	51	Q	113	71	q
050	32	2	082	52	R	114	72	r
051	33	3	083	53	S	115	73	s
052	34	4	084	54	T	116	74	t
053	35	5	085	55	U	117	75	u
054	36	6	086	56	V	118	76	v
055	37	7	087	57	W	119	77	w
056	38	8	088	58	X	120	78	x
057	39	9	089	59	Y	121	79	y
058	3A	:	090	5A	Z	122	7A	z
059	3B	;	091	5B	[	123	7B	{
060	3C	<	092	5C	\	124	7C	|
061	3D	=	093	5D	]	125	7D	}
062	3E	>	094	5E	^	126	7E	~
063	3F	?	095	5F	_	127	7F	

Extended ASCII Characters (128 - 255)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	€	171	AB	«	214	D6	Ö
129	81	□	172	AC	¬	215	D7	×
130	82	‚	173	AD		216	D8	Ø
131	83	ƒ	174	AE	®	217	D9	Ù
132	84	„	175	AF	¯	218	DA	Ú
133	85	…	176	B0	°	219	DB	Û
134	86	†	177	B1	±	220	DC	Ü
135	87	‡	178	B2	²	221	DD	Ý
136	88	ˆ	179	B3	³	222	DE	Þ
137	89	‰	180	B4	´	223	DF	ß
138	8A	Š	181	B5	µ	224	E0	à
139	8B	‹	182	B6	¶	225	E1	á
140	8C	Œ	183	B7	·	226	E2	â
141	8D	□	184	B8	¸	227	E3	ã
142	8E	Ž	185	B9	¹	228	E4	ä
143	8F	□	186	BA	º	229	E5	å
144	90	□	187	BB	»	230	E6	æ
145	91	'	188	BC	¼	231	E7	ç
146	92	'	189	BD	½	232	E8	è
147	93	"	190	BE	¾	233	E9	é
148	94	"	191	BF	¿	234	EA	ê
149	95		192	C0	À	235	EB	ë
150	96		193	C1	Á	236	EC	ì
151	97		194	C2	Â	237	ED	í
152	98	˜	195	C3	Ã	238	EE	î

153	99	™	196	C4	Ä	239	EF	ï
154	9A	š	197	C5	Å	240	F0	ð
155	9B	›	198	C6	Æ	241	F1	ñ
156	9C	œ	199	C7	Ç	242	F2	ò
157	9D	□	200	C8	È	243	F3	ó
158	9E	ž	201	C9	É	244	F4	ô
159	9F	Ÿ	202	CA	Ê	245	F5	õ
160	A0		203	CB	Ë	246	F6	ö
161	A1	¡	204	CC	Ì	247	F7	÷
162	A2	¢	205	CD	Í	248	F8	ø
163	A3	£	206	CE	Î	249	F9	ù
164	A4	¤	207	CF	Ï	250	FA	ú
165	A5	¥	208	D0	Ð	251	FB	û
166	A6	¦	209	D1	Ñ	252	FC	ü
167	A7	§	210	D2	Ò	253	FD	ý
168	A8	¨	211	D3	Ó	254	FE	þ
169	A9	©	212	D4	Ô	255	FF	
170	AA	ª	213	D5	Õ			

Key Codes and Scan Codes Table

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

The table below shows standard codes used on a U.S. keyboard. Other keyboards will be largely, but not entirely, the same, due to the changes required to support characters outside the U.S. English character set.

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. For more details on key codes, scan codes, and ASCII see the general information on ASCII, Key Codes, and ANSI Codes.)

When you press a single key or a key combination, Windows 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 returns these numbers the next time a program requests keyboard input. This section explains how key codes work; for information on using them with Take Command see the key mapping directives in the *TCMD.INI* file, keystroke aliases, and INKEY.

Most Take Command 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 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.

ANSI Codes Reference

Take Commands ANSI support allows you to manipulate the cursor, screen color, and other display attributes through sequences of special characters embedded in the text you display on the screen. These sequences are called "ANSI commands". (For a general description of this feature, see ANSI Support.)

Take Command supports most common ANSI screen commands, but does not provide the complete set of options supported by many operating system or third-party replacement ANSI drivers (for example, Take Command does not support ANSI key substitutions). This section is a quick-reference to the ANSI commands supported by Take Command.

ANSI support within Take Command can be enabled or disabled with the ANSI directive in *TCMD.INI*, the corresponding option on the Display page of the configuration dialogs, or the SETDOS /A command. You can test whether ANSI support is enabled with the _ANSI internal variable.

An ANSI command string consists of three parts:

<ESC>[	The ASCII character ESC, followed by a left bracket. These two characters must be present in all ANSI strings.
parameters	Optional parameters for the command, usually numeric. If there are multiple parameters they are separated by semicolons.
command	A single-letter command. The case of the letter **is** meaningful.

For example, to position the cursor to row 7, column 12 the ANSI command is:

ESC[7;12H

The **parameters** part of this command is 7;12 and the **command** part is H.

To transmit ANSI commands to the screen with Take Command you can use the ECHO command. The ESC character can be generated by inserting it into the string directly (if you are putting the string in a batch file and your editor will insert such a character), or by using Take command's internal "escape" character (Ctrl-X or ASCII 24, in some fonts as an up-arrow []) followed by a lower-case "e".

For example, the sequence shown above could be transmitted from a batch file with either of these commands (the first uses an ESC character directly, represented below by **<ESC>**; the second uses <Ctrl-X>e):

```
echo <ESC>[7;12H
echo e[7;12H
```

You can also include ANSI commands in your prompt, using $e to transmit the ESC character.

Commands

Take Commands ANSI support can accept the following commands (parameters are shown below in lower-case, *e.g.* row and attr, and must be replaced with the appropriate numeric value when using the commands):

ESC[rowsA	Cursor up
ESC[rowsB	Cursor down
ESC[colsC	Cursor right

ESC[colsD	Cursor left
ESC[row;colH	Set cursor position (top left is row 1, column 1)
ESC[2J	Clear screen
ESC[K	Clear from cursor to end of line
ESC[row;colf	Set cursor position, same as "H" command
ESC[attr;attr;...m	Set display attributes; see table of attribute values below
ESC[s	Save cursor position (may not be nested)
ESC[u	Restore cursor position after a save

Display Attributes

The attribute values used for the m command are:

1 High intensity (bright) foreground color

2 Normal intensity

30-37 Set the foreground color:

30=Black
31=Red
32=Green
33=Yellow
34=Blue
35=Magenta
36=Cyan
37=White

40-47 Set the background color, same values as above but substitute 40 for 30 etc.

Settings are cumulative, so (for example) to set bright red foreground you must first set red, then set high intensity:

```
echo e[31;1m
```

Examples

Set the display to bright cyan on blue, and clear the screen:

```
echo e[44;36;1me[2J
```

Set up a prompt which saves the cursor position, displays the date and time on the top line in bright white on magenta, and then restores the cursor position and sets the color to bright cyan on blue, and displays the standard prompt:

```
prompt $e[s$e[1;1f$e[45;37;1m$e[K$d $t$e[u$e[44;36;1m$p$g
```

Glossary

The glossary contains over 200 terms, and is divided into sections by the first letter of each term. Select the section you want to review:

A B C D E F G H I K L M N O P R S T U V W X

Glossary - A

A B C D E F G H I K L M N O P R S T U V W X

Alias Parameter: A numbered variable (*e.g.* %2) included in an alias definition, allowing a different value to be used in the alias each time it is executed.

Alias: A shorthand name for a command or series of commands.

Alternate File Name: See **LFN File System**, and also **SFN**.

AND: A logical combination of two true or false conditions. If both conditions are true, the result is true; if either condition is false, the result is false.

ANSI: Usually a reference to ANSI control sequences, standardized sequences of text characters which control colors on the screen, manipulate the cursor, and redefine keys. Take Command includes support for ANSI screen and cusrsor control sequences. The abbreviation ANSI is for American National Standards Institute, an organization whch sets standards for computer-related systems, including "ANSI" screen control sequences.

APM: A standardized system used by manufacturers of battery-powered computers to control system power management, including shutdown of unused components or of the entire system based on usage patterns. APM can also report whether the system is on AC or battery power, the battery status, and the remaining battery life.

Append: Concatenation of one file or string onto the end of another (this use is not related to the DOS, Windows 95 / Windows NT, and OS/2 external command named APPEND).

Application: A program run from the command prompt or a batch file. Used broadly to mean any program other than the command processor; and more narrowly to mean a program with a specific purpose such as a spreadsheet or word processing program, as opposed to a utility.

Archive: A file attribute indicating that the file has been modified since the last backup (most backup programs clear this attribute). Also sometimes refers to a single file (such as a *.ZIP* file) which contains a number of other files in compressed form.

Argument: See **Parameter**.

ASCII File: A file containing ASCII text, as opposed to a binary file which may contain numbers, or other information that cannot be sensibly interpreted as text.

ASCII: The American Standard Code for Information Interchange, which defines numeric values for 128 different characters comprising the English alphabet, numbers, punctuation, and some control characters.

Attribute: A characteristic of a file which can be set or cleared. The standard attributes are Read-Only, Hidden, System, and Archive; other attributes include Directory and Volume Label.

AUTOEXEC.BAT: A batch file which is executed automatically each time DOS or Windows 95 starts, typically stored in the root directory of the boot drive.

Automatic Batch Files: See **AUTOEXEC.BAT**, **TCSTART**, and **TCEXIT**.

Automatic Directory Change: A Take Command feature which allows you to change directories by typing the directory name and a backslash [\] at the prompt.

Glossary - B

Base Memory: Under DOS and Windows 95, the portion of your computer's memory available for use by DOS, the DOS command processor, and DOS application programs. This area generally consists of the first 640K or 655,360 bytes of the computer's memory.

Base Name: The file name without a drive, path, or extension. For example, in the file name *C:\DIR1\LETTER.DAT* the base name is *LETTER*.

BAT File: See **Batch File**.

Batch File Parameter: A numbered variable (*e.g.* %2) used within a batch file, allowing a different value to be used at that spot in the file each time it is executed.

Batch File: A text file containing a sequence of commands for the command processor to execute. Batch files are used to save command sequences so that they can be re-executed at any time, transferred to another system, etc. The extension of a batch file may be *.BAT*, *.CMD*, or *.BTM*, depending on the operating system and command processor you are using.

Binary File: A file containing information which does not represent or cannot sensibly be interpreted as text. See also **ASCII File**.

BIOS or **Basic Input Output System**: The software (or "firmware") stored on chips inside PC systems. The BIOS provides basic low-level control of devices required to operate the system, such as the keyboard, floppy disk, and screen; it also handles system self-tests at startup, and intiates loading of the operating system.

Block Device: A physical device for input or output which can transmit or receive large blocks of data while the computer is engaged in other activities. Examples include disk, tape, and CD-ROM drives. See also **Character Device**.

Boot Directory: The current directory at the time the system is booted, usually the root directory of the boot drive.

Boot Drive: The disk drive that the system is booted from, usually A: (the floppy disk) or C: (the hard disk).

Boot: The process of starting the computer and loading the operating system into memory. See also **Reboot**, **Cold Reboot**, and **Warm Reboot**.

Border: A narrow strip surrounding the standard text display area of the screen. The color of the border area can be controlled by special video commands under some operating systems.

Break: A signal sent to a program to tell it to halt what it is doing. The **Ctrl-C** key or **Ctrl-Break** key is used to send this signal. Some external commands abort when they receive a break signal; others return to a previous screen or menu, or abort the current operation.

BTM File: A special type of Take Command batch file which is loaded into memory to speed up execution.

Buffer: An area of memory set aside for storage. For example, disk buffers are used to save information as it is transferred between your program and the disk, and the keyboard buffer holds

keystrokes until a program can use them.

Glossary - C

CDFS or CD-ROM File System: The file system which supports CD-ROM drives. This is typically implemented as a distinct file system in 32-bit operating systems like OS/2 and Windows NT. On other platforms it is implemented as a component of or addition to the underlying general file system for disk drives.

Character Device: A physical device for input or output which must communicate with your computer one character at a time. Examples include the console, communications ports, and printers. See also **Block Device**.

Character Mode: A display mode in which output is displayed in a fixed font, typically with 80 columns in a line and 25 lines on the screen (some systems allow you to increase the number of rows and columns to other fixed sizes), and which cannot display graphics or pictures. See also **Graphics Mode**.

CMD File: See **Batch File**.

CMDLINE: An environment variable used to extend the command line passed to another program beyond its normal length limits.

Code Page: A setting which tells DOS or OS/2 which character set to use and how to retrieve and display date, time, and other information in the format appropriate to a particular country. See also **Country Settings**.

Cold Reboot: The process of restarting the computer in a way that physically resets most hardware devices, typically by pressing a reset button, or by turning the power off and back on. See also **Warm Reboot**.

Command Completion: A Take Command feature which allows you to recall a previous command by typing the first few letters of the command, then an up-arrow or down-arrow.

Command Echoing: A feature which displays commands as they are executed. Echoing can be turned on and off.

Command Grouping: A Take Command feature which allows you to group several commands with parentheses, and have them treated as a single command for most purposes.

Command History Window: A pop-up window used by Take Command to display the command history, allowing you to choose a previous command to modify and/or execute.

Command History: A Take Command feature which retains commands you have executed, so that they can be modified and re-executed later.

Command Processor: A program which interprets commands and executes other programs. Sometimes also called a **Command Interpreter**.

Command Recall: See **Command History**.

Command Separator: A character used to separate multiple commands on the same command line.

Command Tail: The portion of a command consisting of all the arguments, *i.e.*, everything but the command name itself.

Compound Command: See **Multiple Commands**.

Compression: An operating system feature which compresses data as it is stored in a disk file, and decompresses it as it is read back, resulting in more efficient use of disk space (at a slight cost in processor time to perform the compression and decompression). More generally, an approach to data storage which reduces repeated or redundant information to a smaller number of bytes in the compressed version than in the original, in order to minimize the space required to store the informatrion.

COMSPEC: An environment variable which defines where to find the character-mode command processor to start a secondary shell under DOS, Windows 95 / Windows NT, or OS/2.

Conditional Commands: A Take Command feature allowing commands to be executed or skipped depending on the results of a previous command. See also **Exit Code**.

CONFIG.SYS: A file used by DOS, Windows 95, and OS/2 to specify which programs should be loaded when the system boots.

Console: The PC keyboard and display.

Console Mode: See **Character Mode**.

Control Character: A character which is part of the ASCII code, but does not have a normal text representation, and which can usually be generated by pressing the Ctrl key along with another key.

Coprocessor: See **Numeric Coprocessor**.

Country Settings: The internal settings which tell the operating system how to interpret keyboard characters which vary from country to country, which character set to use, and how to retrieve and display date, time, and other information in the format appropriate to a particular country. See also **Code Page**.

CPU: The Central Processing Unit which performs all logic and most calculations in a computer. In PC-compatible systems, the CPU is on a single microprocessor chip.

CR or **Carriage Return**: The ASCII character "carriage return" (decimal value 13), generated by pressing the **Enter** key on the keyboard, and stored in most ASCII files at the end of each line.

Critical Error: An error, usually related to a physical or hardware problem with input, output, or network access, which prevents a program from continuing.

Current Directory: The directory in which all file operations will take place unless otherwise specified. The current directory is typically displayed as part of the command prompt. Also called the **Current Working Directory**.

Current Drive: The disk drive on which all file operations will take place unless otherwise specified. The current drive is typically displayed as part of the command prompt.

Cursor: A movable marker on the screen to show where text will be entered when you type at the keyboard, or which object on the screen will be affected when a mouse button is clicked. In character mode only the text cursor is available; graphical systems typically show both a mouse cursor and, when text can be entered, a separate text cursor.

Glossary - D

Date Range: A Take Command feature which allows you to select files based on the date and time they were last modified.

Date Stamp: Information stored in a file's directory entry to show the date on which the file was last modified. See also **Time Stamp**.

Default Directory: See **Current Directory**

Default Drive: See **Current Drive**.

Delete Tracking: An operating system or utility software feature which is designed to allow you to "undelete" or recover files which have recently been deleted. Delete tracking typically works by temporarily retaining the deleted files and / or information about the deleted files in a special area of the disk.

Description: A string of characters assigned to describe a file with the Take Command DESCRIBE command, typically stored in the *DESCRIPT.ION* file in each subdirectory.

Destination: In file processing commands (*e.g.* COPY or MOVE), the name or directory files should have after any copying or modification has taken place, generally the last specification on the command line. See also **Source**.

Detached Process: A program which is "detached" from the normal means of user input and output, and cannot use the keyboard, mouse, or video display.

Device Driver: A program which allows the operating system to communicate with a device, and which is loaded into memory when the system boots. Device drivers are also used to manage memory or for other similar internal functions.

Device: A physical device for input or output such as the console, a communications port, or a printer. Sometimes "device" is used to refer to character devices, and excludes block devices.

Directive: An individual item in the *TCMD.INI* file, used to control the configuration of Take Command.

Directory History Window: See **Directory History**.

Directory History: A Take Command feature which allows you to recall recently-used directory names in a popup window, and choose one to switch to.

Directory Stack: A Take Command feature, implemented through the PUSHD and POPD commands, which allows you to save the current directory and return to it later. See also **Stack**.

Directory Tree: The branching structure of directories on a hard disk, starting at the root directory. The root of the tree is usually considered as the "top" of the structure, so the actual structure can be visualized as an upside-down tree with the root at the top and branches going "down". A portion or branch of the directory tree is sometimes called a "subtree".

Directory: A portion of any disk, identified by a name and a relationship to other directories in a "tree" structure, with the tree starting at the root directory. A directory separates files on the disk into logical groups, but does not represent a physical division of the data on the disk.

DOS Memory: See **Base Memory**.

DOS Session: See **Session**.

DPMI or **DOS Protected Mode Interface**: A specification which allows DOS programs to access memory beyond 1 MB in order to manage larger programs or larger amounts of information than will fit in base memory. DPMI support for DOS programs is provided by some DOS memory managers, and by OS/2, Windows 3.1 and above, Windows 95, and Windows NT.

Drive Letter: A letter used to designate a specific local disk volume, or part or all of a network server drive. In most cases drive letters range from A - Z, but some network operating systems allow the use of certain punctuation characters as drive letters in order to support more than 26 volumes.

Glossary - E

Echo: See **Command Echoing**.

EMS Memory: Under DOS and Windows 95, memory which conforms to the Lotus / Intel / Microsoft Expanded Memory Specification (LIM EMS), which allows programs to access large amounts of memory outside of base memory or extended memory.

Environment: An area of memory which contains multiple entries in the form "NAME=value". See also **Master Environment** and **Passed Environment**.

Environment Variable: The name of a single entry in the environment.

Error Level: A numeric value between 0 and 255 returned from an external command to indicate its result (*e.g.*, success, failure, response to a question). See also **Exit Code**.

Escape Character: In some contexts, the Take Command escape character, which is used to suppress the normal meaning of or give special meaning to the following character. In other cases, the specific ASCII character ESC. The meaning must be determined from the context.

Escape Sequence: A sequence of text characters which has a special meaning and is not treated as normal text. For example, the character sequence **<ESC>]K** (where <ESC> is the ASCII "escape" character, decimal value 27) will cause an ANSI driver to clear the screen from the cursor to the end of the current line, rather than simply displaying the string "ESC]K" on the screen. Similarly, in Take Command, the escape sequence **<Ctrl-X>f** on the command line (where <Ctrl-X> is the ASCII Ctrl-X character, decimal value 24) is translated to a form feed, and is not treated as the literal characters "<Ctrl-X>f".

Executable Extensions: A Take Command feature which allows you to specify the application to be executed when a file with a particular extension is named at the command prompt.

Executable File: A file, usually with the extension *.COM* or *.EXE*, which can be loaded into memory and run as a program.

Exit Code: The result code returned by an external command or an internal command. Take Command internal commands return an exit code of 0 if successful, or non-zero if unsuccessful. See also **Errorlevel**.

Expansion: The process Take Command goes through when it scans a command line and substitutes the appropriate actual values for aliases, alias parameters, batch file parameters, and environment variables. See also **Parsing**.

Extended ASCII Character: A character which is not part of the standard set of 128 ASCII characters, but is used on the PC as part of an extended set of 256 characters. These characters include international language symbols, and box and line drawing characters.

Extended Attributes: An OS/2 High Performance File System (HPFS) feature which allows storage of additional information about a file, separate from the file itself. Extended attributes are typically used to store icons for executable files, property or settings information, and other information added by the user.

Extended Directory Search: A Take Command feature which maintains a directory search "database" or list, typically including all directories in your system,and allows you to change quickly to any directory in

the list.

Extended Key Code: The code for a key on the PC keyboard which has no representation in the standard ASCII character set, such as a function key, cursor key, or **Alt** plus another key. The extended key code for a key is often the same as the scan code for that key.

Extended Memory: Any memory on a computer system with a 286, 386, 486, or Pentium processor which is above the first 1 MB (one megabyte, or 1024*1024 bytes) of memory. See also **XMS**.

Extended Parent Directory Names: A Take Command feature which allows you to use additional periods in a directory name to represent directories which are successively higher in the directory tree.

Extended Wildcard: A Take Command feature which extends the traditional wildcard syntax and allows you to use multiple wildcard characters, and character ranges (*e.g.* [a-f] for the letters A through F). See also **Wildcard**.

Extension: The final portion of a file name, preceded by a period. For example, in the file name *C:\DIR1\LETTER.DAT* the extension is *.DAT*. In a long filename which contains multiple periods, the extension is usually considered to be the portion of the name after the final period.

External Command: A program which resides in an executable file, as opposed to an internal command which is part of the command processor.

EXTPROC: A command processor feature which allows you to designate a specific external program to run a particular batch file.

Glossary - F

FAT File System: The traditional file system used by DOS to store files on diskettes and hard disks; also supported by OS/2 and Windows NT. Uses a **F**ile **A**llocation **T**able to keep track of allocated and unallocated space on the disk.

FAT32 File System: An extension of the FAT file system, available in Windows 95 OEM Service Release 2 ("OEMSR2") and later versions, which supports long filenames, and supports much larger hard disks than the FAT system. FAT32 drives cannot be accessed under any operating system other than Windows 95 OEMSR2 and later versions. See also **VFAT File System**.

FAT-Compatible File Name: See **SFN**.

FF or **Form Feed**: The ASCII character "form feed" (decimal value 12), which typically causes a printer to skip to a new page. The FF character is not normally entered from the keyboard, but in many cases it can be generated, if necessary, by holding the Alt key, pressing 0-1-2, and releasing the Alt key.

File Attribute: See **Attribute**.

File Description: See **Description**.

File Exclusion Range: A Take Command feature which allows you to exclude files from processing by internal commands based on their names.

Filename Completion: A Take Command feature which allows you to type part of a filename on the command line, and have the command processor fill in the rest for you.

Free Memory: Usually, the amount of total memory which is unoccupied and available for applications.

Glossary - G

Graphics Mode: A display mode in which output is displayed in any one of a range of fonts, typically in resizable windows with a variable number of text rows and columns, and which supports the display of graphics and pictures along with text. See also **Character Mode**.

Glossary - H

Hidden: A file attribute indicating that the file should not be displayed with a normal DIR command, and should not be made available to programs unless they specifically request access to hidden files.

History Window: See **Command History Window** and **Directory History**.

History: See **Command History**.

HMA or **High Memory Area**: The area of PC memory located in the first 64K bytes above the 1 megabyte that DOS can address directly. The HMA can be made addressable from DOS programs using special hardware facilities, or an XMS driver.

HPFS or **High Performance File System**: A file system distributed with OS/2 and Windows NT 3.51 and below which allows longer file names, supports larger drives, and provides better performance than the traditional FAT file system.

Glossary - I

IFS or **Installable File System**: A file system which can be loaded when required to support devices such as CD-ROM or network drives, or non-default disk formats like HPFS (in OS/2) or NTFS (in Windows NT). Installable file systems are primarily supported 32-bit operating systems like OS/2 and Windows NT. Depending on operating system design they may be loaded at boot time, or loaded and unloaded dynamically while the system is running.

Include List: A concise method of specifying several files or groups of files in the same directory, for use with all Take Command commands which take file names as arguments.

Inheritance: A system which allows one program to pass information or settings on to another, often to a second copy of the same program.

.INI File: The Take Command initialization file containing directives which set the initial configuration of the command processor.

Insert Mode: When editing text, a mode in which newly typed characters are inserted into the line at the cursor position, rather than overwriting existing characters on the line. See also **Overstrike Mode**.

Internal Command: A command which is part of the command processor, as opposed to an external command.

Internal Variables: Environment variables created by Take Command to provide information about your system. Internal variables are evaluated each time they are used, and are not actually stored in the environment.

Glossary - K

Key Code: The code passed to a program when a key is pressed on the keyboard. Depending on the key that is pressed, and the software handling the keyboard, the code can be an ASCII code, a scan code, or an extended key code.

Key Mapping: A Take Command feature which allows you to assign new keystrokes for command line functions such as manipulating the command history or completing file names.

Keyboard Buffer: A buffer which holds keystrokes you have typed that have not yet been used by the currently executing program.

Keystroke Alias: An alias assigned to a key, so that it can be invoked or recalled with a single keystroke.

Glossary - L

Label: A marker in a batch file, with the format **:name**, allowing GOTO and GOSUB commands to "jump" to that point in the file. See also **Volume Label**.

LF or **Line Feed**: The ASCII character "line feed" (decimal value 10), stored in most ASCII files at the end of each line, after the CR character. The LF character is not normally entered from the keyboard, but in many cases it can be generated, if necessary, by pressing Ctrl-Enter.

LFN or **Long File Name**: A file name which does not conform to FAT file system restrictions, either because it is longer than the traditional 8 character name plus 3 character extension, or because it contains periods, spaces, or other characters not allowed in a traditional FAT file name. See also **SFN**.

LFN File System: A file system which extends the traditional FAT system to support long filenames and possibly larger hard drives. An LFN file system stores both a long and short name for a file, not just a long name. The short name is sometimes called the "alternate" name. See also **LFN**, **SFN**, **VFAT File System**, and **FAT32 File System**.

Logging: A Take Command feature, implemented via the LOG command, which allows you to save a record of the commands you execute.

Glossary - M

Master Environment: The master copy of the environment maintained by the command processor.

Modulo: The remainder after an integer division. For example 11 modulo 3 is 2, because when 11 is divided by 3 the remainder is 2.

Multiple Commands: A Take Command feature which allows multiple commands to be placed on a line, separated by a caret [**^**], or another user-defined character.

Multitasking: A capability of some software (and the related hardware) which allows two or more programs to run apparently simultaneously on the same computer. Multitasking software for PC compatible systems includes operating environments like Windows 3, and complete operating systems like OS/2, Windows 95, and Windows NT.

Glossary - N

Network File System: Software which runs over a network to allow access to files on the server. A network file system may support the same options as the file system used on local drives, or it may be more or less restrictive than the local file system about file names, disk volume capacity, and other similar features.

Network: A system which allows several computers to be connected together to share files, printers, modems, or other resources, and to pass electronic mail or other information between the systems on the network.

NTFS or **New Technology File System**: A file system distributed with Windows NT which allows longer file names, supports larger drives, and provides better performance than the traditional FAT file system.

Numeric Coprocessor: A chip which works in conjunction with an Intel 8086, 80286, 80386, 80486, or Pentium CPU to perform decimal arithmetic ("floating point") calculations. Some 80486s and the Pentium CPU have the numeric coprocessor built in to the CPU chip; in all other cases it is on a physically separate chip, and is optional (when the coprocessor is not avilable, the CPU performs decimal arithmetic through other, much slower methods).

Glossary - O

Operating System: A collection of software which loads when the computer is started, provides services to other software, and ensures that programs don't interfere with each other while they are running.

Option: See **Switch**.

OR: A logical combination of two true or false conditions. If both conditions are false the result is false; if either condition is true the result is true.

Overstrike Mode: When editing text, a mode in which newly typed characters overwrite existing characters on the line, rather than being inserted into the line at the cursor position. See also **Insert Mode**.

Glossary - P

Parameter: A piece of additional information placed after a command or function name. For example, in the command DIR XYZ, XYZ is a parameter. Also used to refer to an alias parameter or batch file parameter.

Parent Directory: The directory in which a particular subdirectory resides, often seen as the directory "above" a subdirectory.

Parsing: The process Take Command performs to analyze the command line, perform alias and environment variable expansion, and find the appropriate internal command or external command to execute. More generally, the process of breaking down a string or message into its individual components in order to process them properly.

Passed Environment: A copy of the master environment created before running an application, so that any changes made by the application will not affect the master environment.

Path: A specification of all the directories a file resides in. For example, the path for *C:\WPFILES\MYDIR\MEMO.TXT* is *C:\WPFILES\MYDIR*. Also used to refer to the environment variable PATH, which contains a series of path specifications used when searching for external commands and batch files.

Pipe: A method for collecting the standard output of one program and passing it on as the standard input of the next program to be executed, signified by a vertical bar "|" on the command line. See also **Redirection**.

Previous Working Directory: The working directory used most recently, just prior to the current working directory. For example, if *C:\DATA* is the current working directory and you switch to *D:\UTIL*, *C:\DATA* then becomes the previous working directory.

Primary Shell: The copy of the character-mode command processor which is loaded by the operating system when the system boots or a session opens.

Glossary - R

RAM or **Random Access Memory**: The physical memory used to store data while a computer is operating. The information in most types of RAM is lost when power is turned off.

RAM Disk: A pseudo "disk drive", created by software, which appears like a normal physical disk drive to programs. Sometimes also called a **Virtual Disk**.

Range: See **Date Range**, **Size Range**, and **Time Range**.

Read-Only: A file attribute indicating that the file can be read, but not written or deleted by the operating system or the command processor unless special commands are used.

Reboot: The process of restarting the computer with software, with the keyboard (*e.g.* by pressing Ctrl-Alt-Del), by pressing a reset button, or by turning the power off and back on. See also **Cold Reboot** and **Warm Reboot**.

Redirection: A method for collecting output from a program in a file, and/or of providing the input for a program from a file. See also **Pipe**.

Registry: A hierarchically organized data file maintained by Windows 3.x, Windows 95, and Windows NT to hold system parameters, hardware and software settings, and other similar information used by the operating system or by other software packages.

REXX: A file and text processing language developed by IBM, and available on many PC and other platforms.

ROM or **Read Only Memory**: A physical memory device used to store information which cannot be readily modified, such as the BIOS built into each PC system. The information in ROM is typically retained when power is turned off.

Root Directory: The first directory on any disk, from which all other directories are "descended." The root directory is referenced with a single backslash [\].

Glossary - S

Scan Code: The physical code for a key on the PC keyboard. For the original U.S. English keyboard layout the scan code represents the physical position of the key, starting with 1 for the key in the upper left corner (Esc), and increasing from left to right and top to bottom. This order will vary for more recent keyboards or those designed for other countries or languages.

Search Path: See **PATH**.

Secondary Shell: A copy of the command processor which is started by another program, rather than by the operating system.

Session: A general term for the individual windows or tasks started by a multitasking system. For example, under OS/2 you might run an OS/2 program in one session, and a DOS program in another.

SFN or **Short File Name**: A file name which follows the rules of the traditional FAT file system: a name of 1 - 8 characters and an extension of 0 - 3 characters, each consisting of only alphabetic and numeric characters plus the punctuation marks **! # $ % & ' ( ) - @ ^ _ ` { }** and **~**. See also **LFN**.

Shell: See **Command Processor**. Also used to refer to a program which gives access to operating system functions and commands through a menu- or mouse-driven system, or which replaces the primary user interface of the operating system.

Size Range: A Take Command feature which allows you to select files based on their size.

Source: In file processing commands (*e.g.* COPY or MOVE), the original files before any copying or modification has taken place, *i.e.*, those specified earlier on the command line. See also **Destination**.

Stack: An area of memory used by any program to store temporary data while the program is running; more generally, any such storage area where the last item stored is normally the first one removed.

Standard Error, Standard Input, and **Standard Output**: The file(s) or character device(s) where a program respectively displays error messages, obtains its normal input, and displays its normal output. Standard error, standard input, and standard output normally refer to the console, unless redirection is used.

Subdirectory: Any directory other than the root directory.

Subtree: See **Directory Tree**.

Swap File: A disk file created by an operating system or a program to store unused information on disk, and thereby free up memory for other purposes.

Switch: A parameter for an internal command or application which specifies a particular behavior or setting. For example, the command "DIR /P" might be referred to as "having the /P switch set".

System: A file attribute indicating that the file belongs to the operating system or command processor, and should not be accessed by other programs.

Glossary - T

Target: See **Destination**.

TCEXIT: A batch file which is executed whenever Take Command exits.

TCSTART: A batch file which is executed whenever Take Command starts.

Time Range: A Take Command feature which allows you to select files based on the time they were last modified.

Time Stamp: Information stored in a file's directory entry to show the time at which the file was last modified. See also **Date Stamp**.

Tree: See **Directory Tree**.

Glossary - U

UMB or **Upper Memory Block**: An XMS Upper Memory Block, whose address is above the end of base memory (normally, above 640K), but within the 1 megabyte of memory that DOS can address directly.

UNC or **Universal Naming Convention**: A common method for accessing files on a network drive without using a "mapped" drive letter. Names specified this way are called UNC names, and typically appear as **\\server\volume\path\filename**, where **server** is the name of the network server where the files reside, **volume** is the name of a disk volume on that server, and the **path\filename** portion is a directory name and file name

Glossary - V

Variable Expansion: The process of scanning a command line and replacing each environment variable name, alias parameter, or batch file parameter with its value.

Variable Functions: Functions provided by Take Command to manipulate strings, dates, and filenames; perform arithmetic; read and write files; and perform other similar functions. Variable functions are similar to static environment variables or internal variables, but have parameters and can perform actions rather than just returning static information.

Variable: See **Alias Parameter**, **Batch File Parameter**, and **Environment Variable**.

VFAT File System: An extension of the FAT file system, available in Windows 95 and Windows NT, which supports long filenames. Also see **LFN** and **FAT32 File System**.

Virtual Disk: See **RAM Disk**.

Volume Label: A special, hidden file placed on any disk, whose name constitutes a "label" for the entire disk.

Volume: See **Disk Drive**.

Glossary - W

Warm Reboot: The process of restarting the computer with software, or with the keyboard (*e.g.* by pressing Ctrl-Alt-Del), typically without physically resetting any hardware devices. See also **Cold Reboot**.

White Space Character: A character used to separate arguments on the command line. The white space characters recognized by Take Command are the space, tab, and comma.

Wildcard: A character ("*" or "?") used in a filename to specify the possibility that any single character ("?") or sequence of characters ("*") can occur at that point in the actual name. See also **Extended Wildcard**.

Windows NT File System: See **NTFS**.

Glossary - X

XMS Memory: A software method for accessing extended memory under DOS and Windows 95. XMS memory is not additional memory, but is a software specification only.

XOR (exclusive OR): A logical combination of two true or false conditions. If both conditions are false or both conditions are true the result is false; if either condition is true and the other is false the result is true.