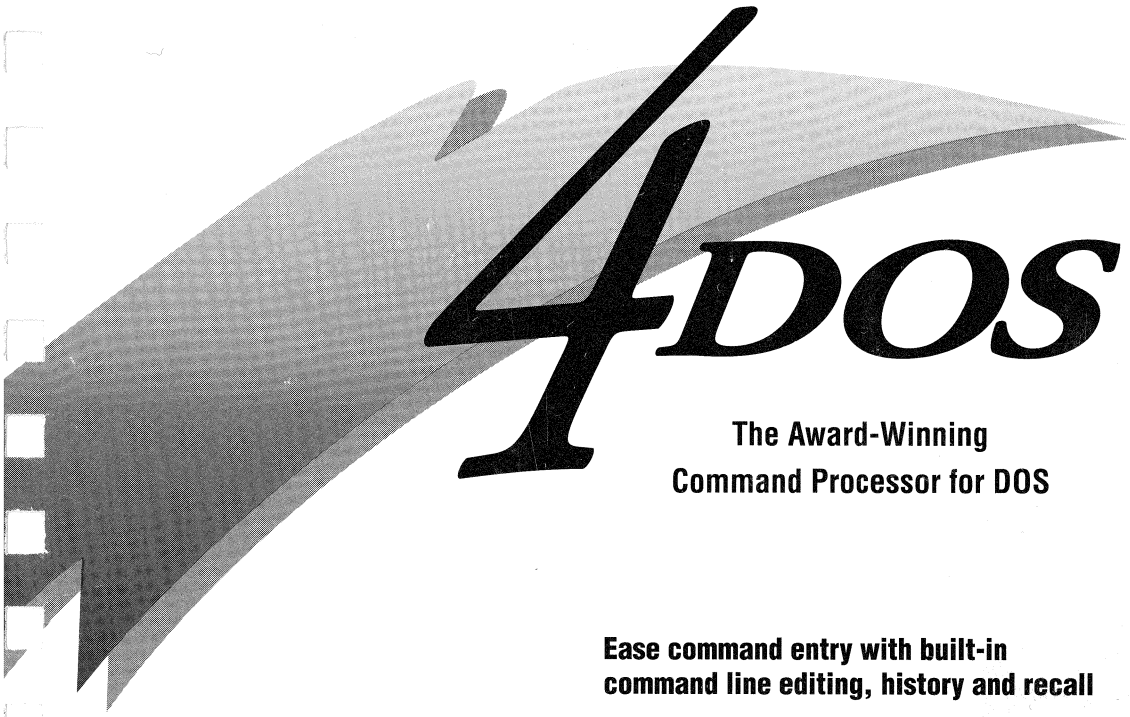


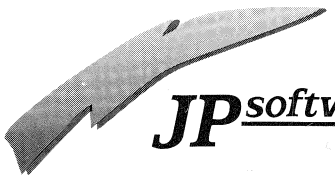
The Award-Winning
Command Processor for DOS

Ease command entry with built-in command line editing, history and recall

Enjoy enhancements to all standard DOS commands, plus over 40 new commands

Get help fast with complete, cross-referenced, on-line DOS help

Speed all your batch files, add power with dozens of new batch programming features



4DOS®

Developed By
Rex Conn and Tom Rawson

Documentation By
Hardin Brothers, Tom Rawson, and Rex Conn

Published By
JP Software Inc.
P.O. Box 1470
East Arlington, MA 02174
USA
(617) 646-3975
fax (617) 646-0904

ACKNOWLEDGMENTS

We couldn't produce a product like 4DOS without the dedication and quality work of many people. Our thanks to:

JP Software Staff: Mike Bessy, Helen Coyne, Hayyim Feldman, Henry Harvey, Susan Mampre, Ellen Stone.

Beta Test Support: David Moskowitz, Guy Scharf, Larry Finkelstein, and Martin Schiff, of CompuServe's CONSULT forum.

Online Support: Brian Miller and Tess Heder of Channel 1 BBS; Don Watkins and Connie Kageyama of CompuServe's IBMNET.

Help System: Scott McGrath and Ross Neilson Wentworth.

Beta Testers: We can't list all of our beta testers here! A special thanks to all of you who helped make 4DOS elegant, reliable, and friendly.

The following tools are used in creating and maintaining 4DOS and the 4DOS Help System:

Compilers:	Microsoft C 6.0 and Macro Assembler 5.1 and 6.0, Borland Turbo Pascal 5.5 and 6.0
Libraries:	Object Professional (Turbo Power Software), Spontaneous Assembly (Base Two Development)
Editors:	Edix (Emerging Technology), Brief (Solution Systems)
Debuggers:	Periscope (The Periscope Company), Soft-ICE (Nu-Mega Technologies)
Version Control:	PVCS (Sage Software)
Documentation:	Microsoft Word for Windows with Adobe Type Manager

Printing by Causeway Print, Boston. Disk duplication by Diversified Systems Group, Issaquah, WA.

CONTENTS

CONTACTING JP SOFTWARE

You can contact JP Software at any of the following addresses. Our normal business hours are 9:00 AM to 5:00 PM weekdays, eastern US time. For more information about contacting us for technical support see page 7.

By mail:

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

By telephone:

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

Electronically:

CompuServe:	General 75300,210 Technical Support 75300,1215
Bix:	"trawson"
MCI Mail:	470-7811
Internet:	General 75300.210@compuserve.com Technical Support 75300.1215@compuserve.com
BBS Support:	Via Channel 1 BBS, Boston, 617-354-8873 at 2400 baud, no parity, 8 data bits, 1 stop bit

INTRODUCTION

Welcome, and thanks for purchasing 4DOS!

We started developing 4DOS when we realized that our computers could be a lot more powerful and a lot more helpful than they were. Whether you are a computer novice or an experienced power user, we think that 4DOS will help you get the most out of your IBM PC or compatible computer.

Technically, 4DOS is a command interpreter or "DOS Shell." That means that it reacts to the commands you type at the C> prompt. We've designed 4DOS so that you don't have to change your computing habits or unlearn anything to use it. If you know how to display a directory, copy a file, or start an application program, you already know how to use 4DOS. 4DOS understands all of the commands you know and adds to them. Its purpose is to make DOS friendlier, easier to use, and much more powerful and versatile, without requiring you to use or learn a new program, a new set of commands, or a new style of work.

Once you have 4DOS installed, you can learn its new features at your own pace. It has more than 40 new commands and scores of enhanced features, but you don't have to learn them all at once. Relax, enjoy 4DOS's power, and browse through the manual occasionally. Press the **F1** key whenever you need help. 4DOS will soon become an essential part of your computer, and you'll wonder how you ever got along without it.

If you want to take 4DOS for a spin without performing a complete installation, see the Guided Tour beginning on page 19.

We are constantly working to improve 4DOS. If you have suggestions for features or commands that you think we should include in the next version, or any other way we could improve our product, please let us know. Many of the improvements in this version of 4DOS were suggested by our users, and while we can't promise to include every suggested feature, we really do appreciate and pay attention to your comments.

If you're an OS/2 user, JP Software offers another product, 4OS2, as a complete replacement for the OS/2 command processor (*CMD.EXE*). 4OS2 provides the same command set and syntax as 4DOS, with several OS/2 commands added. You can use it to maintain a common working environment and run the same

batch files under OS/2 and DOS. 4OS2 is available at a discounted price for 4DOS users. Contact JP Software for more information.

How to Use This Manual

We have designed this manual to serve as a gentle introduction to 4DOS for novice users, as a tutorial for those who want to get the most out of 4DOS, and as a reference manual for advanced 4DOS users.

You will likely find some parts of the manual too simple or too technical for your tastes. Unless you are convinced that one of those sections holds just the information you need for a specific task, feel free to skip to the next part of the manual that is more to your liking. You can use almost every feature of 4DOS without having to worry about other features or commands.

As you read the manual, you will see the symbol ❖ next to certain paragraphs. This indicates a more in-depth discussion or a more advanced topic which you can skip if you wish to stick with the basics. Come back to this topic later for more details, or if you're having trouble with the particular issue it discusses. In many cases, the remainder of the section you're reading after such a symbol will be devoted to similar advanced information. If you see the ❖ next to a section heading, it means the entire section contains such information.

You may find the information in such marked sections useful even if you're relatively new to computers or to 4DOS. However you can skip the marked section and still understand and use the basic topic of the section you're reading.

An exclamation point [!] to the left of a paragraph means that paragraph contains a caution or warning you may need to observe when using the feature it discusses.

This manual is divided into eight chapters and three appendices, plus a glossary and index. Here's a brief overview of what you'll find in each:

Chapter 1 / 4DOS Features

We begin with a short summary of some of 4DOS's features to give you a taste of what 4DOS is about. Start here if you are new to 4DOS.

Chapter 2 / Installation

Everyone should read this short chapter. The installation instructions are simple (and quite different from previous versions of 4DOS).

Chapter 3 / A Guided Tour of 4DOS

This chapter will take you on a short, interactive tour of some of 4DOS's most powerful features. By the time you finish the tour, you will have a good idea of what 4DOS will do for you. If you are new to 4DOS, be sure to take the tour.

Chapter 4 / General Concepts

This reference chapter is an introduction to several terms and concepts that we use throughout the manual. If you're a novice, you might want to browse through the entire chapter. If you're a power user and all of the topics in this chapter seem simple, then go on to the next chapter. If you think you need to brush up a bit on the basics of a couple of terms and ideas, you'll probably find them here.

Chapter 5 / Using 4DOS

This chapter is for everyone. It contains a description of 4DOS's features and lots of examples to help you learn to use each one. Even if you are a novice user and want to ignore some of these features until later, skim through this chapter to get an idea of what is available and where to find the information that you will eventually want. 4DOS offers both features which are not related to specific commands, and a complete set of over 80 internal commands. This chapter has complete reference information on all of the non-command features, and introduces a few commands as well. Reference information on 4DOS commands is in Chapter 8.

Chapter 6 / Options and Tuning

This chapter is for advanced users who want to be sure that 4DOS is running at top efficiency on their systems. It includes detailed information on setting up 4DOS and on changing your 4DOS configuration.

Chapter 7 / Using 4DOS With Your Hardware and Software

This chapter contains tips on using 4DOS with many popular kinds of PC hardware and software, and information that will be useful if you think 4DOS may conflict with your hardware or other software.

Chapter 8 / 4DOS Command Reference

4DOS knows over 80 internal commands. This chapter explains the purpose of each command and tells you how to use it. It has examples that will help you learn each command and the technical details you will need to get the command to behave exactly as you wish.

Appendices

We've included some helpful tables here, and some information for those who like to know technical details. There are three appendices.

Appendix A lists all 4DOS error messages. Look here if you need an explanation of an error message, or if you aren't sure if the message came from 4DOS.

Appendix B lists codes for the ASCII character set (your computer's internal method for representing letters, digits, and other characters) and for the keys on your keyboard. These codes can be useful with certain 4DOS commands.

Appendix C covers technical information and programming interfaces. You shouldn't need to use it unless you are an advanced user or a programmer writing add-on software for 4DOS.

Glossary

If you need help with any of the terminology in this manual, look here.

Index

If you can't find the information you need, this should help you find it.

Quick Reference Card

If you can't remember a key combination, command format, color name, or other general information, look here. Whether you are a 4DOS novice or expert, you will probably want to keep the Quick Reference card near your computer for those times when you need just a small jog to your memory.

Additional Files

Files distributed with 4DOS cover important additional information beyond what's included in this manual. *README.DOC* contains general notes, highlights of the latest release, and brief installation instructions for those upgrading from a downloaded copy of 4DOS. *UPDATE.DOC* covers any changes or corrections in the manual, and detailed information for users with older versions on what has changed in the latest release. *APPNOTES.DOC* contains application notes for a variety of other software packages to help you use those packages to best advantage with 4DOS.

Registration and Upgrade Information

If you purchased 4DOS from a software dealer, your copy came with a registration card. **Please fill out this card and return it promptly to JP Software.** Returning the registration card ensures that we have a record of your registration, and enables you to receive ongoing technical support and notices of upgrades. If you purchased 4DOS directly from JP Software, you are already registered and no registration card is necessary.

Once you are a registered 4DOS user, you will receive:

- A subscription to *The Prompt Solution*, JP Software's publication for 4DOS users and other customers. *The Prompt Solution* includes tips for using 4DOS, notification of upgrades, and information about 4DOS enhancements and other products from JP Software.
- Technical support via electronic mail, fax, mail, or telephone.
- A free or low-cost upgrade to the next version of 4DOS; see below for details.

4DOS is upgraded regularly through **maintenance releases**, designed to fix minor problems or improve compatibility, and **major upgrades** which contain enhancements and additional features. Maintenance releases are identified by a change in the hundredths digit of the version number, for example from 4.0 to 4.01 or 4.02. New versions are identified by a change in the tenths digit or "ones" digit, for example from 4.0 to 4.1 or 5.0.

As a registered user you will automatically be notified when a major upgrade of 4DOS is released. The first major upgrade released after your initial purchase of 4DOS can be obtained at no charge if you download it from one of our support areas (see below). A new manual will be available for download from the same sources. If you do not have a modem or prefer to obtain the new version on diskette, there is a minimal processing charge; there is also a charge for a new printed manual. After you've received this first free or low-cost upgrade, there will be a standard upgrade charge to get additional major upgrades as they are released.

We don't send out notices when maintenance releases become available, because you don't usually need them unless you're having a problem. However, you can download maintenance releases from our support areas (see below), or order them on disk at a nominal cost. If you call with a problem that's been addressed in a maintenance release, we'll mail you a copy on disk or assist you in downloading it. Downloading a maintenance release or obtaining it on disk from JP Software is separate from and does not affect your eligibility for a free or low-cost upgrade to the next major upgrade of 4DOS.

As a registered user, you can download maintenance releases and your free or low-cost 4DOS upgrade from the JP Software support area on CompuServe (GO PCVENB, library 10), Bix (ibm.vendors/listings), or Channel 1 (see below). The file name(s) used for 4DOS updates vary from time to time depending on the extent of the changes made, but typically will look like *4DOS4*.ZIP*. The "*" refers to one or more characters further identifying the file. Check the directory listings and file descriptions on the service you are using to identify the files you need. Downloads are also available from many other local bulletin boards and online services. Lists of download locations, including bulletin boards outside the USA, are published from time to time in *The Prompt Solution* and are included in the SUPPORT.BBS file that comes with 4DOS.

We offer BBS support through Channel 1 in Boston, one of the largest and best-run bulletin boards in the US. Channel 1 is an independent bulletin board and is not owned or operated by JP Software. To access Channel 1 set your modem to 2400 baud, no parity, 8 data bits, and 1 stop bit, and dial 617-354-8873 (if you have a high-speed modem, additional numbers are available once you are online). Join conference 5 for 4DOS downloads. You can leave 4DOS support messages and download 4DOS files at no charge, but if you want to use any of Channel 1's other excellent services you may need to purchase a membership to have sufficient online time.

Technical Support

Technical support for 4DOS is available to all users. When you contact us for support please give us your 4DOS serial number so that we can verify your status as a customer, keep track of your inquiry properly, and contact you if necessary.

Often the best way to contact us for support is by modem in one of the following public 4DOS support conferences. The numbers in parentheses indicate the usual delay, in days, to receive a reply to a message.

CompuServe / ZiffNet: Primary support is via the JP Software section of the CompuServe PCVENB forum (GO PCVENB, section 10) (1 day). We monitor other CompuServe and ZiffNet forums, but response may vary depending on our workload.

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

BIX: Support is available via the ibm.vendors conference, topic 4DOS (3-5 days).

In addition, you can contact JP Software for support by mail, telephone, fax, or electronic mail. Addresses and phone numbers are listed on page vi of this manual.

Our goal is to return all telephone messages within 24 hours (weekends and holidays excluded). 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 may have tried to return your call and been unable to reach you.

Before contacting us for support, please check if the manual or other documentation answers your question. If you can't find what you need, try Chapter 4 / General Concepts (page 41), and the Index. If you're having trouble getting 4DOS to run properly, either alone or with your particular hardware or software, see Chapter 7 / Using 4DOS with Your Hardware and Software, on page 135, and the *APPNOTES.DOC* file on your 4DOS disk. Also look through the *README.DOC* and *UPDATE.DOC* files that came with 4DOS, as they may contain updates to this manual or other important information.

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

- What exactly did you do? A concise description of what steps you must take to make the problem appear is much more useful than a long analysis of what might be happening.
- What went wrong? At what point did the failure occur? If you saw an error message or other important or unusual information on the screen, what exactly did it say?
- Briefly, what techniques did you use to try to resolve the problem? What results did you get?
- What are the contents of your *CONFIG.SYS*, *AUTOEXEC.BAT*, *4START*, *4EXIT*, and *4DOS.INI* files, 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, are there any clues as to what programs you're using when the problem occurs?

CHAPTER 1 / 4DOS FEATURES

4DOS is a complete computing environment that works with all versions of MS-DOS and PC-DOS from 2.0 to 5; with DR-DOS 3.4, 5.0, and above; and in the OS/2 DOS compatibility box. It is compatible with virtually all application programs, with Microsoft Windows, and with task switching programs like DESQView and Back & Forth.

4DOS replaces the traditional DOS user interface with a more modern, friendlier, and more powerful one. 4DOS is completely compatible with traditional DOS commands, and adds dozens of new features that aren't available with any version of DOS. This chapter

Minimal Memory Usage

4DOS can take advantage of extended memory, expanded memory, and memory managers for 80286, 386, and 486 computers that let programs "load high." If you load both 4DOS and the environment "high," 4DOS will use only 256 bytes of base memory, less than any version of the traditional DOS command processor, *COMMAND.COM*. (See page 124.)

Online Help

4DOS has complete, full-screen, context-sensitive help for all of its commands and all DOS utilities. The F1 key pops up the help system at any time from the 4DOS prompt. The help system is cross-referenced and includes examples. (See page 63.)

Directory Navigation

4DOS's enhanced CD command lets you specify where to look for subdirectories you're changing to. CDD lets you switch drives and directories simultaneously, and the new automatic directory change feature changes directories for you when you type a directory name at the command line.

Faster and Better Batch Files

4DOS speeds up traditional DOS batch files, and introduces a new kind of batch file processing that is 5 to 10 times faster than traditional batch files as well. (See page 97.)

In addition, 4DOS has more than two dozen new batch file commands, and many other enhancements that will make your batch files easier to write and far more powerful.

Command Line Enhancements

4DOS replaces the traditional command line with a much friendlier and more powerful command line environment. Among the features which 4DOS offers are:

Command Line Editing: You can use the cursor keys to make corrections, in the same way that you would with a text editor. (See page 56.)

Command History and Recall: 4DOS keeps track of each command you type. You can recall any command and issue it again, or edit it to create a slightly different command. Commands can be recalled one at a time, or you can pop up a window of recently entered commands and choose the one you want. (See page 57.)

Automatic Filename Expansion: If you type part of a filename, with or without wildcards, 4DOS will fill in the complete filename for you at the touch of a key. (See page 60.)

Multiple Commands on a Single Line: You can type a series of commands on a single line instead of waiting for each one to finish before you issue the next one. (See page 62.)

Multiple Filenames: Most 4DOS commands can operate on multiple files at once. For example, you can copy several files at once from your hard drive to drive A with a command like:

```
copy *.wks *.dat *.txt  a:\
```

Point-and-Shoot File Selection: The SELECT command lets you choose files with the cursor keys instead of remembering their

names. It gives you a full "point-and-shoot" environment for other 4DOS commands. (See page 273.)

Aliases

An alias is a command you create, assigned to a name you select or to a key combination of your choosing. You can use aliases to rename commands, to set command defaults, and to create new commands that are a combination of other commands. When aliases are combined with the 4DOS multiple command feature, they act like very fast batch files. (See pages 94 and 164.)

File Descriptions

4DOS lets you assign a description (up to 40 characters long) to each of your files and directories. The descriptions are displayed with the DIR and SELECT commands, and move with their files when you use a COPY, DEL, MOVE, or RENAME command. (See page 197.)

Executable Extensions

Executable extensions let you associate a file extension with the program that processes files of that type. For example, you could associate *.BAS* files with the BASIC interpreter, or *.DBF* files with dBase or Foxbase. 4DOS runs the appropriate program automatically whenever you type the name of a file that has an extension you have defined as executable. (See page 75.)

Compatibility

You can use 4DOS:

- with all monochrome, CGA, EGA, and VGA video systems, with any number of screen rows and columns.
- with DOS-compatible networks, including Novell Netware, 3Com 3+, Banyan Vines, and Artisoft LANTastic.
- with all popular memory managers and task switchers.
- with virtually all commercial applications, utility programs, and memory-resident utilities (TSRs).

New Commands and Options

4DOS includes over 80 commands. A few are the same as traditional commands. Many are compatible with traditional commands but are enhanced with several additional options. The majority are unique to 4DOS. If you have always wished that DOS had a command to help in a special situation, you will probably find that command in 4DOS. A complete list of 4DOS commands and options, along with explanations of how to use them, begins on page 157.

CHAPTER 2 / INSTALLATION

Before you install 4DOS (or any other software, for that matter), you should make a bootable system diskette so you can recover in case of a power failure or other interruption during the installation process. To do so, put a fresh diskette in drive A and then type:

```
format a: /s
```

The FORMAT command will take a few minutes to prepare the floppy diskette and then will copy your system files to drive A. Once the process is complete, you should test your bootable floppy by leaving it in drive A and simultaneously pressing the **Ctrl**, **Alt**, and **Del** keys to make sure your system will boot up properly. Once that's done, put the floppy away in a safe place and reboot your computer normally.

Automated Installation

Now you are ready to install 4DOS. Put the distribution diskette into your first disk drive, drive A. (You can use drive B if you prefer.) Then log onto drive A by typing:

```
a:
```

and press the **Enter** key. One of the files on the distribution diskette, *README.1ST*, contains information that you should read before you install 4DOS on your computer. Type:

```
type readme.1st
```

to view the file. If you want to print a copy of the file, type:

```
copy readme.1st prn
```

Now you can start the installation process. Type:

```
install
```

and press the **Enter** key. If you are using a color video adapter with a monochrome monitor (for example, a laptop computer with a monochrome EGA display), type:

```
install /m
```

Once the installation program has started, just follow the instructions on the screen and 4DOS will install itself on your system.

The installation program will ask whether you want to perform a complete installation, a partial installation in order to run the 4DOS Tour, or whether you want to retrieve specific files from the 4DOS library. Choose the full installation to put all of 4DOS on your system, or the Tour option if you want to see what 4DOS can do before you install it permanently. The installation program will not make any changes to your *CONFIG.SYS* or *AUTOEXEC.BAT* files unless you give it permission to do so.

If you elect to perform a full installation, reboot your computer when the installation program is done. You will then have all the power of 4DOS available to you.

We know some users feel unsure about running automated installation programs on their computers. 4DOS's automated installation is carefully written to be very well-behaved. It won't modify or erase any existing files without asking you, and it takes a very straightforward, step by step approach.

You can halt the installation process at any time and return to the DOS prompt by pressing **Ctrl-X** (hold down the Ctrl key and then press "X").

❖ Manual Installation

The 4DOS files are contained in a special library file on the distribution diskette. You cannot simply copy files from the diskette onto your system. You **must** use the installation program to extract and decompress the 4DOS files if you want to perform a manual installation, or if you need to replace a damaged 4DOS file on your hard disk.

If you want to install 4DOS manually, first start the automatic installation program using the instructions above. Select the **Extract all files** option and extract the 4DOS files onto your hard disk or another floppy disk. On a hard disk, place the files in their own directory. If you are upgrading from a previous version of 4DOS, use a new directory -- don't overwrite your existing files.

Once you've extracted the files, you can go through the 4DOS Tour (see page 19) if you want to try 4DOS before completing the installation. When you're ready to finish the installation process, all you need to do is add one

line to your *CONFIG.SYS* file (before modifying *CONFIG.SYS*, be sure you have a bootable floppy disk as discussed above):

```
SHELL=D:\PATH\4DOS.COM D:\PATH /P
```

"d:\path" means the drive and directory where your 4DOS files are stored. The second "d:\path" on the SHELL= line sets the COMSPEC environment variable, and can be left out if *4DOS.COM* is in the root directory of your boot drive. Be sure to delete or REMark out any old SHELL= line for *COMMAND.COM* after you add the new SHELL= line for 4DOS.

Next, add the following line to your *AUTOEXEC.BAT* file:

```
D:\PATH\KSTACK.COM
```

where "d:\path" is the drive and directory where your 4DOS files are stored.

Once you've finished modifying *CONFIG.SYS* and *AUTOEXEC.BAT*, reboot your system to start 4DOS. For details on setting up the SHELL= line and *AUTOEXEC.BAT*, and on the *4DOS.INI* file, which controls 4DOS configuration, see Chapter 6 / Options and Tuning (page 107).

❖ Uninstalling 4DOS

In the extremely unlikely event that you have trouble booting your computer after you install 4DOS, you can also remove it quite easily. We don't expect you to have any trouble, but we know some people feel more comfortable knowing how to uninstall a product as well as install it. Or, you may need to remove 4DOS from one system if you are moving it to another system.

To temporarily remove 4DOS from your system, first find the location of *COMMAND.COM* on your disk (for example, in the root directory, or the DOS directory). Boot the system and use any standard editor to edit your *CONFIG.SYS* file (before modifying *CONFIG.SYS* be sure you have a bootable floppy disk as discussed above). Look for a line which begins:

```
SHELL=D:\PATH\4DOS.COM ...
```

("d:\path" means a drive letter and directory name). Insert the characters "REM " at the beginning of this line. This converts it into "remarks" or

comments. (If you are using DOS 3 or below, REMarks are not recognized in *CONFIG.SYS*, so the changed lines will produce a harmless "Unrecognized command" error when the system boots.) Next, add a new line like this:

```
SHELL=D:\PATH\COMMAND.COM /P
```

where "d:\path" is the proper drive and directory for *COMMAND.COM*. If you were previously running *COMMAND.COM* with a /E:nnnn switch to set the size of your environment, you can add it to this line as well.

After *CONFIG.SYS* has been modified, edit your *AUTOEXEC.BAT* file to remove any changes made to accommodate 4DOS. Look for two commands:

```
SET COMSPEC=D:\PATH\4DOS.COM
D:\PATH\KSTACK.COM
```

The first command will not be present on most systems. If it's there, change it to read:

```
SET COMSPEC=D:\PATH\COMMAND.COM
```

where "d:\path" is replaced by the correct drive and directory for *COMMAND.COM*. The second command can be deleted, or you can place "REM " in front of it to convert it to a comment. The second command may be present without the "d:\path" portion or the ".COM" portion; the key thing to look for is "KSTACK". If you wish, you can look for the PATH command, and remove the 4DOS directory from the directories listed there. However, there's probably no reason to do so unless you're permanently removing 4DOS from your system.

Now reboot your system, and you should be back up and running under *COMMAND.COM*. Correct the problem that gave you trouble with 4DOS (contact our technical support department if you need help, see page 7). Once the problem is fixed, edit your *CONFIG.SYS* file again to remove the REMs on your 4DOS lines and put one on your *COMMAND.COM* line instead, restore any COMSPEC command in *AUTOEXEC.BAT*, and you can boot with 4DOS again.

To completely remove 4DOS from your system, change *CONFIG.SYS* and *AUTOEXEC.BAT* as described above, then delete your 4DOS files.

Again, we don't expect you to need to use the procedure above -- but we thought you should have it anyway, just in case it makes you feel a little more at ease about installing a new product on your computer.

CHAPTER 3 / A GUIDED TOUR OF 4DOS

This chapter will acquaint you with some of the features of 4DOS. It isn't a substitute for the rest of the manual, but it will help you understand how familiar 4DOS seems if you are used to working at the traditional command prompt. At the same time, it will introduce you to many of 4DOS's most popular features and enhancements. By the time you finish working through this chapter, you will have a feeling for how easy and friendly 4DOS is compared to the traditional DOS command processor.

This tour is designed to be used interactively. Sit down at your computer with the manual. Each time an example is shown, try it. The tour will create all of the files it uses; it won't modify or change your existing files and subdirectories at all. Each section of the tour is self-contained so you can skip any sections you wish and return to them later.

If you come across terms or concepts in this chapter that you are unsure about, refer to Chapter 4 / General Concepts, the Glossary on page 325, or the Index.

Starting The Tour

Before you can start the tour, you need to install 4DOS. If you haven't done so yet, see page 13 for instructions. Select "Tour Installation" from the INSTALL menu to copy the necessary 4DOS files to your disk without making any modifications to your *CONFIG.SYS* or *AUTOEXEC.BAT* files. If you have already done a full installation and 4DOS is running on your system, just skip the step below that actually starts 4DOS (the step where the command "4DOS" is entered).

During the course of the tour, we'll ask you to enter several commands. The text that you should enter is always shown in **bold** type. Your entries are shown here in lower case, but you can type in either upper or lower case.

The computer's prompts and responses are displayed in normal, non-bold type. The display may look slightly different on your system, depending on how your prompt is set up, what disk drive you're using, and what files are in your 4DOS directory. Don't worry about any minor differences.

In some cases, the computer's output will be too long to fit on a line of this manual and remain readable. We've truncated those lines and placed three dots [...] at the end to show you that the actual line displayed on your screen will be a little longer. Some commands create more lines of output than we need to display here to show how they work. We've put a line of nine dots [... ... ...] in those locations to indicate that additional information will be shown on your screen.

To start the tour, first change to the directory where 4DOS was installed, using the CD command. For example, if you've installed 4DOS in the directory *C:\4DOS*, enter the following command:

```
C:\> cd 4dos
C:\4DOS>
```

Now start 4DOS by typing:

```
C:\4DOS> 4dos
```

You'll see a sign-on message from 4DOS that looks similar to this:

```
4DOS EMS swapping initialized (96K)
4DOS 4.0   DOS 5.0
Copyright 1988-1991  Rex Conn & JP Software Inc.  ...
Registered for use on a single computer.
```

and then a prompt like this:

```
c:\4dos>
```

Depending on how your prompt was previously defined, you may see the 4DOS prompt in lower case, which many people find easier to read. A lower case prompt is the 4DOS default, and we'll show the prompt that way throughout the tour. Once you have 4DOS installed you can use the PROMPT command (see page 259) to change the prompt to upper case if you prefer.

If you want to exit from 4DOS at any time, just type the command EXIT at the 4DOS prompt and press Enter. This will return you to the traditional DOS command line:

```
c:\4dos> exit
C:\4DOS>
```

Basic Commands

We'll begin the tour by demonstrating some of the most common, and most familiar, 4DOS commands. We'll also show you some of the enhancements that 4DOS has added to those commands to make your computing easier.

First, enter the single-character command **?**. You'll see a display like this:

```
c:\4dos> ?
?             ALIAS         ATTRIB        BEEP          ...
CANCEL        CD            CDD           CHCP          ...
COLOR         COPY          CTTY          DATE          ...
DESCRIBE      DIR           DIRS          DRAWBOX       ...
 ... ... ...
UNALIAS       UNSET         VER           VERIFY        ...
Y
```

The **?** command displays a list of all of 4DOS's commands. You certainly don't have to memorize them all -- we'll show you how to get help with any command in a few moments. Nor are we going to demonstrate all of these commands in this tour. If you want complete information about a command, turn to the alphabetic Command Reference that begins on page 157.

Now try a DIR command to see a list of files in the current directory. DIR displays a list of file names, sizes, dates, and times:

```
c:\4dos> dir

 Volume in drive C is JPS_TEST    Serial number is ...
 Directory of  c:\4dos\*.*

.             <DIR>       8-22-91   14:21
..            <DIR>       8-22-91   14:21
4dos.com         99280    8-26-91    4:00
4dos.doc        345005    8-26-91    4:00
4dos.ico           766    8-26-91    4:00
4dos.pif           545    8-26-91    4:00
... ... ...
tour2.btm         2765    8-26-91    4:00
update40.doc     37946    8-26-91    4:00
vendor.doc        4434    8-26-91    4:00
     833,056 bytes in 21 file(s)  851,968 bytes allocated
  18,651,136 bytes free
```

The 4DOS DIR display should look familiar. But, unlike the traditional DIR display, 4DOS shows file names in lower case and in alphabetical

order. 4DOS also gives you some totals at the end of the display that help you see how much space your files are using.

By using some of DIR's options, you can make the display easier to read. The directory display you just saw probably didn't fit on your screen. You can tell DIR to pause at the end of each page by using the command DIR /P. The **/P** is an example of a **switch** or **option** which modifies the behavior of a command:

```
c:\4dos> dir /p

 Volume in drive C is JPS_TEST     Serial number is ...
 Directory of  c:\4dos\*.*

.             <DIR>       8-22-91   14:21
..            <DIR>       8-22-91   14:21
4dos.com         99280    8-26-91    4:00
4dos.doc        345005    8-26-91    4:00
4dos.ico           766    8-26-91    4:00
4dos.pif           545    8-26-91    4:00
... ... ...
tour2.btm         2765    8-26-91    4:00
update40.doc     37946    8-26-91    4:00
Press any key when ready...
vendor.doc        4434    8-26-91    4:00
     833,056 bytes in 21 file(s)   851,968 bytes allocated
  18,649,088 bytes free
```

You might prefer to display directories in 2 columns. DIR will do that if you include the **/2** option. If you add the **/V** option, it will perform a vertical sort, with file names running alphabetically down the first column and then down the second column. (We've left off the end of the second column of the display, since it doesn't fit on a manual page. You'll be able to see it on the screen.)

```
c:\4dos> dir /2/v

 Volume in drive C is JPS_TEST     Serial number is ...
 Directory of  c:\4dos\*.*

.             <DIR>       8-22-91   14:21   helpcfg.exe  ...
..            <DIR>       8-22-91   14:21   keystack.sys ...
4dos.com         99280    8-26-91    4:00   order.frm    ...
4dos.doc        345005    8-26-91    4:00   readme.doc   ...
4dos.ico           766    8-26-91    4:00   support.bbs  ...
4dos.pif           545    8-26-91    4:00   sysop.doc    ...
4dosm.ico          766    8-26-91    4:00   tour1.btm    ...
4help.exe        45632    8-26-91    4:00   tour2.btm    ...
aliases           8164    8-26-91    4:00   update40.doc ...
```

```
appnotes.doc     82722   8-26-91    4:00   vendor.doc    ...
dos.hlp         120555   8-26-91    4:00
     833,056 bytes in 21 file(s)  851,968 bytes allocated
  18,653,184 bytes free
```

DIR has many other formatting and file selection options. You'll use a few below; all of them are explained with the DIR command on page 198.

Next, you'll use a simple batch file called *TOUR1.BTM* to create a dummy file. (A *.BTM* batch file is similar to a traditional *.BAT* batch file but faster. See page 97 for an explanation of the differences between the two.) To run *TOUR1.BTM*, enter the command:

```
c:\4dos> tour1
```

This batch file creates a small file called *FILE1* in your current directory. The contents of the file aren't important; we're simply using it to demonstrate some of 4DOS's file-handling capabilities. You can verify that *FILE1* has been created by using DIR again.

Now use the COPY command to copy the contents of *FILE1* to a new file, *FILE2*:

```
c:\4dos> copy file1 file2
c:\4dos\file1 => c:\4dos\file2
     1 file(s) copied
```

4DOS performed the copy just like the traditional COPY command does. You may notice that the output is a little friendlier: COPY tells you exactly what file it copied and where it copied the file to, along with a count of files at the end.

Now try renaming a file. If you've used the traditional RENAME command or its synonym REN, this will look familiar:

```
c:\4dos> ren file1 file3
c:\4dos\file1 -> c:\4dos\file3
     1 file(s) renamed
```

Like COPY, the 4DOS REN command tells you just what it did. You now have two files, *FILE2* and *FILE3*, in the current directory. You can use one of 4DOS's enhancements to add the extension *.TST* to both of them:

```
c:\4dos> ren file2 file3 *.tst
c:\4dos\file2 -> c:\4dos\file2.tst
c:\4dos\file3 -> c:\4dos\file3.tst
     2 file(s) renamed
```

Unlike the traditional REN command, 4DOS lets you rename multiple files with a single command. All of the 4DOS file processing commands like COPY, DEL, MOVE, and RENAME accept multiple file names, so you can do in one command what used to require a separate command for each file.

Now delete the files you have just created. You could use a simple command like DEL *.TST, but that would delete any other *.TST* files in the current directory also. To protect against erasing files that you might want to keep, add a **/P** option to DEL so that it will prompt you before it deletes each file. Answer Y to the prompts shown below to let 4DOS delete your test files:

```
c:\4dos> del *.tst /p
Delete c:\4dos\file3.tst (Y/N)? y
Delete c:\4dos\file2.tst (Y/N)? y
     2 file(s) deleted
```

To verify all these actions, you can do a DIR or DIR /P to look at what files are left in the directory. All the files you've just created (*FILE1*, *FILE2*, *FILE3*, etc.) should be gone.

The features we've demonstrated here -- traditional commands with enhancements that make your work easier -- are present throughout 4DOS. If you want more details, pick a DOS command that you're familiar with and look up the corresponding 4DOS command in the reference section to get an idea of what's new and improved in 4DOS. (Remember that 4DOS only replaces internal commands like COPY and DIR, not external commands like DISKCOPY and FORMAT.)

Command Line Editing and History

This section demonstrates 4DOS features which make it easy to correct typing mistakes at the DOS prompt, to repeat previous commands, and to get help with any command.

We can't show you exact examples here as easily as we can in the other sections of the tour, since the display depends on the exact keystrokes you

type and we want you to experiment a little. So we'll guide you through what 4DOS can do, and you'll see the results on your screen.

First, create a typing error by mistake. Enter an incorrect DIR command:

```
c:\4dos> dur /2
```

Traditionally, you would have had to use the Backspace key to erase most of the line in order to correct this error, or press Esc and start over. 4DOS makes corrections much easier. Press the Home key followed by the right arrow. The cursor will move to the start of the line and then right one space to the "u" in "dur". Type an "i" and the command should be correct. To execute it, press Enter; if you'd rather not, press Esc.

That's a simple example of 4DOS's command line editing. You can use the left and right arrow keys, Home and End, and Backspace and Del to move around and modify your command line at any time, just like you do in your word processor or editor. Other keys let you move the cursor a word at a time, delete words, change between overstrike and insert modes, and perform many other operations. Esc always clears the line and lets you start over. For a summary of all the command line editing keys see page 56, or your 4DOS Quick Reference card.

4DOS also remembers the commands you type in a **command history**. To see the commands you have used so far during this tour, we'll use the 4DOS HISTORY command to display the entire history list:

```
c:\4dos> history
?
dir
dir /p
dir /2/v
tour1
ren file1 file3
ren file2 file3 *.tst
del *.tst /p
dir /2
```

4DOS records about 1000 characters of history. You can reduce or enlarge this amount to suit your own needs.

Now press the up-arrow key once. The last command in the HISTORY display (DIR /2 in the example above) will appear at the prompt. You can use the editing keys to modify this command, which is much easier than

retyping it. To execute the command again, in either its modified or unmodified form, press Enter. To clear the line, press Esc.

If you'd like to see more commands, first press Esc to clear the command line. (To see all commands in the history, you should always start with an empty command line.) Now press up-arrow several times. Each time you press it, 4DOS will back up one more line in the command history. Once you find the line you're looking for, you can modify it if you like and then press Enter to execute it.

Now imagine that you have been working for a while. An hour ago you did a complex DIR command and you need to do it again. You could scroll through an hour's worth of command history. But another 4DOS feature called **command completion** will save you time.

First, type the beginning of the command -- DI, or DIR. Now press the up-arrow. 4DOS will recall the newest command in the history that starts with the characters you typed (if there are no matches, 4DOS beeps). Press up-arrow again to retrieve the command before that, still matching to the characters you originally typed.

You can test this out easily using DI or DIR, because you've put several DIR commands into the command history during the tour.

Perhaps you would prefer to look at a list of commands that you have entered and choose from the list. First press the Esc key to clear the command prompt. Then press the PgUp key. 4DOS will display a **history window** in the upper right corner of your screen showing the commands you've recently typed. You can move around in this window with the up-arrow and down-arrow keys. PgUp and PgDn move the display a page at a time. When you find the command you want, press Enter to re-execute it as is, or press any other key to move the line down to the prompt where you can edit it as usual.

You can find more information on command line editing and history beginning on pages 56 and 57. And you can use directives in your *4DOS.INI* file (see page 116) to set the length of the history list and the size of the history window, and to redefine the editing and history keys.

You can also get help any time you need it from the 4DOS command prompt. Just press the F1 key. You'll see a display of all of 4DOS's HELP topics. If you would like some help with COPY, move the cursor bar to

COPY, or just type COPY and 4DOS will move the bar for you. Press Enter to see help on the topic; once you're there, scroll through the help using the vertical arrows or PgUp and PgDn keys. The highlighted items within a topic are cross-references to other topics; select one with the horizontal arrow keys and press Enter to view it. Press Esc or F1 to return to the topic list.

For more help using the Help display, select the topic -HELP- from the topic list. If you have a mouse, see the -MOUSE- topic for information on using the mouse within the Help system.

Now exit the help system (use Esc) and start entering a command:

```
c:\4dos> copy *.doc a: /
```

After that "/" you want to use the option that tells COPY to prompt you before replacing an existing file, but you can't remember what it is. Just leave the command as it is and press F1. 4DOS will display help on the command you're entering, and you'll see that the option you want is **/R**. Press Esc to exit help and you'll be right back on the command line. You can type the R and press Enter to execute your command, or press Esc instead to clear the line if you don't really want to execute the command.

4DOS has may more features that can help you at the command line. Most of them are described in Chapter 5 / Using 4DOS, in the section **At the Command Line** starting on page 55.

More About Files

This section of the tour lets you explore some more of 4DOS's features that will help you manage files and directories. You're about to use another batch file to create some files with which you can experiment. The batch file you'll use is called *TOUR2.BTM*. First, take a look at the batch file with the 4DOS LIST command:

```
c:\4dos> list tour2.btm
```

As you can see on your screen, LIST is a full screen file viewer. You can scroll and page through the text. You may notice that some lines near the end of the file extend beyond the edge of your screen. To make the off-screen text visible use the left and right arrows to scroll the display horizontally, or press **W** to turn line wrapping on and off. LIST also lets

you search for text and print the file you're viewing, two capabilities that we won't demonstrate on this tour. The file *TOUR2.BTM* is simply a batch file that creates other files, using 4DOS's ECHO command. You don't have to worry about how it works; we just used it to show you LIST.

Now run *TOUR2.BTM*:

```
c:\4dos> tour2

Please wait ...

File creation completed
```

TOUR2 creates three files called *FILE1*, *FILE2*, and *FILE3*. The contents of the files aren't important since we're just using them for demonstration purposes. You'll be deleting and recreating them several times through the rest of the tour.

Now try 4DOS's SELECT feature. SELECT lets you choose files for any 4DOS command from a full-screen list. We'll use it here to delete one of the files that TOUR2 created. Enter the command:

```
c:\4dos> select del (file*)
```

This command tells 4DOS to let you select from files that begin with the characters "file", and then pass the name of each file you select to the DEL command for action.

The display on your screen should include a two-line header and then the list of files, just as they'd look in a directory display. Press the spacebar to "mark" the first file -- a triangular mark will appear to the left of the file name. You can scroll around in the display and mark and unmark files with the spacebar as you like. When you hit Enter, the command will be executed, deleting the files you've marked. For now, mark *FILE1* and leave the other files unmarked, so that *FILE2* and *FILE3* are not deleted. You'll use them in the next step.

SELECT is also handy if you want to copy a group of files to a floppy disk, perhaps to take your work home for the night or to make a quick backup. If you'd like to try it, put a blank, formatted floppy disk in drive A. Then enter this command:

```
c:\4dos> select copy (file*) a:
```

Mark one or both of the files in the SELECT display, and the marked file(s) will be copied to the floppy disk when you hit Enter.

SELECT is a "prefix" command: it goes before another command and modifies what that second command does. Another useful prefix command is EXCEPT, which lets you do something *except* to one or more files. Before you try EXCEPT, create new copies of *FILE1*, *FILE2*, and *FILE3* (TOUR2 will overwrite any old copies of these files remaining from the last time it was run):

```
c:\4dos> tour2

Please wait ...

File creation completed
```

Now use EXCEPT to delete all but one of the files that TOUR2 created:

```
c:\4dos> except (file1) del file*
Deleting c:\4dos\file2
Deleting c:\4dos\file3
     2 file(s) deleted
```

The EXCEPT command protected *FILE1* from being deleted. To verify that, use the DIR command. When you're done, run TOUR2 one more time so the files are there for the next step.

Next, create two subdirectories within the current directory:

```
c:\4dos> md dira dirb
```

Notice that you can create both subdirectories with a single command. Traditionally, you would have needed two MD commands to do the same thing. To verify that the directories are there, use DIR but ask it to display only subdirectories and not files (if you're curious, **/A:D** stands for "Attributes: Directory"; see page 198 for more details):

```
c:\4dos> dir /a:d

 Volume in drive C is JPS_TEST  Serial number is ...
 Directory of  c:\4dos\*.*

.            <DIR>       8-22-91  14:21
..           <DIR>       8-22-91  14:21
DIRA         <DIR>       8-27-91  10:23
DIRB         <DIR>       8-27-91  10:23
```

```
           0 bytes in 4 file(s)          0 bytes allocated
  18,608,128 bytes free
```

OK, things look right. Now move the demonstration files to those directories. 4DOS has a built-in MOVE command to move a file from one directory or drive to another, something DOS users have wished for for a long time.

Here are the commands to move *FILE1* to *DIRA*, and *FILE2* and *FILE3* to *DIRB*:

```
c:\4dos> move file1 dira
c:\4dos\file1 -> c:\4dos\dira\file1
     1 file(s) moved

c:\4dos> move file2 file3 dirb
c:\4dos\file2 -> c:\4dos\dirb\file2
c:\4dos\file3 -> c:\4dos\dirb\file3
     2 file(s) moved
```

As usual, 4DOS tells you exactly what it's doing.

Now that you've created a subdirectory structure, it's time to get a comprehensive look at it. Use DIR to look for all the files whose names begin with "file" in the current directory and all of its subdirectories -- that's the DIR /S option:

```
c:\4dos> dir /s file*

 Volume in drive C is JPS_TEST    Serial number is ...
 Directory of  c:\4dos\dira\file*.*

file1              22   8-27-91  10:37
          22 bytes in 1 file(s)      2,048 bytes allocated
  18,587,648 bytes free

    Total for:  c:\4dos\dira\file*.*
          22 bytes in 1 file(s)      2,048 bytes allocated

 Directory of  c:\4dos\dirb\file*.*

file2              22   8-27-91  10:37
file3            1938   8-27-91  10:37
       1,960 bytes in 2 file(s)      4,096 bytes allocated
  18,587,648 bytes free

    Total for:  c:\4dos\dirb\file*.*
       1,960 bytes in 2 file(s)      4,096 bytes allocated
```

```
Total for:  c:\4dos\file*.*
   1,982 bytes in 3 file(s)     6,144 bytes allocated
```

DIR has displayed the directory header, filenames, and totals for each of the two subdirectories that contain files matching the name you entered, FILE*. It also has displayed a grand total.

You've seen that DIR can look at several subdirectories at once. Now do the same thing with DEL, and delete the files you put in your demonstration subdirectories, along with the subdirectories themselves, with a single command. To do so, you need to use two options: /S and /X. The first tells DEL to delete files in the current directory and all of its subdirectories. The /X option makes DEL remove each subdirectory if all the files within it are deleted:

```
c:\4dos> del /s/x dira dirb
c:\4dos\dira\*.* : Are you sure (Y/N)? Y
Deleting c:\4dos\dira\file1
c:\4dos\dirb\*.* : Are you sure (Y/N)? Y
Deleting c:\4dos\dirb\file2
Deleting c:\4dos\dirb\file3
     3 file(s) deleted
```

Note the safety feature that 4DOS gave you here. When you tell DEL to delete DIRA and DIRB, 4DOS recognizes them as directory names and assumes you mean "delete all files in this directory". Since you're deleting all the files, 4DOS displays a prompt that tells you what's about to be deleted and asks you whether you really want to do the deletion. Once you answer Y, the files are deleted and the subdirectory is automatically removed because you used the /X switch.

4DOS also lets you use **file descriptions**, so you can describe a file's contents more clearly than with an 8-character file name. The descriptions can be up to 40 characters long.

First, run TOUR2 again to regenerate the three demonstration files. Then enter a DESCRIBE command, along with a description for each file (you can enter any description you like; you don't have to use the text shown):

```
c:\4dos> tour2
c:\4dos> describe file*
Describe "c:\4dos\file1" : Tour file 1
Describe "c:\4dos\file2" : Tour file 2
Describe "c:\4dos\file3" : Tour file 3
```

Now look at the descriptions with a DIR command:

```
c:\4dos> dir file*

 Volume in drive C is JPS_TEST     Serial number is ...
 Directory of  c:\4dos\file*.*

file1              22   8-27-91  10:48 Tour file 1
file2              22   8-27-91  10:48 Tour file 2
file3            1938   8-27-91  10:49 Tour file 3
       1,982 bytes in 3 file(s)      6,144 bytes allocated
  18,604,032 bytes free
```

The descriptions will appear any time you ask for a standard, single-column directory display. They will also appear when you use the SELECT command. They can be a lifesaver when you have files whose contents you can't remember, or when you have large groups of files with similar names.

Directory Navigation

4DOS doesn't just make it easier to access files; it also makes your life much easier when you're navigating through the hard disk directory structure. You're probably already familiar with the traditional CD command, which you use to change directories. 4DOS adds a "go back" option to CD, which is invoked by using the minus sign [-] instead of a directory name. Try this:

```
c:\4dos> cd \
c:\> cd -
c:\4dos>
```

The CD - changes back to the directory you were in before the most recent CD command. It's a convenient way to switch back and forth between two directories.

4DOS also lets you change the drive and directory at the same time with the CDD command so you don't have to switch drives first and then change directories. Here's an example using CDD. Before you try it, put a floppy disk in drive A:

```
c:\4dos> cdd a:\
a:\> cdd -
c:\4dos>
```

As you can see, the minus works with CDD as well.

For more complex sequences of directory navigation, you can use PUSHD and POPD. These commands maintain a directory "stack" and let you make several changes, then move back through the directories you've been to. They can change both drive and directory, like CDD. For example:

```
c:\4dos> pushd a:\
a:\> pushd c:\
c:\> popd
a:\> popd
c:\4dos>
```

4DOS also offers you a special environment variable, CDPATH, to help you find the right directory without a lot of typing. CD, CDD, and PUSHD use CDPATH to find the directory you want to change to if they can't find it in the current directory. This can help a lot when you have long but commonly used directory names. For example, say you have a directory called *C:\DBASE\REPORTS* which contains a subdirectory for each month of the year. If you set CDPATH like this:

```
c:\4dos> set cdpath=c:\dbase\reports
```

Then you can change to the JANUARY subdirectory like this:

```
c:\4dos> cd january
c:\dbase\reports\january> cd -
c:\4dos>
```

CD found the JANUARY subdirectory, saving you from typing the entire name, because its parent directory was listed in CDPATH. For more details about CDPATH see page 114.

Aliases

Aliases are one of 4DOS's most powerful features. Simple aliases are very easy to set up and use (that's what we'll discuss here). Complex aliases allow you to configure your system just about any way you want, and can take the place of many small batch files.

The purpose of aliases is to rename or reconfigure 4DOS commands. They are defined and viewed with the ALIAS command. In this tour, we'll show you how to set up aliases for the DIR command. Of course, you can use aliases to enhance any command; for more examples see the ALIAS command on page 164, and the sample file ALIASES that comes with 4DOS.

Here's a popular favorite for anyone who uses DIR and wants to be able to use a simple D instead:

```
c:\4dos> alias d = dir
```

To see what aliases you've defined, enter ALIAS with no parameters:

```
c:\4dos> alias
d=dir
```

To use the alias, just enter its name at the prompt, like any command:

```
c:\4dos> d

 Volume in drive C is JPS_TEST     Serial number is ...
 Directory of  c:\4dos\*.*

.             <DIR>       8-22-91  14:21
..            <DIR>       8-22-91  14:21
4dos.com         99280    8-26-91   4:00
4dos.doc        345005    8-26-91   4:00
4dos.ico           766    8-26-91   4:00
4dos.pif           545    8-26-91   4:00
... ... ...
tour2.btm         2765    8-26-91   4:00
update40.doc     37946    8-26-91   4:00
vendor.doc        4434    8-26-91   4:00
     835,038 bytes in 24 file(s)  858,112 bytes allocated
  18,599,936 bytes free
```

Once you've defined an alias, you can use it anywhere a command can be used: at the command prompt, in a batch file, or inside another alias.

As an example, build on the D alias you have already defined. Suppose you regularly use 4DOS's 2-column directory listings, discussed near the start of the tour. Each time you want a directory listing, you use the command DIR /2. Define an alias, D2, to let you execute this command with the touch of a couple of keys. Since you have already defined D, you can define D2 in either of these ways (choose one to try):

```
c:\4dos> alias d2 = dir /2
c:\4dos> alias d2 = d /2
```

If you want to see the results, just enter ALIAS again:

```
c:\4dos> alias
d=dir
d2=d /2
```

To use this alias, just enter the command D2 and press Enter. You can pass file names to these aliases just like you would to the original command. For example, D2 *.DOC will give you a 2-column display of the *.DOC* files.

You can create aliases that are even easier to use with 4DOS's **keystroke aliases**. These aliases let you assign an Alt or Function key to an alias so you can invoke it by pressing a single key. Say you'd like to assign the 2-column directory display to F5. Just define an alias like D2 above, and make the alias name the key name, with an at-sign [@] before it:

```
c:\4dos> alias @f5 = dir /2
```

Now press F5 and the DIR /2 command will appear on the command line. Press Enter to execute it, or type some additional arguments and then press Enter if you'd like. (You can also make key aliases execute as soon as you press the key. For details on how, see the ALIAS command on page 164.)

You can use an alias to redefine how a standard command works, without changing its name. Suppose you always want DIR to display its output in 2 columns, with a vertical sort and a pause at the end of each page. You might think of using something like this:

```
c:\4dos> alias dir = dir /2/p/v
```

Go ahead and try that, then do a DIR. You'll get an error:

```
c:\4dos> dir
Alias loop
c:\4dos>
```

The "alias loop" error is caused because 4DOS tries to reinterpret the DIR command inside your alias as another attempt to run the same alias. It's easy to change the alias so this doesn't happen:

```
c:\4dos> alias dir = *dir /2/p/v
```

The [*] tells 4DOS that what follows is not an alias. Try the definition above (you can scroll back to the incorrect definition with up-arrow and modify it). Then do a DIR and you'll see the results. Using this method, you can redefine the default options for any 4DOS command.

Aliases can contain multiple commands and can do much fancier things than what you've seen here. They're great for creating shorthand names for commonly used programs like your word processor or database, and they will often help get programs loaded faster as well -- if you put the full name of the program in an alias, 4DOS doesn't have to search your PATH for it.

For complete details, see the overview of aliases beginning on page 94, and the ALIAS command reference on page 164.

When you read about aliases in the rest of the manual, you'll notice that most alias definitions are shown with back-quotes [`] around the part after the name. Some aliases require these back-quotes when they are defined at the command line or in a batch file, to make it clear to 4DOS what is and isn't part of the alias.

We show aliases that way elsewhere because it's the safest way to enter them at the command line. If you always use the back-quotes, you don't have to worry about whether a particular alias needs them or not. We didn't use them here because none of these aliases require back quotes and we didn't want to add extra typing during the tour.

Other Commands

There are a few other commands that we'll just touch on, so that you can see some of 4DOS's other capabilities.

First, turn on the 4DOS **LOG** facility, which records all the commands you enter in a file. Enter the command:

```
c:\4dos> log /w mylog
```

You won't see anything else happen, but you've turned logging on. We'll return to the 4DOS log later.

4DOS has a couple of commands that let you control screen color. The examples here will work on any system with a color video board (the commands work on monochrome systems, too, but you're restricted to the colors white and black). Try clearing the screen to a specific color:

```
c:\4dos> cls bright white on magenta
```

Now let's set a different color:

```
c:\4dos> color bright yellow on blue
```

The behavior of the COLOR command varies, depending on whether you have an ANSI driver loaded. If you don't, COLOR will change the color of the entire screen immediately. If you do have an ANSI driver loaded, COLOR only affects the color of text displayed after it's executed, and not the color of text already on the screen. If you don't know, don't worry about it -- just experiment with COLOR and see.

The FREE and MEMORY commands help you keep track of system resources. FREE tells you about free space on your disk drives (and is much faster than CHKDSK). MEMORY tells you about memory resources, including expanded and extended memory and 4DOS's internal alias and history storage areas. Here are examples of the output from our test system; try the commands on your system and see what values you get.

```
c:\4dos> free
 Volume in drive C is JPS_TEST    Serial number is ...
  41,826,304 bytes total disk space
  23,232,512 bytes used
  18,593,792 bytes free

c:\4dos> memory
     655,360 bytes total DOS RAM
     612,256 bytes free

   7,815,168 bytes total EMS memory
     688,128 bytes free

      12,288 bytes free XMS memory  (HMA in use)

       1,792 bytes total environment
         233 bytes free

       6,144 bytes total alias
       1,045 bytes free

       1,024 bytes total history
```

The TIMER command lets you time events. The following line also shows 4DOS's ability to accept multiple commands on one line, separated by a caret [^]. It starts the timer, runs the *TOUR2.BTM* file to create the three demonstration files, deletes the three files, and then stops the timer and displays the time the whole operation took. Enter this command to time the entire sequence on your computer:

```
c:\4dos> timer ^ tour2 ^ del file1 file2 file3 ^ timer
Timer 1 on: 11:10:01

Please wait ...

File creation completed
Deleting c:\4dos\file1
Deleting c:\4dos\file2
Deleting c:\4dos\file3
     3 file(s) deleted
Timer 1 off: 11:10:06 elapsed: 0:00:05.11
```

Now return to the log that you started a few minutes ago. Turn logging off, then take a look at what was recorded. Enter the following two commands; the output will pause at the end of each page:

```
c:\4dos> log off
c:\4dos> type mylog /p
[ 8-22-91 11:05:02] cls bright white on magenta
[ 8-22-91 11:05:54] color bright yellow on blue
[ 8-22-91 11:07:08] free
[ 8-22-91 11:07:49] memory
[ 8-22-91 11:09:55] timer
[ 8-22-91 11:09:56] tour2
  ... (commands from TOUR2.BTM will be displayed here)
[ 8-22-91 11:10:00] del file1 file2 file3
[ 8-22-91 11:10:01] timer
[ 8-22-91 11:12:35] log off
```

As you can see, the log contains every command you entered plus a date and time stamp. It's a complete record of system activity, including commands you type and those entered from batch files and aliases. You can use it as a record of your work, for security purposes, or for anything else you desire. You may want to clean up the directory now by deleting this demonstration log with a DEL MYLOG command.

Batch Programming

This final section demonstrates a very little bit of what 4DOS can do for your batch files. If you've never worked with batch files, you may want to skip this section. If you aren't sure, give it a try and stop if things seem too complex. You don't have to be a batch file programmer to use 4DOS.

Rather than having you go to the trouble of writing actual batch files, we'll demonstrate some of the parts of 4DOS's batch file power that work just as well at the prompt.

Some 4DOS batch file improvements aid in communicating with the user. You can make sounds:

```
c:\4dos> beep 440 2 880 8 660 4
```

You can draw boxes and lines. Enter each of these commands on one line; use the second set of commands if you have a monochrome monitor:

```
c:\4dos> cls bright white on blue
c:\4dos> drawbox 10 10 20 70 4 bright cyan on black fill
          black
c:\4dos> drawhline 15 10 61 1 bright cyan on black

c:\4dos> cls bright white on black
c:\4dos> drawbox 10 10 20 70 4 black on white fill white
c:\4dos> drawhline 15 10 61 1 black on white
```

Notice that 4DOS correctly connects the line to the box where they intersect. 4DOS has additional commands like SCREEN and SCRPUT, which we won't demonstrate here, to display text anywhere on the screen and in any color.

4DOS also helps you ask the user for input. Try this (be sure to use two percent signs before the second "letter"):

```
c:\4dos> inkey Enter a letter: %%letter
Enter a letter: A
```

The letter you typed was stored in your environment in the variable named LETTER. Use the SET command to view it:

```
c:\4dos> set
COMSPEC=C:\4DOS\4DOS.COM
... ... ...
LETTER=A
```

The user can also type full strings if you use the INPUT command:

```
c:\4dos> input Enter a string: %%string
Enter a string: Type anything you like here ...
```

Again, SET will let you view the string, stored in the environment variable STRING.

Once you've collected some input, you will probably want to test it with the 4DOS IF and IFF commands. Here's one example. Enter this on one line (there's plenty of room; 4DOS command lines can be up to 255 characters

long). Note that two percent signs are used in the INKEY command and one is used in the IFF, and that a double equal sign is used in the IFF statement.

```
c:\4dos> inkey Enter a letter: %%letter ^
         iff "%letter"=="A" then ^ echo hi ^ else ^
         echo bye ^ endiff
```

Try using the up-arrow to repeat the command several times, giving different responses to the "Enter a letter" prompt.

4DOS offers dozens of additional batch file improvements. For more information see the section on batch files beginning on page 96, and the reference for each individual command.

Conclusion

This has been a very fast tour of some of the most popular features of 4DOS. There are many more features, commands and options to explore, as well as ways to customize 4DOS so that it suits your computing habits and needs. One of 4DOS's best features is its ability to adapt to your way of working instead of requiring you to adapt to it.

If you selected "Tour Installation" when you ran the 4DOS INSTALL program, you will probably want to perform a full installation now. You can put the 4DOS disk in your floppy drive, run INSTALL, and follow the instructions on the screen. If you need help, refer to the installation instructions on page 13.

To learn more about specific 4DOS commands, look through the Command Reference section of this manual, beginning on page 157. To learn more about the dozens of 4DOS features that aren't related to specific commands, read through Chapter 5 / Using 4DOS, beginning on page 55.

CHAPTER 4 / GENERAL CONCEPTS

You can start using 4DOS as soon as you finish installing it, because 4DOS is compatible with the traditional commands you're used to. But most users find that the more they know about their computer systems, the more power they get from 4DOS. And, the more experienced they become with 4DOS, the more they want to know about their computer system as a whole.

This section of the manual explains some fundamental concepts about your computer, DOS, and 4DOS. It should help you understand the terms and concepts in the pages that follow. If you find some of the concepts overwhelming, just remember that they are here when you need them. If you find this material too simple, skim over the topics and then go on to the next section. Each topic in this chapter is independent, so if you read it straight through you won't necessarily find a natural "flow" from one topic to another.

If you come across terms or concepts in this chapter that you are unsure about, refer to the Glossary on page 325 or the Index.

DOS and the Command Interpreter

When you turn on your computer, it first runs some internal diagnostics and then looks for a boot disk, either a floppy disk in drive A or your hard disk. There is nothing magical about a boot disk; it simply has a copy of DOS and a command interpreter available, plus a small block of special information that tells the computer that it is indeed a boot disk.

The core portion of DOS consists of two hidden files (files that are normally invisible to the DIR command). They aren't hidden to keep you from copying or examining them, but rather to keep you from accidentally erasing them. The names of the files depend on the version of DOS you are using, but they are usually something like *MSDOS.SYS* and *IO.SYS*. These two files contain the operating system, which controls your disk files and directories, loads and runs programs, maintains the system time and date, and performs other housekeeping tasks.

After the computer loads DOS into memory, it performs some standard initialization tasks and then looks for a file called *CONFIG.SYS*, which contains user-specified initialization commands. You can view and edit your *CONFIG.SYS* file with any text editor. One of the optional lines in

CONFIG.SYS begins with the word **SHELL**. That line names the command interpreter that DOS should load as it completes its initialization.

A command interpreter is a program that accepts your instructions and carries them out. The command interpreter shipped with DOS is called *COMMAND.COM*. Once you have 4DOS installed, *4DOS.COM* replaces *COMMAND.COM* as your command interpreter. Both *COMMAND.COM* and 4DOS are normal programs that know how to translate your commands into actions. 4DOS is simply a much more powerful command interpreter than *COMMAND.COM*.

Primary and Secondary Shells

Technically, the command interpreter is a **shell:** a program that understands your commands and makes the correct calls to DOS to perform various operations, including running programs.

The command interpreter that runs when the computer boots up is called the **primary shell**. Any command interpreter that is run by an application program with a "shell to DOS" feature, or that is run by a multitasking program like Windows or DESQView, is a **secondary shell**. 4DOS can be run as a primary shell and as a secondary shell.

A secondary shell has all the same features as a primary shell, but you can leave a secondary shell with the EXIT command. There is no way to exit from the primary shell, because your computer needs a shell present to operate. (You can exit from the primary shell in one special case: if you're running in the DOS compatibility box of OS/2 version 2.0.)

There are only a few differences between primary and secondary shells. Generally, less memory is available when a secondary shell is running, because at least part of the program that started it is still in memory waiting to spring back to life when you exit. And normally only the primary shell automatically executes the instructions in your *AUTOEXEC.BAT* file.

You may also see the term **shell** used to describe programs which assist you in managing your files (for example XTree or Lotus Magellan). This is a different and less precise meaning of "shell" than the one used above and elsewhere in this manual. Such programs are also sometimes called "visual shells" because they use a menu or graphical interface to receive

commands. Unlike 4DOS, these programs are not command interpreters, and cannot replace *COMMAND.COM*.

AUTOEXEC.BAT, 4START, and *4EXIT*

When a primary shell, either *COMMAND.COM* or 4DOS, gets control from DOS during boot up, one of its first responsibilities is to look for and run a batch file called *AUTOEXEC.BAT*, if that file is available in the root directory of the boot drive. This file is simply a list of commands that you want to have executed every time your computer boots up. If *COMMAND.COM* cannot find *AUTOEXEC.BAT*, it asks you for the time and date. 4DOS skips that step and immediately displays a standard prompt (*e.g.*, **c:\>**).

Every time 4DOS starts as either a primary or secondary shell, it also looks for an optional batch file called *4START.BTM* or *4START.BAT*, and runs the file if it finds it. 4START is a convenient place to put special 4DOS configuration commands.

Whenever you exit from a 4DOS secondary shell, 4DOS looks for another optional file called *4EXIT.BTM* or *4EXIT.BAT* and runs the file if it finds it. *4EXIT* is not necessary in most circumstances, but is a convenient place to put commands to save information from a secondary shell before it exits.

AUTOEXEC.BAT, *4START*, and *4EXIT* are called automatic batch files because they run without your intervention at specific times. *4START* and *4EXIT* should not be used to load any memory resident programs (TSRs). Otherwise, these three files can include any commands that could be part of any batch file or any commands which you could type from the command line.

For more details about batch files and batch file commands, see pages 96 and 158.

Command Processing

Whenever you type something at the command line and press the **Enter** key, you have given a command to 4DOS, which must figure out how to execute your command. If you understand the general process that 4DOS uses, you will be able to make the best use of the 4DOS commands.

4DOS begins by dividing the line you typed into an **action word** and a **command tail**. The action word is the first word in the command; the tail is everything that follows the command word. For example, in the command line

```
dir *.txt /p/o/n
```

the action word is "dir," and the command tail is "*.txt /p/o/n."

To decide what activity to perform, 4DOS makes five attempts to understand the action word:

First, 4DOS tries to match the action word against its internal list of aliases, which are synonyms that you have defined for commands. If it finds a match between the action word you typed and one of the aliases you've defined, it replaces the action word you typed with the action word from the alias. (This substitution is done internally, and is not normally visible to you). Once it has finished with the aliases, 4DOS continues with the steps listed below to identify the meaning of the new action word.

4DOS first tries to match the action word against its list of more than 80 internal commands, which are actions that are built into 4DOS. If it finds a match, 4DOS performs the internal command and then waits for a new instruction from you.

If there is no match with any of 4DOS's internal commands, it looks for an executable file (one with a *.COM* or *.EXE* extension) whose name matches the action word. It runs the executable file if it finds one. This is called an **external command** or **external program**.

Next, 4DOS looks for a batch file (with a *.BTM* or *.BAT* extension) that matches the action word. If it finds such a file, it then reads each line in the file as a new command.

Finally, 4DOS checks to see if the action word matches the name of a file with an extension that you have defined as executable. If a match is found, 4DOS runs the program you specified when you defined the executable extension. (Executable extensions are used to associate file extensions with the specific program that processes a particular type of file; see page 75 for details).

4DOS first tries the last three steps in the current directory. If the action word doesn't match a *.COM*, *.EXE*, *.BTM*, or *.BAT* file or an executable

extension in the current directory, 4DOS repeats its search in every directory in your PATH. The PATH is a list of directories that 4DOS (and some applications) search to look for executable files. If all these searches fail, 4DOS displays an "Unknown command" error message and waits for your next instruction.

If 4DOS finds a matching internal command in the first step, that command begins its activity by examining the command tail. Some commands require information in the command tail, some accept optional information, and others don't allow any command tail at all. The command tail for internal commands usually contains filenames, directory names, option switches which modify the command's behavior, or other information.

4DOS makes the command tail available to every alias, batch file, and external command that it executes. Aliases and batch files can examine the command tail by using "replaceable parameters," (see page 98). External commands examine the command tail in a manner determined by the programmer who wrote that particular external program.

It is up to each command to examine the command tail to make sure that the information it has received makes sense. If you use an option switch which the command doesn't understand, or if you omit a required piece of information, the command issues an error message and ends. Therefore, the command lines that you create must fit the format that each alias, internal or external command, or batch file expects.

The process that internal and external commands go through to separate the individual elements of the command line, make sure you used the correct syntax, and understand what you have requested, is called **parsing**.

Files and Paths

You may have dozens, hundreds, or thousands of files stored on your computer's disks. DOS 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.

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. (Systems with a single drive use both **A:** and **B:** for that drive. DOS asks you to swap diskettes as necessary to turn that single physical drive into two separate logical disk drives.)

Normally, drive **C:** is the first (or only) hard disk drive. Later versions of DOS and some utility programs can divide a large hard disk into multiple logical drives that are usually called **C:**, **D:**, **E:**, etc. Also, the DOS utility SUBST lets you use drive letters as a substitute for directory names. Many network systems (LANs) use a similar feature to give drive letters to sections of the network file server drives.

Some computers also have "RAM disks", which are areas of memory set aside by software (a "RAM disk driver") for use as fast, but temporary storage. RAM disks are also assigned drive letters, typically using letters beyond that used by the last hard disk in the system, but before any 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 RAM disk, and **G:** and **H:** as network drives.

Directories are used to divide the files on a drive 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.

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 naming rules as files (see below): a base name followed by an optional extension. However, some application programs do not properly handle subdirectory names that have an

extension. It is best to use only an 8 character (or less) name, without an extension, for subdirectories.

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 (excluding the file name and extension, but including the drive letter and colon).

DOS remembers 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, DOS 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:4DOS.DOC* refers to the file *4DOS.DOC* in the current 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. 4DOS allows you to use additional periods to specify directories further "up" the tree (see page 179).

Finally, each file has a **filename**. 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. You can use alphabetic and numeric characters plus the punctuation marks ! # $ % & ' ( ) - @ ^ _ ` { } and ~ in both the base name and the extension. Because the exclamation point [!], percent sign [%], caret [^], at-sign [@], parentheses [()], and back-quote [`] also have other meanings to 4DOS, it is best to avoid using them in filenames.

Each file also has **attributes** which define characteristics of the file which may be useful to DOS, to you, or to an application program. Attributes can

be set with 4DOS's ATTRIB command (see page 173) and viewed with the ATTRIB and DIR commands. Every time a program modifies a file, DOS sets the **Archive** attribute, which signals that the file has been modified since it was last backed up. This can be used by 4DOS 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; this can be used to protect important files from damage. The **Hidden** and **System** attributes prevent the file from appearing in normal directory listings. The 4DOS DIR command (see page 198) has options which allow you 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, DOS records the system time and date in a **time stamp** in the file's directory entry. Several 4DOS commands and variable functions, and many backup and utility programs, use this time stamp to determine the relative ages of files.

The Environment

The command interpreter keeps a list of information about your computer in memory. This list is called the **environment**. Every program receives a copy of the environment when it begins, and many programs use some of its information to configure themselves or to find files.

The environment is arranged as a series of **variables** and their related values. Each variable is a name stored in upper case. The name is followed by an equal sign [=] and some text. You can view the environment with the SET command, and add new entries or edit existing entries with SET and ESET. A typical environment entry looks like this:

LIB=c:\lib

In this example, the name of the variable is LIB and its value is "c:\lib."

The format and meaning of each entry in the environment is up to the program that uses the particular variable. Environment variables can contain just about anything, and can be used for any purpose the author of a program desires. The "purpose" of the environment as a whole is simply to hold small amounts of text which programs can then access according to

their own rules. Most environment variables are used by single programs for their own information; a few have well-defined meanings and are used by many different programs.

One of the most important environment variables is called PATH. The text of PATH is a list of subdirectories separated by semi-colons. 4DOS searches each of the subdirectories listed in the PATH entry to find executable files that aren't in the default directory. Many programs also search the PATH list to find their own files.

4DOS uses several environment variables to control its own behavior, and provides a wide range of facilities for manipulating and managing the environment. See page 113 for additional details.

Memory

The memory in your computer is organized in bytes. Normally, the amount of memory in a computer is discussed in terms of kilobytes (KBytes or 1,024 bytes) and megabytes (MBytes or 1,048,576 bytes or 1,024 KBytes). The amount of memory available in your computer is determined by the number of memory chips you have installed.

In an ideal world, there would be little more to say about memory. But because of the history of PCs, the needs of large application programs, and the capabilities of advanced CPUs, there are many different kinds of memory. The original 8088 CPUs of the PC and PC/XT can address 1 MByte of memory. Of that, a maximum of 640KBytes is allocated as **base**, **conventional**, **DOS**, or **low DOS** memory (all these terms mean the same thing). The other 384 KBytes, known as **upper memory**, are set aside for the computer's built-in ROM BIOS, video adapter cards, hard disk controllers, and other expansion hardware.

When base memory became too limiting, **expanded memory** (or **EMS memory**) was developed to give programs more data space. Expanded memory adds up to 16 MBytes which programs can access, 64KBytes at a time, through a window in upper memory. In 8088 / 8086 (PC and XT), and 80286 (AT) based computers, expanded memory typically requires an add-on board and support software. In 386 and 486 computers, expanded memory is typically provided without additional hardware, using the capabilities of the 386/486 chips.

The 80286 CPU used in the AT, and modern 386 and 486 CPUs, can use much more than the 8088's original 1 MByte of memory. An 80286 can use a total of 16 MBytes; the 386 and 486 can use up to a whopping 4,096 MBytes. This **extended memory** is not normally available to DOS-based programs, however, without special programming techniques and the help of DOS extenders or memory managers.

The memory terms used in this manual include

BASE memory: The 640 Kbytes or less that has traditionally been available for DOS and DOS-based applications.

EMS or LIM EMS Memory: Memory which conforms to the Expanded Memory Specification, developed by Lotus, Intel, and Microsoft, that lets programs and utilities share expanded memory.

Extended Memory: Memory beyond 1 MB in 80286, 386, and 486 computers. This memory may be accessed directly, in which case it is referred to as Extended Memory, or through XMS software, in which case it is referred to as XMS Memory (see below).

XMS Memory: Extended memory managed by software which conforms to the Extended Memory Specification (XMS). XMS lets programs share extended memory without conflict. This specification divides extended memory into extended memory blocks (EMBs). XMS software also usually manages the HMA and the UMBs (see below).

HMA: The first 64K bytes of extended memory, located just above 1 MB. Certain specialized programs such as DESQView, some network drivers, and portions of MS-DOS 5.0 and DR-DOS 5.0 can be loaded into the HMA instead of taking up valuable space in base memory.

UMBs: 386 and 486 computers can electronically move pieces of extended memory into unused space in the upper memory area between 640KB and 1 MB. Each block of this memory is called an Upper Memory Block (UMB). With DOS 5.0 or special 386/486 memory managers, memory-resident programs can be loaded into these UMBs instead of taking up valuable space in base memory. Some 8086, 8088, and 80286 systems can also use UMBs with

appropriate additional hardware and software (see page 136 for details).

ASCII and Key Codes

Internally, computers use numbers for everything. To represent the text that you type, a computer must translate each letter to and from a number. For all PC-compatible computers, the code used for this translation is called ASCII (American Standard Code for Information Interchange). ASCII codes are used both for the characters you type and for the characters that are displayed on the screen.

The original ASCII code has 128 values for upper and lower case letters, numerals, punctuation marks, and control codes. The control codes correspond to pressing the **Ctrl** key plus an alphabetic character. Some control codes are also represented on the keyboard with such keys as **Tab**, **Enter**, **Backspace**, and **Esc**. IBM, in its original PC, defined an additional 128 **extended ASCII** codes for math symbols, international characters, the line characters used to draw boxes, and some miscellaneous symbols. You can enter extended ASCII codes on the keyboard by holding down the **Alt** key while you type the code number on the numeric key pad.

Do not confuse **extended ASCII** with **extended key codes.** The latter include special codes that the computer generates when you press a function key, cursor key, or **Alt** plus a letter. Those keys do not have any representation in either the ASCII or extended ASCII code systems. Another set of key codes, called **scan codes**, is discussed in the section on the keyboard below.

Appendix B on page 311 has a complete list of ASCII, extended ASCII, and extended key codes.

The Keyboard

The original IBM PC, PC/XT, and virtually all XT-compatible computers use an 83-key keyboard with 10 function keys. The earliest PC/AT computer and compatibles added an 84th key, called SysReq, but left the rest of the keyboard the same. Most 80286, 386, and 486 computers now use an "enhanced" keyboard with 101 or more keys, including 12 or more function keys.

When you press a single key or a key combination, the computer translates the keystroke into two numbers. For all alphabetic, numeric, and punctuation keys, the **Tab**, **Enter**, **Backspace**, **Esc** keys and **Ctrl** plus an alphabetic key, these numbers are an ASCII code plus a scan code. The ASCII code represents the key's meaning; the scan code identifies which specific key was pressed. For example, many keyboards have two plus [+] keys, one above the equal sign and one on the numeric keypad. Both generate the same ASCII code, but they generate different scan codes.

Keys which are not represented by ASCII codes are translated to an ASCII 0 plus an **extended key code**. These keys include the function keys, the cursor keys, and **Alt** plus a key. The extended key code for a key is generally the same as the scan code for that key.

Some keys, like the **Alt**, **Ctrl**, and **Shift** by themselves or in combination with each other, plus the **Print Screen**, **SysReq**, **Scroll Lock**, **Pause**, **Break**, **Num Lock**, and **Caps Lock** do not have any code representations at all. The computer performs special actions automatically when you press these keys (for example, it switches your keyboard into Caps Lock mode when you press the Caps Lock key), and does not report the keystrokes to whatever program is running unless the program has been written specially to accept them.

It is up to the computer to smooth over the differences between the different keyboards. That is part of the reason why not all keyboards can be used with all computers.

Appendix B has a complete list of ASCII and scan codes of each of the keys on your keyboard.

Video

4DOS is a "character-based" program, which means that it works in text mode on your computer. In text mode, the screen displays text in a single font, but cannot mix fonts or display graphics. 4DOS can run graphics programs which change your screen to graphics mode, but the screen must be returned to text mode whenever 4DOS is active. In the early days of the IBM PC, text mode was a single, simple video operating environment. Today, advanced video boards and video software have created a wide range of text modes.

The original IBM PC monochrome and CGA color video cards can display 80 columns and 25 rows of text. Newer, advanced video systems normally run in this 80 x 25 display mode but can also display more columns and rows. For example, EGA video cards can display 43 rows of text and VGA video cards can display 50 rows. With special driver programs, a VGA can display 60 rows or more and up to 132 columns of text. Each of these different screen configurations is a different text mode.

4DOS supports whatever number of rows and columns of text you decide to use. It doesn't have commands to switch from one screen size to another -- you will need to use the software that came with your computer or video board to do that -- but it can read and work with the number of rows and columns on your screen.

ANSI Drivers

Every version of DOS includes a program called *ANSI.SYS*. This program lets you use text colors other than drab white on black, redefine keys, and control screen output. Commercial ANSI drivers are available as replacements for *ANSI.SYS*. The commercial programs usually include many new features, boost screen display speed dramatically, and support text displays that have more than 80 columns and 25 lines.

4DOS automatically determines whether you have an ANSI driver installed. If you do, 4DOS will use the driver to clear the screen and set screen colors.

Several 4DOS commands provide replacements for traditional *ANSI.SYS* commands. For example, 4DOS has commands to set the screen colors and display text in specific colors. These commands are easier to understand and use than traditional *ANSI.SYS* control sequences. Some of these commands manipulate screen colors directly. Others use an ANSI driver if one is installed, but save you the work of figuring out complex ANSI control sequences. Any special interaction between 4DOS commands and the *ANSI.SYS* driver is described in the documentation for each command.

CHAPTER 5 / USING 4DOS

4DOS is both a collection of commands and a set of features which make your computer easier to use. The commands are explained in the **Command Reference** section which begins on page 157. This section of the manual primarily explains each of the 4DOS features that are not directly related to individual commands.

Most of the features described in this section are easy to use, but a few are more technical in nature. Such features are marked with a ❖ next to the feature name or the paragraph which describes the feature's operation.

As you read through this section, we urge you to experiment with the features that catch your interest and pass over any which seem too complicated. Come back to this section as you gain expertise with 4DOS, and you will probably discover that the more complex features will seem easy and very useful. 4DOS doesn't require that you learn any more than you want, and even if you are a computer novice, you'll find some features that will interest you immediately.

If you come across terms or concepts in this chapter that you are unsure about, refer to Chapter 4 / General Concepts, the Glossary on page 325, or the Index.

At the Command Line

4DOS 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 4DOS's way of asking you to enter a command, an alias or batch file name, or the instructions necessary to begin an application program.

This section of the manual explains the 4DOS features that will help you while you are typing in commands, and how 4DOS interprets keystrokes entered at the command line. The keystrokes discussed here are the ones 4DOS normally uses. If you prefer using different keystrokes to perform these functions, you can reassign virtually all 4DOS keystrokes with keystroke directives in the *4DOS.INI* configuration file (see page 116).

Command Line Editing

The 4DOS command line works like a single-line word processor, allowing you to edit any part the command line at any time before you press Enter to execute the command, or Esc to erase it. 4DOS extends the command line to a maximum of 255 characters, and allows you to edit the command line even when it exceeds the width of your screen.

4DOS recognizes the following editing keys when you are typing a command (the words **Ctrl** and **Shift** mean to press the Ctrl [Control] or Shift key together with the other key named):

Cursor Movement:

Left arrow	Move the cursor left one character.
Right arrow	Move the cursor right one character.
Ctrl-Left arrow	Move the cursor left one word.
Ctrl-Right arrow	Move the cursor right one word.
Home	Move the cursor to the beginning of the line.
End	Move the cursor to the end of the line.

Insert and Delete:

Ins	Toggle between insert and overtype mode.
Del	Delete the character at the cursor.
Backspace	Delete the character to the left of the cursor.
Ctrl-L	Delete the word or partial word to the left of the cursor.
Ctrl-R or **Ctrl-Backspace**	Delete the word or partial word to the right of the cursor.
Ctrl-Home	Delete from the beginning of the line to the cursor.
Ctrl-End	Delete from the cursor to the end of the line.
Esc	Delete the entire line.
Ctrl-C or **Ctrl-Break**	Cancel the command.
Enter or **Return**	Execute the command line.

❖ Sometimes you may need to have 4DOS interpret a keystroke literally and place it on the command line instead of performing the usual action listed above. 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 4DOS prompt, because 4DOS would interpret it as a "Delete word right" command.

To get around this problem, use the special keystroke **Alt-255**. You enter Alt-255 by holding down the **Alt** key while you type **255** on the numeric keypad, then releasing the **Alt** key (you must use the number keys on the numeric pad; the row of keys at the top of your keyboard won't work). When 4DOS sees the Alt-255, it interprets the next keystroke literally and places it on the command line, ignoring any special meaning it would normally have as a 4DOS 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 *4DOS.INI* (see page 116), and Alt-255 itself can be reassigned with the CommandEscape directive.

Command History and Recall

Each time you execute a command, 4DOS saves the command line in a **command history list**. 4DOS lets you display the saved commands, search the list, modify commands, and rerun commands.

The simplest use of the command history list is to repeat a command exactly. For example, you might enter the command

```
c:\> dir b:*.wks;*.doc
```

to see some of the files on drive B. You might move some new files to drive B and then want to repeat the DIR command. Just press the **Up Arrow** key repeatedly to scan back through the history list. When the DIR command appears, press **Enter** to execute it again.

After you have found a command, you can edit it before pressing **Enter**. You will appreciate this feature when you have to execute a series of commands that differ only slightly from each other.

The history list is "circular". If you move to the last command in the list and then press the down arrow one more time, you'll see the first command in the list. Similarly, if you move to the first command in the list and then press the up arrow one more time, you'll see the last command in the list.

You can have 4DOS 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 **Up Arrow**. You only need to enter enough characters to identify the command that you want to find. For example, to find the DIR command, enter DI and then press **Up Arrow**. If you press the **Up Arrow** key a second time, 4DOS will display the next command that matches. 4DOS will beep if there are no matching commands. The search process stops as soon as you type one of the 4DOS 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 **Up Arrow** again.

You can specify the size of the command history list with the History directive in *4DOS.INI* (see page 122). When the list is full, 4DOS discards the oldest commands to make room for new ones. You can also use the HistMin directive in *4DOS.INI* to enable or disable history saves and to specify the shortest command line that 4DOS will save (see page 126).

You can prevent 4DOS from saving a command line by beginning it with an at-sign [@].

Command History Keys:

Up Arrow	Recall the previous (or most recent) command, or the most recent command that matches a partial command line.
Down Arrow	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 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
@	As the first character in a line: Do not save the current line in the history list when it is executed.

Use **F3** when your new command is different from your old one by just a character or two at the beginning. For example, suppose you want to execute a DIR on several file names then use DEL to delete those same files. After the DIR is complete type DEL and press F3; the rest of the command line will be completed for you. Check that it's correct, and then press Enter to delete the files. F3 also retrieves the entire previous command (like up-arrow) if nothing has been typed on the line.

Use **Ctrl-E** to "get your bearings" by returning to the end of the list if you've scrolled around so much that you aren't sure where you are any more.

Use **Ctrl-K** to save some work when you've typed a long command and then realize that you weren't quite ready. For example, if you forget to change directories and notice it after a command is typed or mostly typed, but before you press Enter, just press Ctrl-K to save the command without executing it. Use the CD or CDD command to change to the right directory, press up-arrow twice to retrieve the command you saved, make any final changes to it, and press Enter to execute it.

Command History Window

You can also view the command history in a scrollable **history window**, and select the command to modify or re-execute from those displayed in the window. To activate the history window press **PgUp** at the command line. 4DOS will display a window in the upper right corner of the screen showing the command you most recently executed. If this command is the last one entered, you'll see it and several previous commands. If it's in the middle of your history, you'll see the current command and a few commands before and after it.

You can scroll the history window up and down one line at a time with **Up Arrow** and **Down Arrow**, and one page (screen) at a time with **PgUp** and **PgDn**. The display is not circular as it is at the prompt, it has fixed beginning and end points. The **Ctrl-D** (delete from history) and **Ctrl-E** (end of history) keys work within the history window as they do at the command line. **Ctrl-PgUp** will go to the beginning of the history, and **Ctrl-PgDn** will go to the end (the same as **Ctrl-E**).

Entering any other key within the history window will close the window, move the currently selected line down to the command line where you can

edit it, and will take effect as if you had typed the key at the command line. When the line is moved to the prompt, the cursor will be placed at the right hand end of the line just as it is when you scroll through commands without using the history window. For example, **Enter** will move the current line to the command prompt and execute it immediately. **End** will move the current line to the prompt and leave the cursor at the end of the line. **Home** will move the current line to the prompt and then move the cursor to the start of the line. Other standard command line editing keys will work similarly: the line will be moved to the prompt then the editing key will be processed.

You can control the position and size of the history window with directives in *4DOS.INI* (see page 126). You can also change the keys used in the window with key mapping directives in *4DOS.INI* (see page 131).

Command History Window:

PgUp	(from the command line) Open the command history window.
Up Arrow	Scroll the display up one line.
Down Arrow	Scroll the display down one line.
PgUp	(inside the window) Scroll the display up one page.
PgDn	Scroll the display down one page.
Ctrl-PgUp	Go to the beginning of the history list.
Ctrl-PgDn or **Ctrl-E**	Go to the end of the history list.
Ctrl-D	Delete the selected line from the history list.
Enter	Execute the selected line.
Any other key	Move the selected line to the command line for editing, then perform the key's normal action.

Filename Completion

4DOS's filename completion can help you by filling in a complete file name on the command line when you only remember part of the name. For example, suppose you want to copy a file. You know that its name begins *AU* but you can't remember the rest of the name. Type

```
c:\> copy au
```

and then press the **Tab** key or **F9** key. 4DOS 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 4DOS found the file that you want, simply complete the command. If it didn't find the file that you were looking for, press **Tab** again to substitute the next filename that begins with *AU*. When there are no more filenames that match your pattern, 4DOS 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 4DOS will return to the previous matching filename. After you back up to the first filename, 4DOS will beep each time you press **Shift-Tab**.

If you want to enter more than one matching filename on the same command line, press **F10** when each desired name appears. 4DOS will keep that name and place the next matching filename after it on the command line. You can then use **Tab** (or **F8**) and **Shift-Tab** (or **F9**) 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 4DOS's extended wildcards (see page 71). 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**, 4DOS will use *.* as the matching pattern. If you type a filename without an extension, 4DOS 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 (see page 74), 4DOS will use the part of the include list at the cursor as the pattern to match.

If you would rather select files from a list of matching file names, see the SELECT command on page 273.

Filename Completion Keys:

F8 or Shift-Tab	Get the previous matching filename.
F9 or Tab	Get the next matching filename.
F10	Keep the current matching filename and display the next matching name immediately after the current one.

Multiple Commands

At times, you probably know the next two or three commands that you want 4DOS to execute. Instead of waiting for each one to finish before you type the next, you can type them all 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 can type the following command line:

```
c:\> copy *.txt a: ^ chkdsk a:
```

If you don't like using the caret as the command separator, you can pick another character using the SETDOS command (see page 280) or the CommandSep directive in *4DOS.INI* (see page 125).

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 batch files (see page 96) and alias definitions (see page 94) as well as from the command line.

Automatic Directory Changes

4DOS's automatic directory change feature gives you a quick method for changing directories. You can use an automatic directory change in place of the CD or CDD command. To do so, simply type the name of the directory you want to change to at the prompt, with a backslash [\] at the end, and 4DOS will switch to that directory. For example:

```
c:\> 4dos\
c:\4dos>
```

This feature can make directory changes very simple when it's combined with the CDPATH environment variable (see page 114). CDPATH includes a list of directories for the CD and CDD commands to search if

the directory you name does not exist below the current directory. Automatic directory changes use CDPATH as well. For example, suppose CDPATH is set to C:\;D:\;E:\, and the directory WIN exists on drive E:. You can change to this directory with a single word on the command line:

```
c:\4dos> win\
e:\win>
```

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

Internally, automatic directory changes use the CDD command; the text before the backslash can be anything that could be included after CDD on the command line, except "-". Arguments like "...." are allowed. For more information on directory changes see CD on page 179 and CDD on page 181.

Temporarily Disabling Aliases

At times, you may want to temporarily disable an alias that you have defined. You may have an alias that changes the defaults of a particular command, for example, and want to run the unmodified version of the command. To do so, precede the command name with an asterisk [*]. For example, if you have an alias for DIR which displays directories in 2-column paged mode by default, you can use the following command to display a directory in the normal single-column, non-paged mode:

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

Command Line Help

4DOS includes a complete help program (called *4HELP.EXE).* The help system includes complete help for all 4DOS internal commands, all standard DOS external commands, and many 4DOS features. It is indexed and fully cross-referenced, so you can move easily among related commands.

You can start the 4DOS help system to get help for any 4DOS or DOS command in several different ways.

If you type **HELP** at the 4DOS prompt, a list of all help topics will be displayed. Move the cursor bar to the topic you want using a mouse or the arrow keys, and press Enter to see help on that topic.

If you type **HELP** followed by a topic on the command line, 4DOS will skip the opening help screen and go directly to that topic. For example, if you need help with the COPY command, you can type:

```
help copy
```

If you press **F1** at the 4DOS prompt, 4DOS will display the list of all help topics just as if you had entered the HELP command. If you have already typed part or all of a command on the line, 4DOS will provide "context-sensitive" help by using the first word on the line as a help topic. If it's a valid topic, you will see help for that topic automatically; if not, you will see the list of all help topics and you can pick the topic you want. For example, if you press **F1** after entering each of the command lines shown below you will get the display indicated:

```
c:\>                        List of help topics
c:\> copy *.* a:            Help on COPY
c:\> c:\util\map            List of help topics
```

If you type the name of any 4DOS internal command at the 4DOS prompt, followed by a slash and a question mark [/?] like this:

```
copy /?
```

then 4DOS will display help for the command in a "quick-reference" style. Output from a /? display may be redirected with > or >>.

/? will only access the 4DOS help system when you use it with a 4DOS internal command. If you use it with an external command name, the external command will be executed and will interpret the /? parameter according to its own rules. Some external commands, including MS-DOS 5.0 external utility programs, do display help when run with a /? parameter, but this a characteristic of these commands and does not depend on 4DOS. Many other external commands do not have this feature.

Once you've started the 4DOS help system with **HELP** or **F1**, you can use a standard set of keystrokes to navigate. The table below gives a brief summary of keys you can use in the help topic list, and in a help text

screen; for details see the topic **-HELP-** in the help system itself. For details on mouse usage, see the topic **-MOUSE-** in the help system.

Help topic list keys:

Arrow Keys	Move the highlight to a different topic.
Enter	Display help on the highlighted topic.
Esc	Return to 4DOS.
Any other key	Attempt to match the characters typed with one of the names in the topic list.

Help text screen keys:

Up Arrow	Scroll up one line in the display.
Down Arrow	Scroll down one line in the display.
PgUp	Scroll up one page in the display.
PgDn	Scroll down one page in the display.
Left Arrow	Move the cross-reference highlight to the previous item.
Right Arrow	Move the cross-reference highlight to the next item.
Enter	Switch to the topic shown by the highlighted cross-reference item.
Esc	Return to the topic list, or back to 4DOS if this topic was displayed directly without using the topic list.
F1	Go to the topic list in order to select a new topic.
Alt-N	View the next topic in the topic list.
Alt-P	View the previous topic in the topic list.
Alt-F1	View the previously displayed topic.
Alt-X	Return directly to 4DOS without restoring the original screen contents.

The help system normally restores the screen when exiting. Use **Alt-X** to leave a page of help text on the screen so you can refer to it.

Input and Output

4DOS commands and many 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. But you can temporarily change these devices for special tasks.

For example, suppose you want a printed list of the files in a directory. If you change the standard output to the printer and issue a DIR command, the task is easy. DIR prints to standard output, and you have redirected standard output to the printer, so the DIR command prints filenames instead of displaying them on the screen. You can just as easily send the output of DIR (or any other command) to a file or a serial port.

4DOS has three methods of manipulating input and output: Redirection, Piping, and the Keystack. All three are explained in this section.

Redirection and piping affect the standard input, standard output, and standard error devices. They do not work with application programs which read the keyboard hardware directly, or which write directly to the screen.

Redirection

Redirection replaces **standard input**, **standard output**, and **standard error** with another device like the printer or serial port, or with a file. The redirection lasts for one command and then everything returns to normal. You have to use some discretion when you use redirection with a device; there is no way to get input from the printer, for example.

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; etc.).

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 /b1 *.dat > dirlist
```

You can use both input and output redirection for the same command, if both are appropriate:

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

Here are the redirection options supported by 4DOS:

To get input from a file or device instead of from the keyboard:

```
< filename
```

To redirect standard output to a file or device:

```
> filename
```

To redirect standard output and standard error to a file or device:

```
>& filename
```

To redirect standard error only to a file or device:

```
>&> filename
```

If you want to append output to the end of a file, replace the first "**>**" in the last three commands above with "**>>**" (use **>>**, **>>&**, and **>>&>**).

- When output is directed to a file with **>**, **>&**, or **>&>**, if the file already exists, it will be overwritten. You can protect existing files by using the SETDOS /N1 command (see page 280) or the NoClobber directive in *4DOS.INI* (see page 127).

- When output is appended to a file with **>>**, **>>&**, or **>>&>**, the file will be created if it doesn't already exist. Setting NoClobber will also prevent the creation of a new file. You can temporarily override the current setting of NoClobber by using an exclamation mark [**!**] after the redirection symbol. For example, to redirect the output of DIR to the file *DIROUT*, and allow overwriting of any existing file despite the NoClobber setting:

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

- 4DOS 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.

- For another method of changing the standard input and output devices see CTTY on page 192.

Piping

You can also create a "pipe," which means sending the standard output of one command to the standard input of another command. To send the standard output of *command1* to the standard input of *command2*:

```
command1 | command2
```

To send the standard output and standard error of *command1* to the standard input of *command2*:

```
command1 |& 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 DOS 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 4DOS's LIST command for full-screen viewing (see page 245):

```
c:\> set | sort | list /s
```

- 4DOS creates one or two temporary files to hold the output of pipes. The files are named *P1.$00* and *P2.$00*. By default, these files are stored in the root directory of the boot drive, but you can override this with either the TEMP4DOS or TEMP environment variable (see page 113). The last character of the extension will change with the shell nesting level (0 for the primary shell, 1 for the first secondary shell, and so on).

- The 4DOS commands TEE and Y (see pages 287 and 300) are "pipe fittings" which add more flexibility to pipes.

Keystack

The 4DOS Keystack overcomes two weaknesses of input redirection: many programs ignore standard input and read the keyboard directly, 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 4DOS Keystack, which is often used in batch files and aliases, 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.

The Keystack is invoked with the KEYSTACK command (see page 240). It depends on a small resident program called *KSTACK.COM*, which must be installed in your *AUTOEXEC.BAT* file (see page 112). If you don't have *KSTACK.COM* installed, the KEYSTACK command will display an error message.

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 the KEYSTACK command for details on how key names are entered and on using numeric key values along with or instead of key names.

Some programs may require a delay between keystrokes. You can insert a delay with the **/W** option, followed by a delay time in 1/18-seconds. To add a 1-second delay between the keystrokes in the previous example:

```
c:\> keystack F1 /W18 Enter
```

Some programs clear all keystrokes from the keyboard buffer and then accept input. Place a **0** (zero) in the keystack to tell such programs that the keyboard buffer is empty. This example reports an empty keyboard buffer and then types *myfile*:

```
c:\> keystack 0 "myfile"
```

Some programs require both the ASCII code and the key's scan code. To put both together in the Keystack, multiply the scan code by 256, add the ASCII code, and use the resulting numeric value as an argument to KEYSTACK. For example, the **Enter** key has a scan code of 28 and an ASCII code of 13. The combined code is (28 * 256) + 13 = 7181. To put the combined code for the Enter key on the keystack:

```
c:\> keystack 7181
```

If a program has different uses for the similar keys on the regular keyboard and the numeric keypad, it will need combined codes.

The following command creates an alias (see page 164) that will run a dBASE report called *TIMEREP* (it should be entered on one line):

```
c:\> alias drpt `keystack "use times index times" Enter
     "report form timerep to print" Enter "quit" Enter
      ^ dbase`
```

This command creates an alias called *DRPT* which puts the following characters on the keystack:

the characters "use times index times"
the Enter key's code
the characters "**report form timerep to print**"
the Enter key's code
the characters "**quit**"
and one more Enter key

The alias then runs the program *dBASE* which receives those characters just as if you had typed them.

You may have to experiment with some programs to find the proper sequence of keystrokes. Programs which bypass both DOS and the computer's BIOS, and read keystrokes directly from the keyboard hardware, will not accept input from the 4DOS Keystack. Few programs fit into this category except memory-resident utilities.

When you use the Keystack, remember that you must put the keystrokes into the Keystack before you run the program that will receive them. The Keystack will hold the keystrokes until a program asks for them.

See **Appendix B** on page 311 for a complete list of ASCII, extended key, and scan codes, and KEYSTACK on page 240 for more information.

File Processing

Most 4DOS commands (like COPY, DIR, etc.) and many external commands 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 4DOS's shorthand forms of naming files: **Wildcards**, **Multiple Filenames, Include Lists**, and

Executable Extensions. These four features are explained in this section.

Remember throughout this section that a filename is a **base name** of 1 to 8 characters, optionally followed by an **extension** which is a period [.] and 1 to 3 more characters.

Wildcards

Wildcards let you specify a file or group of files by typing a partial filename. 4DOS scans the appropriate directory to find all of the files that match the partial name you have specified.

Most 4DOS commands accept filenames with wildcards anywhere that a full filename can be used. 4DOS recognizes 2 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, regardless of the length of each file's name:

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

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

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

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

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

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

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

A question mark [?] matches any single filename character. Also, DOS automatically extends all base names to 8 characters and all extensions to 3 characters by adding blanks at the end of the names, if necessary. The question mark wildcard will match one of these blanks as well as a normal character. 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
```

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, is unique to 4DOS. These wildcards work only with 4DOS internal commands, not with external programs that accept file names and wildcards, unless such programs have been written especially to parallel 4DOS's features.

❖ In some cases, the question mark wildcard may be too general. You can also tell 4DOS 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 (unbracketted) question mark wildcard. A normal question mark wildcard matches any character or an implied blank at the end of a name or extension. 4DOS will match a question mark inside brackets with any character but not with an implied blank. For example,

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

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

❖ 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 4DOS wildcards:

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

Multiple Filenames

Most 4DOS file processing commands can work with multiple files at one time. To use multiple file names, you simply list the files one after another on the command line, separated by spaces. You can use wildcards in any or all of the filenames. For example, to copy all *.TXT* and *.DOC* files from the current directory to drive A, you could use this command:

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

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

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

! Multiple filenames are handy when you want to match a group of files which cannot be defined with a single filename and wildcards. They let

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

Like extended wildcards and include lists (see below), the 4DOS multiple filename feature will work with internal 4DOS commands but not with external programs unless those programs have been written to handle multiple file names on the command line.

If you have a list of files to process that's too long to put on the command line or too time-consuming to type, see the SELECT command on page 273 for another way of passing multiple file names to a command.

Include Lists

Any 4DOS command that accepts multiple filenames can also accept one or more include lists. An include list is simply a group of filenames, with or without wildcards, separated by semi-colons [;]. All files in the include list must be in the same directory. You may not add a space on either side of the semi-colon.

If you used an include list instead of multiple file names for the previous examples, they would look like this:

```
c:\> copy *.txt;*.doc a:
c:\> copy a:\details\*.txt;*.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 4DOS 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 most 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 (see above), the 4DOS include list feature will work with internal 4DOS commands but not with external programs unless they have been programmed especially to parallel 4DOS's features.

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, 4DOS looks for a *.COM*, *.EXE*, *.BTM*, or *.BAT* file with that name to execute (*.COM* and *.EXE* files are executable programs; *.BTM* and *.BAT* files are batch files). You can add to this list of extensions and have 4DOS take the actions you want with files that have other extensions as well. You could have 4DOS 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.

4DOS uses environment variables to define what program or batch file to run for each defined file extension. To create an executable extension, you use the SET command to create a new environment variable.

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
```

The syntax for creating an executable extension is

```
set .ext=d:\path\program [options]
```

where *.EXT* is the executable file extension, *D:\PATH\PROGRAM* is the full name of the program or batch file to run, and *[options]* are any command-line startup options you want to specify for the program. The pathname is optional if the program is in a directory on your PATH. The program to run must be a *.COM*, *.EXE*, *.BTM*, or *.BAT* file or an internal command. It cannot be an alias.

The following example defines *BASICA.COM* as the processor for *.BAS* files:

```
c:\> set .bas=c:\dos\basica.com
```

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

```
c:\> pushcart
```

4DOS will execute the command:

```
c:\dos\basica.com pushcart
```

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

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

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

```
c:\> hello -i30
```

4DOS will expand the command line and execute this command:

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

Notice that 4DOS inserts the value of the environment variable at the beginning of the line, including any options, then appends the original file name plus its extension, and then the remainder of the original command line.

If the 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.

- 4DOS searches for executable commands in the following order: *.COM*, *.EXE*, *.BTM*, *.BAT*, and executable extensions in the order they appear in the environment. It first searches the current directory, and then each subdirectory specified by the PATH environment variable (if a "." is used in the PATH the current directory is **not** searched first; see the PATH command on page 255 for details). 4DOS recognizes environment variables as executable extensions if they begin with a period followed by 1 to 3 valid filename characters.

  You may need to take this search order into account when using executable extensions. Using the *.BAS* example above, if you had a file *FORMAT.BAS* in the current directory and entered the command FORMAT A:, 4DOS would run the BASIC interpreter specified by the executable extension, instead of finding the standard DOS FORMAT command as you intended. You can get around this by remembering that the DOS FORMAT command is in the file *FORMAT.COM*. If you entered the command FORMAT.COM A: then 4DOS would not find a match for the executable extension, and would continue the usual search sequence until it found the FORMAT command.

- Executable extensions may include wildcards, so you could, for example, have 4DOS run your text editor for any file with an extension beginning with **T** by defining an executable extension called *.T**.

The Environment

The **environment** is a collection of information about your computer that every program receives. You can view the environment by typing SET, and modify it with the ESET, SET, and UNSET commands (see pages 212, 277, and 295).

Each entry in the environment consists of a variable name (usually in upper case) followed by an equal sign and a string of text. Some variables are of general use to many programs; some are used only by one program or group of programs. The content and form of the text string following the equal sign is defined by the program that uses each particular

environment variable. The text strings can be used from the 4DOS command line, by application programs, and within aliases and batch files.

The text string can include any characters except nulls (ASCII 0). The maximum length for the variable name, equal sign, and text string is 255 characters.

4DOS can automatically substitute the text for the variable name in a command. To create the substitution, include a percent sign [%] and a variable name on the command line or in an alias or batch file. For example, if you create a variable named BACKUP like this:

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

and then type

```
c:\> del %BACKUP
```

4DOS will execute 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 4DOS to accept 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".

- If you want to pass a percent sign, or a string beginning with a percent sign, to a command you must use two percent signs in a row. Otherwise 4DOS will see your single percent sign as the beginning of a variable name, and will not pass it 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%%
```

- Environment variables may be used to contain alias names. Normally 4DOS expands aliases before environment variables. However if you use an environment variable name (with a leading percent sign) as the first word in a command line, 4DOS will substitute the variable value for the name, then check for any alias name which may have been included within

the variable's value. For example, the following commands would generate a 2-column directory of the *.TXT* files:

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

❖ The trailing percent sign that was traditionally required for environment variable names is not usually required in 4DOS, 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 concatenate 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%")
```

4DOS Configuration Variables

The following environment variables have special meanings in 4DOS. Chapter 6 / Options and Fine Tuning (see page 113) explains the details of how to set and use each of them, except COLORDIR which is explained under the DIR and SELECT commands on pages 198 and 273. You can see the current value of each variable, if it exists, with the SET command.

CDPATH tells 4DOS where to search for directories specified by the CD and CDD commands and in automatic directory changes.

CMDLINE is the fully expanded text (up to 255 characters long) of the currently executing 4DOS command line. 4DOS sets CMDLINE just before it invokes any *.COM*, *.EXE*, *.BTM*, or *.BAT* file.

COLORDIR controls directory display colors used by DIR and SELECT.

COMSPEC contains the full path and name of 4DOS itself. COMSPEC is often used by applications which have a "shell to DOS" feature.

PATH is a list of directories that 4DOS 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.

PROMPT defines the 4DOS command line prompt.

TEMP specifies the directory where 4DOS should store temporary pipe files if the TEMP4DOS variable doesn't exist. Some other programs also use TEMP to define where they should place their temporary files.

TEMP4DOS specifies where 4DOS should store temporary pipe files.

❖ 4DOS Internal Variables

The following variables are not actually stored in the environment, but can be used in commands, aliases, and batch files just like any other environment variable. The values of these variables are stored internally in 4DOS 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, which will be stored in the environment.

These internal variables are often used in 4DOS batch files and aliases to examine system resources and adjust to the current computer settings. You can examine the contents of any internal variable from the command line with a command like this:

```
c:\> echo %variablename
```

In the list below, the possible values for most variables are shown in double quotes for ease of understanding. The actual values returned by the variables do not include the double quotes.

? 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 exit code is returned, the value of **%?** is undefined.

_? contains the exit code of the last internal 4DOS command. It is set to 0 if the command was successful, non-zero if not. You must use or save this value immediately, because it is set by every internal command.

_4VER is the current 4DOS version (for example, "4.0").

_ALIAS contains the free space in the alias list, in bytes.

_ANSI contains "1" if 4DOS's internal flags indicate that ANSI.SYS or a compatible driver is installed; "0" if not. The internal flags which determine the value of _ANSI depend on the SETDOS /A option (see page 280) and the ANSI directive in *4DOS.INI* (see page 124), as shown in the table below. If SETDOS /A is 0 or ANSI is set to Auto, 4DOS tests for the presence of an ANSI driver. Because there is no standard and 100% reliable way to detect an ANSI driver, you may need to experiment to see if this variable works properly with your particular driver when 4DOS is allowed to test for its presence.

SETDOS /A	ANSI Directive	_ANSI Value
0 (default)	Auto (default)	Result of test
1	Yes	1
2	No	0

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

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

_CODEPAGE is the current code page number (see CHCP on page 183).

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

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

_CPU is the cpu type, returned as a string:

86	8086 and 8088	286	80286
186	80186 and 80188	386	i386
200	NEC V20 and V30	486	i486

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

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

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

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

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

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

_DOS is the operating system type ("DOS" or "OS2"). 4DOS always returns "DOS", and 4OS2 always returns "OS2". This is useful if you have batch files running in both modes.

_DOSVER is the current DOS version (for example, "5.0"). In the OS/2 DOS compatibility box the version number will be 10.2 for OS/2 1.2, 10.3 for OS/2 1.3, 20.0 for OS/2 2.0, and so on.

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

_DV is "1" if DESQView is loaded or "0" otherwise.

_ENV is the free space in the environment, in bytes.

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

_MONITOR is the monitor type ("mono" or "color").

_NDP is the coprocessor type, returned as a string:

0	no coprocessor is installed
87	8087

287	80287
387	80387 or 80486DX

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

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

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

_TIME contains the current system time in the format hh:mm:ss. The separator character may vary depending upon your country information (see the CHCP command on page 183).

_VIDEO is the video card type ("mono", "cga", "ega", or "vga").

_WIN is the current Microsoft Windows mode. This variable will always be zero except when 4DOS is running under Microsoft Windows, or under a DOS session in OS/2 2.0:

0	Windows is not running
1	Windows 2
2	Windows 3 in 386 enhanced mode
3	Windows 3 in real or standard mode
20	OS/2 2.0 DOS compatibility box with Windows support

The _CWD, _CWDS, _CWP, _CWPS, and _DISK variables will return their result in upper or lower case depending on the value of the SETDOS /U switch (see page 280) or the UpperCase directive in *4DOS.INI* (see page 127). The _MONITOR and _VIDEO variables always return lower case. The _BG, _DOW, and _FG variables return the first letter of the result in upper case and the rest in lower case.

You can use these variables in a wide variety of ways depending on your needs. Here are just a few examples. Some of these examples rely on the IF command (page 229) or the IFF command (page 235) to test the value of a variable and perform different actions based on that value.

In a batch file, set the color based on the video card type:

```
iff "%_video"=="mono" then
  color bright white on black
else
  color bright white on blue
```

```
endiff
```

Call another batch file if 4DOS is running under DESQView:

```
if "%_dv" == "1" call dvstart
```

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

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

Set up a prompt for the primary shell which displays the time and current directory, and a different one for secondary shells which includes the shell level rather than the time. Also set different background colors for the two shells, without changing the foreground color. You might use a sequence like this in your *4START* file, which is executed each time 4DOS starts:

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

❖ 4DOS 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 variable functions are useful in aliases and batch files to check on available system resources, manipulate strings and numbers, and work with filenames. Some of the variable functions, like @DISKFREE, are shown with "**b|k|m**" as one of their arguments. Those functions return a number of bytes, kilobytes, or megabytes based upon a "**b|k|m**" argument:

b	return the number of bytes
K	return the number of kilobytes (bytes / 1,024)
k	return the number of thousands of bytes (bytes / 1,000)

M return the number of megabytes (bytes / 1,048,576)
m return the number of millions of bytes (bytes / 1,000,000)

@ASCII[c]: Returns the numeric value of the specified ASCII character as a string. For example **%@ASCII[A]** returns 65.

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

N Normal (no attribute bits set)
R Read-only
H Hidden
S System
D Directory
A Archive

The attributes (other than N) can be combined; ATTRIB will only return a 1 if **all** the attributes match.

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

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

@DESCRIPT[filename]: Returns the file description for the specified filename (see the DESCRIBE command on page 197).

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

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

@DISKUSED[d:,b | k | m]: Returns the amount of disk space in use by files and directories on the specified drive.

@DOSMEM[b | k | m]: Returns the amount of free base memory.

@EMS[b | k | m]: Returns the amount of free EMS memory.

@EVAL[expression]: Evaluates an arithmetic expression. @EVAL supports addition (+), subtraction (-), multiplication (*), division (/), and

modulo (%%). The expression can contain environment variables and other variable functions. @EVAL also supports parentheses, commas, and decimal places. Parentheses can be nested. The maximum number size is 16 digits to the left of the decimal point and 8 digits to the right of the decimal point. @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 = 11, not as (3 + 4) * 2 = 14. To change this order of evaluation, use parentheses to specify the order you want.

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

@EXTENDED[b|k|m]: Returns the amount of extended memory.

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

@FILESIZE[filename,b|k|m]: Returns the size of a file.

@FILETIME[filename]: Returns the time a file was last modified, in hh:mm format. The separator character will vary with the country definition in use on your system.

@FULL[filename]: Returns the fully qualified path name of a file.

@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.

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

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

@LINE[filename,n]: Returns line "n" from the specified file. The first line in the file is numbered 0. "**EOF**" is returned for all line numbers beyond the end of the file.

@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.

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

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

@READY[d:]: Returns "1" if the specified drive is ready; otherwise returns "0".

@REMOTE[d:]: Returns "1" if the specified drive is a remote (network) drive; otherwise returns "0".

@REMOVABLE[d:]: Returns "1" if the specified drive is removable (i.e., a floppy disk or removable hard disk); otherwise returns "0".

@SEARCH[filename]: Searches for the filename using the PATH environment variable, appending an extension (*.COM*, *.EXE*, *.BAT*, *.BTM*, or executable extension) if one isn't specified. Returns the fully-expanded name of the file including drive, path, base name, and extension, or an empty string if a matching file is not found. If wildcards are used in the filename, **@SEARCH** will search for the first file that matches the wildcard specification, and return the drive and path for that file plus the wildcard filename (*e.g.*, *E:\UTIL*.COM*).

@SUBSTR[string,start,length]: Returns a substring, starting at the position "start" and continuing for "length" characters. If the length is negative, the start is relative to the right side of the string. The first character in the string is numbered 0; if the length is negative, the last character is numbered 0. For example, %@SUBSTR[%_time,0,2] gets the current time and extracts the hour. If the string includes commas, it must be quoted with double quotes ["] or back-quotes [`]. The quotes **do** count in calculating the position of the substring to be extracted.

@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.

@TRUENAME[filename]: Returns the true, fully-expanded name for a file. TRUENAME will see "through" a JOIN or SUBST, and requires DOS 3.0 or above. Wildcards may not be used in the filename.

@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. This function allows you to create a temporary file without overwriting an existing file. @UNIQUE only works in DOS 3.0 and above.

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

@WORD[n,string]: Returns the "nth" word in a string. The first word is numbered 0.

@XMS[b|k|m]: Returns the amount of free XMS memory.

You can use these variable functions in a wide variety of ways depending on your needs. We've included a few examples below to give you an idea of what's possible.

To set the prompt to show the amount of free base memory (see the PROMPT command, page 259, for details on including variable functions in your prompt):

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

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

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

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

```
rem  Temporary variables used to shorten example lines:
rem    DD is _date, DY is yymmdd date, TM is _time
set dd=%_date
```

```
set
dy=%@substr[%dd,6,2]%@substr[%dd,0,2]%@substr[%dd,3,2]
set lastdate=0
iff exist c:\onceaday.dat then
  set lastdate=%@line[onceaday.dat,0]
endiff
iff %dy gt %lastdate then
  set tm=%_time
  iff "%@substr[%tm,0,2]%@substr[%tm,3,2]" gt "0600" then
    rem Commands to be executed once a day go here
    echo %dy > c:\onceaday.dat
  endiff
endiff
```

❖ Advanced Features

The next four 4DOS features are designed for advanced users of DOS and 4DOS. If you are a novice user, you might want to skim over this section and return to it as your computing skills and needs progress.

Conditional Commands

When an internal command or external program finishes, it returns a result called the exit code. 4DOS's conditional commands allow you to perform tasks based upon the previous command's exit code. Most 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 4DOS internal commands return an explicit exit code, but not all application programs do. Conditional commands will behave unpredictably if you use them with programs which do not return an explicit exit code.

Command Grouping

Command grouping allows you to logically group a set of commands together by enclosing them in parentheses. The parentheses are similar in function to the BEGIN and END block statements in some programming languages. Command grouping is a feature of the OS/2 command processor which 4DOS makes available to you under DOS as well.

There are two primary uses for command grouping. One is to execute multiple commands in a place where normally only a single command is allowed. For example, suppose you want to copy then rename all the *.WKQ* files on drives A: and B: using the FOR command. You could do it like this:

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

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

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

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

This kind of command grouping is most useful with the EXCEPT, FOR, GLOBAL, and IF commands. You can not 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** the command grouping parentheses, you just can't use command grouping to specify a group of commands for SELECT to execute.)

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

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

Using command grouping, these commands can be written much more simply (enter this 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. 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 that 4DOS displays a "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 511-character limit **includes** the space required to expand aliases and environment variables invoked within the group.

Escape Character

4DOS 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 sequences. The default 4DOS escape character is Ctrl-X (ASCII 24), which will be displayed here -- and on your screen -- as an up arrow [↑]. You can use the SETDOS /E command or the EscapeChar directive in *4DOS.INI* to select a different escape character if you wish.

4DOS recognizes six special characters if they are preceded by the escape character. The combination of the escape character and one of these characters is translated to a single character by 4DOS, as shown below. These are useful for redirecting codes to the printer; ↑e is also useful to generate ANSI "escape sequences" in your PROMPT, ECHO, or other output commands; and ↑r is used in keystroke aliases:

↑b	backspace
↑e	the ASCII ESC character (ASCII 27)
↑f	form feed
↑n	line feed
↑r	carriage return
↑t	tab character

If you follow the escape character with any other character, the escape character is removed and the second character is copied directly to the command line. This allows you to suppress the normal meaning of special characters (such as ? * / \ | " ` > < and &).

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

```
c:\> echos ↑f↑eY > prn
```

Argument Quoting

When it begins to parse the command line, 4DOS looks for carets [^] to break the line into individual commands, for redirection symbols, and for white space (blanks, tabs, and commas) to separate commands from arguments. It also looks for percent signs [%] which designate alias or batch file replaceable parameters (%1, %2, etc.), or environment variables, and substitutes the appropriate value for each variable (this process is called **variable expansion**). Normally, the separation characters and the percent sign cannot be passed to a command as part of an argument. However, you can include any of these special characters in an argument by enclosing the entire argument in 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 will be performed on an argument enclosed in back quotes. Redirection symbols inside the back quotes will be ignored.

The back quotes will be removed from the command line before the command is executed.

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

For 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
```

When an alias is defined in a batch file or from the command line, its argument should be enclosed in back quotes to prevent the expansion of replaceable parameters, variables, and multiple commands until the alias is invoked. Back quotes should **not** be used when defining aliases to be read in an ALIAS /R file.

Aliases

Much of the power of 4DOS 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 4DOS commands, operate as very fast in-memory batch files, and perform actions based on the results of other actions.

This section of the manual will show you some examples of the power of aliases. You can use these examples as the basis for your own aliases. See the ALIAS command (page 164) for complete details about writing your own aliases.

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

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

After the alias has been defined this way, every time you type the command ROOT, 4DOS will execute the command CD \.

Aliases can also create customized versions of 4DOS commands. For example, the 4DOS DIR command can sort a directory in various ways. You can create an alias called DE that means "sort the directory by filename extension, and pause after each page while displaying it" like this:

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

Aliases can be used to execute sequences of commands as well. The following command creates an alias called W which saves the current drive and directory, changes to the WP directory on drive C, runs the program *E:\WP51\WP.EXE*, and, when the program terminates, returns to the original drive and directory:

```
c:\> alias w = `pushd c:\wp ^ e:\wp51\wp.exe ^ popd`
```

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

```
c:\> alias wp = `e:\wp51\wp.exe`
c:\> alias w = `pushd c:\wp ^ wp ^ popd`
```

If you enter the W command, 4DOS will execute the PUSHD command, detect that the next command (WP) is another alias, execute the program *E:\WP51\WP.EXE*, and -- when the program exits -- return to the first alias, execute the POPD command, and finally return to the prompt.

You can use aliases to change the default options for both internal 4DOS commands and external commands. Suppose that you always want the DEL command to prompt before it erases a file:

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

You may have a program on your system that has the same name as an internal 4DOS command. Normally, 4DOS will run the internal command and not let you run the program you desire, unless you explicitly add its full path on the command line (type *C:\UTIL\LIST.COM*, for example). 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. For example, if you want to run a program called *HELP.EXE* when you type HLP, this alias will do the trick:

```
c:\> alias hlp = `c:\dos\help.exe`
```

Another approach you could take would be to rename the internal 4DOS command and use the original name for the external program. The following example renames the 4DOS LIST command as DISPLAY and then uses a second alias to run LIST.COM whenever you type LIST:

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

You can also assign an alias to a key, so that every time you press the key, the command will be invoked. After you enter this next example, 4DOS will display a 2-column directory with paging whenever you press Shift plus F5:

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

That example will put the DIR command on the command line when you press F5 and wait for you to enter file names to display and then press Enter. This next example clears the screen whenever you press Alt-F1. The up-arrow character [↑] is the 4DOS escape character, which you enter by pressing Ctrl-X (it is entered **twice** before the **r** at the end of this alias):

```
c:\> alias @Alt-F1 = `cls↑↑r`
```

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 4DOS will display the answer:

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

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

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

Now if you type

```
c:\> in c:\letters wp letter.txt
```

4DOS will change to the *C:\LETTERS* subdirectory, execute the command WP *LETTER.TXT* and then return to the current directory.

The distribution diskette includes a sample alias file called *ALIASES* which contains several useful aliases and demonstrates many alias techniques. Also, see the ALIAS and UNALIAS commands on pages 164 and 294 for more information and examples. See page 101 for tips about using aliases inside your batch files.

Batch Files

A batch file is a file that contains a list of commands for 4DOS to execute. 4DOS 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.

.BAT Files and .BTM Files

4DOS can execute a batch file in two different modes. In the first, traditional mode, 4DOS reads and executes the lines of the batch file individually. In the second mode, 4DOS reads the entire batch file into memory at once. The second mode can be 5 to 10 times faster, especially if most of the commands in the batch file are internal 4DOS commands. However, only the first mode can be used for self-modifying batch files (which are rare), for batch files which install memory-resident utilities, and for batch files larger than 64K bytes.

4DOS decides which batch file mode to use by the file's extension. It runs files with a *.BAT* extension in the slower, traditional mode. Files that have a *.BTM* extension are run in the faster, more efficient mode. You can change the execution mode inside of a batch file (of either type) with the LOADBTM command (see page 247).

Echoing

By default, 4DOS displays or "echoes" every batch file line 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 (see page 280) or the BatchEcho directive in *4DOS.INI* (see page 125).

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

```
@echo off
```

4DOS also has a command line echo that is unrelated to the batch file echo setting. See the ECHO command on page 208 for details about both settings.

Batch File Variables

Like aliases and application programs, batch files can examine the command line that is used to invoke them. 4DOS separates the command tail (everything on the command line after the batch file name) into individual parameters or **variables** by scanning for the spaces, tabs, and commas that separate the parameters. A batch file can work with individual parameters or with the command tail as a whole.

4DOS numbers these replaceable parameters from **%1** to **%127**. It is up to the batch file to determine the meaning of each parameter. 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.

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**". The default value of "**n**" is 1, so **%&** contains the entire command tail. The values of these special parameters can change if you use the SHIFT command (see page 285).

For example, suppose the 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 on pages 229 and 235).

- Batch files can also use environment variables, internal variables, and variable functions. See pages 80 - 89 for a complete list of the internal

variables and variable functions available. 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 (including date arithmetic), manipulate strings and substrings, extract parts of a filename, and perform simple file access.

- 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 (page 295) to remove these variables from the environment at the end of your batch file. You can also use SETLOCAL and ENDLOCAL (pages 284 and 211) 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.

- Be careful not to confuse the various kinds of variables that 4DOS provides. Replaceable parameters, environment variables, internal variables, and variable functions look similar and have similar uses but they are not the same.

Batch File Commands

Several 4DOS commands are particularly suited to batch file processing. Each command is explained in detail in the **Command Reference** section of this manual, beginning on page 157. Here is a list of some of the commands you might find most useful:

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

CALL executes one batch file from within another.

CANCEL terminates all batch file processing.

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

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 be redirected to a file or 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 places keystrokes into the 4DOS Keystack.

LOADBTM changes the batch file operating mode.

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

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

REM places a remark in a batch file.

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

SCRPUT displays a message in color.

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

SHIFT changes the numbering of the replaceable parameters.

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

TIMER starts or reads a stopwatch.

VSCRPUT displays a vertical message in color.

These commands, along with the 4DOS internal variables and variable functions, make the 4DOS batch file language extremely powerful. The distribution diskette contains a number of sample batch files that demonstrate some of the things you can do with 4DOS batch files.

❖ Batch File Tips

This section gives you some tips on batch file programming, including a few things we've found useful and a few that you should stay from.

One way to simplify batch file programming is to use aliases. Not aliases that can be used from the command line, but aliases that hide unnecessary detail inside a batch file. For example, suppose you want to implement a multiple choice list in a batch file that will let you select among several different applications. This example shows one way to do so:

```
alias in `pushd %1 ^ %2& ^ popd`
alias choice `elseiff "%userchoice"=="%1" then`
:dispmenu
screen 8 0
text
 Enter your choice:
      1. Word Processing
      2. Spreadsheet
      3. Communications
endtext
inkey %%userchoice
iff "1"=="2" then ^ rem  Always fail and go to next line
choice 1 ^ in d:\letters  c:\wp51\wp.exe
choice 2 ^ in d:\finance  c:\quattro\q.exe
choice 3 ^ in d:\comm  c:\comsw\pcplus.exe
else
  scrput 23 0 bri whi on red Invalid choice, try again
  goto dispmenu
endiff
```

The first 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 the

name of an application program plus possible command line parameters, which the alias executes. This alias could be used from the command line.

The second alias, CHOICE, expects 1 command line argument. It tests whether an environment variable called *userchoice* has the same value as the command line argument. But the test comes between ELSEIFF and THEN statements. If you try to use CHOICE from the command line, 4DOS will generate an error because it will see ELSEIFF and THEN without the necessary IFF and ENDIFF to define a block.

The next 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 batch file tests the user's keystroke to decide what action to take.

Since the CHOICE alias starts with an ELSEIFF command, there has to be an IFF. But the first condition must fail in order to get to the CHOICE aliases, so the IFF statement starts with a condition that will always be false. The three lines after the IFF are easy to follow and much shorter than they would be without the aliases. If it had to be typed in full, the first choice line would look like this:

```
elseiff "%userchoice"=="1" then pushd d:\letters ^
  c:\wp\wp51.exe ^ popd
```

There's another side to aliases in batch files. If you're going to distribute your 4DOS batch files to others, you need to remember that they may have aliases defined for the commands you're going to use. For example if the user has aliased CD to CDD and you aren't expecting this, your file may not work as you intended. There are two ways to address this problem. First, you can use SETLOCAL, ENDLOCAL, and UNALIAS to clear out aliases before your batch file starts and restore them at the end. For example:

```
setlocal
unalias *
rem Aliases cleared, set up new aliases
alias ...
rem Other batch file commands go here
endlocal
```

Remember that SETLOCAL and ENDLOCAL will save and restore not only the aliases but also the environment and the current drive and directory.

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 tells 4DOS not to interpret the command that follows it 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, 4DOS will skip alias processing, discard the asterisk, and process the rest of the command 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.

There are several tricks you can use in debugging batch files. Probably the simplest is to turn ECHO on at the beginning of the file while you're testing it. This will give you a picture of what 4DOS is doing when it executes the file. 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.

Another trick is to insert PAUSE commands wherever you need them in order to be able to watch what's happening. For complex debugging, you can make an alias that uses INKEY for a similar purpose, but helps you out a little more. For example (enter this on one line):

```
alias step `set skey=^inkey Step:  %%skey^iff
"%skey"=="S" then ^ set /P ^ elseiff "%skey"=="X" then ^
quit ^ endiff`
```

Now anywhere you insert a STEP command in your file you'll get the **Step:** prompt during execution. At that point you can type S to do a SET /P and display the current contents of your environment variables, X to exit the batch file, and any other key to go on. Of course you'll need to

create an alias tailored to your particular needs, but this gives you the basic idea.

If you can't figure out how 4DOS is expanding your aliases and variables, 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 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 DOS output devices can't be recorded, but you can still gain a lot of useful information if your batch file does much output.

If you're using redirection to see the output, remember that any prompts for input will probably go to the output file and not to the screen, so you need to know in advance the sequence of keystrokes required to get through the entire batch file, and enter them by hand or with KEYSTACK. (As an example, the raw text for the 4DOS tour in this manual was created by writing a batch file containing the demonstration commands, testing it to get the proper keystroke sequence, and executing a command which used KEYSTACK to send the keystrokes and redirected the batch file's output to a disk file.)

Because of the power of the 4DOS batch language, some programmers begin to treat it as a full, general-purpose language. But the batch language is interpreted one line at a time, which can lead the unwary into some subtle traps.

In particular, the IFF and GOTO instructions may not work together as many programmers expect. 4DOS has no way of telling whether a GOTO is branching within the same IFF block, into another IFF block, or out of all IFF blocks, so it has to make an assumption. It assumes that the

branch is leaving the IFF block and it therefore cancels all pending IFF processing. For example, the following commands **will NOT work:**

```
iff "%_monitor" == "color" then
  iff %@diskfree[c:,K] gt 100 then
    set var1=Y
    goto point1
  else
    set var1=N
  endiff
else
  :point1
  rem  do something else here
endiff
```

If the GOTO in the 4th line is executed, the commands after :POINT1 will execute properly. But 4DOS will generate an error message when it reaches the last ENDIFF, because it assumed that the GOTO branched outside of all IFF blocks. You can override this restriction with GOTO /I, but do so only if you are absolutely certain that your GOTO command is branching entirely within the current active IFF statement, and not into another IFF statement or a different IFF nesting level. Using /I under any other conditions will cause an error later in your batch file.

Because the IFF state is saved whenever a GOSUB occurs or whenever another batch file is CALLed, GOTO only cancels IFF processing within the current subroutine or batch file. This allows you to use GOTO inside a subroutine without affecting any IFF blocks that surround the GOSUB statement which called the subroutine.

GOSUB and GOTO can also have unexpected consequences when mixed with chains to new batch files. The GOSUB, and any block that contains it, remains active after the second batch file ends. Here is another example that **will NOT work**:

BAT1:

```
rem  do something here
gosub sub1
quit
:sub1
rem  do something else here
if [some condition] BAT2
return
```

BAT2:

```
rem  do some more work
BAT1
```

If the condition near the end of *BAT1* is true, then *BAT2* is launched. But *BAT2* ends by restarting *BAT1*. The original *BAT1*'s subroutine is still pending. Since 4DOS has no way to detect this infinite loop, it will eventually suffer a stack overflow and hang the system. If you do want to chain from one batch file to another, set an environment variable inside the GOSUB block and then use that variable outside the block to see whether you should CALL another batch file.

❖ CHAPTER 6 / OPTIONS AND TUNING

Normally, the 4DOS install program will properly set up your system to run 4DOS. It must, however, make certain assumptions about your computer and the way that you will use 4DOS. This chapter explains how you can tune 4DOS to make it as efficient and as useful as possible in your computing environment.

Nearly everything in this chapter is for advanced users and those with unusual needs. If 4DOS works the way you want it to after the automatic installation, you can skip this chapter. You may, however, want to skim this material to see what options are available.

Configuration Files

4DOS uses five configuration files. Two are for general DOS configuration; the other three are unique to 4DOS. The general DOS files are *CONFIG.SYS* and *AUTOEXEC.BAT*. The specialized 4DOS files are called *4START.BTM*, *4EXIT.BTM*, and *4DOS.INI*. These five files are discussed in order in this section.

! Anytime you change a configuration file, a typographical mistake or other error could make your system lock up or run erratically. Before you make any changes to any of these files, we strongly urge you to take the some precautions.

First, create a boot-up floppy diskette (you can use the FORMAT /S command) and make sure that you can use it to boot your computer. Second, make a backup copy of all five configuration files. You might want to create copies that have the same base name plus an extension of *.BAK*. Copy the *.BAK* files to your boot-up floppy for safe keeping. You also might want to make a copy of your computer's SETUP information (on 80286, 386, and 486 computers) and save it on this floppy. If the battery that keeps that data in your computer fails, this file will be invaluable after you install a new battery. A short text file should be sufficient.

With these two precautions, if something goes wrong, you will be able to boot with your floppy diskette and copy the files back to their original names. You'll only have to spend a few minutes recovering your system. You probably will want to follow the same precautions each time you

install a new application program that changes your DOS or 4DOS configuration files.

Setting up CONFIG.SYS

The *CONFIG.SYS* file contains instructions for DOS to execute before it loads a command processor. Only one line in the *CONFIG.SYS* file affects 4DOS. It tells DOS to use 4DOS as the command processor instead of *COMMAND.COM*. The format of this line is:

SHELL=d:\path\4DOS.COM [d:\path] [@d:\path\inifile] [/E:nnnn] [/F] [/P] [//iniline]... [command]

"SHELL=" identifies this line as defining the command processor that DOS will load after it finishes executing the commands in *CONFIG.SYS*. It is required.

If 4DOS is not in the root directory of the boot drive, replace the first "d:\path\" (immediately after "SHELL=") with the 4DOS drive and directory (if you are using DOS 2.x, 4DOS must be in the root directory of the boot drive). The drive and path must be correct or your system won't boot (and you'll be very happy that you made the boot-up floppy we suggested). The remainder of the items on this line are optional. If they are used, you should not include the square brackets. In the descriptions below, "d:" means a drive letter and "\path" means a subdirectory name.

d:\path This is the "d:\path" option shown in square brackets above (not the "d:\path" immediately after "SHELL="). It sets the drive and directory where 4DOS is stored. 4DOS uses this path to set the COMSPEC environment variable. When running under MS-DOS or PC-DOS 3.0 or above, 4DOS will normally find itself automatically and this option will not be needed. When running under DR-DOS this option is required unless *4DOS.COM* is in the root directory of the boot drive. When 4DOS is running as the primary command processor, you can tell if COMSPEC has been set correctly by typing the following line at the 4DOS prompt:

```
echo %comspec
```

If 4DOS has properly located itself, the location of *4DOS.COM* will be correctly displayed. If the COMSPEC is incorrect, you can set it yourself in AUTOEXEC.BAT, or add this "d:\path" option on the SHELL= line.

@d:\path\inifile

This option sets the path and name of the *4DOS.INI* file, which is discussed below. If the INI file is in the root directory of your boot drive and named *4DOS.INI*, or you aren't using any INI file, this option is not needed. Otherwise, it must be included. If you include only a path and not a file name, the name *4DOS.INI* will be assumed.

/E:nnnn

This option sets the size of the environment, in bytes. If you don't use this option, 4DOS will allocate 512 bytes for the environment. You can use any value from 256 to 32000 as the environment size. For example, to set an environment of 1,000 bytes, you would enter the option this way:

```
/E:1000
```

You can also set the environment size with the Environment directive in the *4DOS.INI* file (see below).

/F

This option tells 4DOS to automatically provide a Fail response to all critical errors, without prompting or waiting for a user response. It is rarely used except on systems that must run unattended, like bulletin boards. We do **not** recommend use of this option on a normal system, because you will not have a chance to react to a critical error and correct the problem that caused it. For more information on critical errors see page 143. **/F** only affects critical errors detected by 4DOS, and will not affect critical error handling for many application programs which perform this function themselves.

/P

This option tells 4DOS to load permanently and to run *AUTOEXEC.BAT*. When 4DOS is loaded from the SHELL= command in CONFIG.SYS, it will detect that it is the primary shell and set the **/P** option automatically. Under very rare circumstances you may want to load

4DOS permanently and have it run AUTOEXEC even though you are not loading it from *CONFIG.SYS*; in such cases you must set **/P** yourself. 4DOS will not run *AUTOEXEC.BAT* without a **/P**. Do not use this option in secondary shells, or you will be unable to return to the primary shell.

//iniline This option tells 4DOS to treat the text appearing between the // and the next space or tab as a *4DOS.INI* directive (see page 116 for information on *4DOS.INI* directives). The directive should be in the same format as a line in *4DOS.INI*, but may not contain spaces, tabs, or comments. Directives on the SHELL= line override any corresponding directive in *4DOS.INI*. This is a convenient way to place one or two simple directives on the SHELL= line without having to modify or create a *4DOS.INI* file, but is primarily intended for use in secondary shells (see below).

command This option tells 4DOS to run the command included on the SHELL= line. The command will be run after *4START* and *AUTOEXEC.BAT* but before displaying the prompt. It can be any valid alias, internal or external command, or batch file, and can include multiple commands (but see the Caution below). All other startup options (such as **/F** and **/P**) must be placed before the command, because 4DOS will treat characters after the command as part of the command and not as additional startup options. Use this option if you want 4DOS to run a file other than *AUTOEXEC.BAT* when your system boots: simply rename *AUTOEXEC.BAT* and place the new name, including its full path, at the end of the SHELL= line.

Caution

! There is a bug in all versions of MS-DOS and PC-DOS from 2.0 through 4.01. In all these versions, the SHELL= line in the *CONFIG.SYS* file may not contain more than 31 characters following the name of the shell program (*i.e.*, beginning with the space after the "M" in "*4DOS.COM*"). If the line is too long, the options will not be passed properly to 4DOS and a variety of errors can occur. You can set all necessary 4DOS options without exceeding this limit, especially if you put *4DOS.COM* and

4DOS.INI in the root directory of your boot drive. This limit is not present in MS-DOS 5.0 and above or in DR-DOS.

4DOS and DOS 2

We recommend using DOS 3.1 or above, but 4DOS can be used with DOS 2. The only special consideration is that 4DOS must be loaded differently under DOS 2.x, because certain DOS 2 functions require that *COMMAND.COM* be loaded as the primary command processor. Therefore, you must load *COMMAND.COM* first and then 4DOS. Assuming that all files are in the root directory of your boot drive, the SHELL= line in DOS 2.x should look like this:

```
shell=command.com /c 4DOS [options]
```

Note that due to the "/c 4DOS" on the SHELL line, fewer options can be used before running into the 31-character limit. You must, however, use the **/P** option, or *AUTOEXEC.BAT* will not run. 4DOS will not automatically detect that it is the primary shell and set **/P** for you when run under DOS 2.

Startup Options for Secondary Shells

In most cases, secondary shells do not use or require any of the startup options defined for the primary shell in *CONFIG.SYS*. Usually, you can only set explicit options for a 4DOS secondary shell when you define a 4DOS window in a multitasking system such as Back & Forth, DESQView, Windows, or the MS-DOS shell (DOSSHELL); or in rare cases when you run a secondary copy of 4DOS directly from the command line. If you do need to set options for secondary shells, you can use any of the following:

@d:\path\inifile
: Set INI file name, as in *CONFIG.SYS* (see above). This option is not necessary if you want 4DOS to use the same INI file that you used for the primary shell, as values from that file -- including those in its [Secondary] section -- will be passed automatically to secondary shells.

/C command
: This option forces 4DOS to execute a command and then return to the parent program. It is used by some applications to start the command processor, run one command, and the return to the application. This option

can be used when 4DOS is run as a secondary shell, but never in the SHELL statement in the *CONFIG.SYS* file. All other startup options must be placed before the command, because 4DOS will treat characters after the command as part of the command and not as additional startup options. This option cannot be used with the **command** option (below).

/E:nnnn Set the environment size, as in *CONFIG.SYS* (see above).

/F Force an automatic "Fail" on critical error, as in *CONFIG.SYS* (see above).

//iniline This option tells 4DOS to treat the text appearing between the // and the next space or tab as a *4DOS.INI* directive (see page 116 for more information on *4DOS.INI* directives). The directive should be in the same format as a line in *4DOS.INI*, but may not contain spaces, tabs, or comments. Directives on the SHELL= line override any corresponding directive in *4DOS.INI*. This allows you to use *4DOS.INI* directives directly on the command line when starting 4DOS in a window of a multitasking system, rather than having to create separate copies of *4DOS.INI* to accommodate small configuration changes in different windows.

command This option tells 4DOS to run the command included on the line, with the same restrictions and considerations as the **command** option in *CONFIG.SYS* (see above). The command will be run after *4START* but before displaying the prompt. Use this option if you want 4DOS to run a batch file or execute a command when it starts, for example to run a specific batch file when 4DOS is started in a window of a multitasking system. This option cannot be used with the **/C command** option (above).

Using AUTOEXEC.BAT

If 4DOS is the primary command processor, it is up and running before *AUTOEXEC.BAT* is executed. You generally won't need to make any changes to *AUTOEXEC.BAT* to make it run properly under 4DOS,

although once you get used to some of 4DOS's batch file enhancements, you may want to use them to streamline *AUTOEXEC*.

If you want to use the 4DOS KEYSTACK command, you will normally load the program *KSTACK.COM* from your *AUTOEXEC.BAT* file. To do so, include the following line in your file. Its location in the file is unimportant as long as you place it before any KEYSTACK commands:

```
d:\path\kstack.com
```

Replace the "d:\path" with the path to the *KSTACK.COM* program, which will normally be stored in your 4DOS directory.

You may want to use *AUTOEXEC.BAT* to configure 4DOS the way you want it by setting some of the environment variables that 4DOS uses. You may also want to include a SETDOS command (see page 280) to set configuration variables (most of these variables can also be set in the *4DOS.INI* file discussed below). All of these settings are optional.

4DOS uses seven environment variables. Five of the seven (all except PATH and PROMPT) are created with the SET command (see page 277) using this format:

```
set name=value
```

The seven variables and their uses are explained below:

COMSPEC: The COMSPEC variable is the path and filename that programs use to launch a secondary shell. Normally, this will be set automatically by 4DOS as it installs itself. However, in rare circumstances you may wish to load *4DOS.COM* for secondary shells from a directory other than the one it's in when you boot (for example, from a RAM disk). In these cases, you will need to reset the COMSPEC variable. For example, if you want to load secondary copies of *4DOS.COM* from the root directory of drive D, you would use this command:

```
set comspec=D:\4DOS.COM
```

PATH: The PATH variable lists the directories that 4DOS should search for executable files that aren't in the current directory (executable files include *.COM*, *.EXE*, *.BTM*, and *.BAT* files and files

with executable extensions). The PATH variable is normally set with the PATH command. See PATH on page 255 for more details.

PROMPT: The prompt variable defines the 4DOS command prompt. It is normally set with the PROMPT command. See PROMPT on page 259 for details about the options available.

TEMP4DOS: This variable contains the drive and directory that 4DOS will use to store temporary pipe files (see page 66). If you have a RAM disk, piping will be faster if you direct 4DOS to use it for the pipe files. This variable should hold a single path. The path can be terminated with a backslash [\] if you choose, but this is not required. For example, to direct pipe temporary files to the *TEMP* directory on drive *D*:

```
set temp4dos=d:\temp
```

TEMP: If TEMP4DOS does not exist, 4DOS looks for this variable to find the path for temporary pipe files (if neither exists, 4DOS uses the root directory of the drive from which it was started). TEMP is also used by some other programs to set a path for their temporary files. The format of TEMP is the same as TEMP4DOS.

COLORDIR: This variable sets the colors that 4DOS will use for directories displayed by the DIR and SELECT commands. See those commands (page 198 and 273) for details about COLORDIR.

CDPATH: This variable defines default directories to be used by the CD and CDD commands, and by automatic directory changes. If CD, CDD, or an automatic directory entry cannot find the path you specify on the command line, 4DOS will try appending the command line path to each path in this variable. For example, you may have a *C:\LETTERS* directory that has subdirectories named *JAN*, *FEB*, *MARCH*, etc. You can put *C:\LETTERS* into CDPATH and get to the FEB directory from anywhere on your system by typing CDD FEB or FEB\.

The format of the CDPATH variable is the same as the PATH variable: a list of directories separated by semicolons. For example:

```
set cdpath=c:\letters;c:\data;c:\memos
```

Using 4START and 4EXIT

4DOS runs two special batch files automatically, *4START.BTM* and *4EXIT.BTM* (you can make them *.BAT* files, if you prefer). *4START* is executed whenever 4DOS is started as a primary or secondary shell. In the primary shell, it is executed before *AUTOEXEC.BAT*. If 4DOS is started as a secondary shell with the **/C command** option, *4START* is executed before the command.

A *4START* file is normally very short. You should NOT load any memory-resident programs (TSRs) from *4START* because they will be reloaded every time a secondary shell is started. Normally, *4START* is used to vary the PROMPT setting or screen colors from shell to shell. The LOG command is sometimes used in *4START* to record that a new shell has begun. A PAUSE command inserted temporarily in *4START* will let you see if a particular program is running a secondary shell with the **/C command** option to accomplish some of its own work. Do not load aliases from *4START* -- 4DOS passes them along to a secondary shell automatically, so you only need to load them once, in *AUTOEXEC*.

The *4EXIT* batch file is executed every time a secondary 4DOS command processor ends. A secondary shell can end because of an EXIT command or after completing the command specified with a **/C command** startup option. Normally, *4EXIT* is used to LOG a record that the shell has ended. You might also use *4EXIT* to view the results of the command that an application has given to 4DOS, or to save the history accumulated during that shell with the HISTORY command with output redirected to a file.

4DOS looks for the *4START* and *4EXIT* batch files in three places:

- If the *4DOS.INI* file has a 4StartPath entry, 4DOS will use it.
- If the COMSPEC environment variable is set, 4DOS will look first in the root of the COMSPEC drive and then in the COMSPEC directory.
- If the above searches fail, 4DOS will look in the root directory of the current drive.

The easiest way to make sure that 4DOS finds the *4START* and *4EXIT* batch files is to specify their location with the 4StartPath directive in the *4DOS.INI* file (see below). If you don't, put *4START* and *4EXIT* in the root

directory of the boot drive. If the COMSPEC drive is not the same as the boot drive, then you should also put a copy of *4START* and *4EXIT* in the root directory of the COMSPEC drive.

Using the 4DOS.INI File

4DOS uses a file of initialization information called the *4DOS.INI* file. You can create, add to, and edit this file with any ASCII text editor to set 4DOS options and alter the way that 4DOS works. The 4DOS installation program will create a *4DOS.INI* file for you if one does not already exist on your system. Most of this section explains the options available through *4DOS.INI*. You only need to include entries in *4DOS.INI* for any settings that you want to change from their default values. If you are happy with all of 4DOS's default values, you don't need a *4DOS.INI* file at all.

Some settings in *4DOS.INI* are initialized when you install 4DOS, so you may have a *4DOS.INI* file even if you didn't create one yourself. You should not delete this file unless you've checked carefully to be sure that you don't need any of the settings the initialization program put there.

Most lines in the *4DOS.INI* file consist of a one-word **directive**, an equal sign [=], and a value. For example, in the following line, the word "Environment" is the directive and "2048" is the value:

```
Environment = 2048
```

Any spaces before or after the equal sign are ignored. The directive name may be abbreviated to the minimum length needed to make it unique; 4DOS will display an error if the name you use is unknown or ambiguous. We recommend that you use full-length directive names, since future versions of 4DOS may add directives that would make your abbreviations ambiguous and cause an error.

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 *4DOS.INI* file and can be used to separate groups of directives. You can place comments in the file by beginning a line with a semicolon [;]. You can also place comments at the end of any

line except one containing a text string value. To do so, enter at least one space or tab after the value, a semicolon, and your comment, like this:

```
Environment = 2048          ;set standard environment size
```

If you try to place a comment at the end of a string value, the comment will become part of the string and will probably cause an error.

When 4DOS detects an error while processing the *4DOS.INI* file, it displays an error message, ignores the line that caused the error, and prompts you to press a key to continue processing the file. This allows you to note any errors before the startup process continues. The directive in error will retain its previous or default value. Only the most catastrophic errors (like a disk read failure) will cause 4DOS to ignore all or a large part of the *4DOS.INI* file. If you don't want 4DOS to pause after each error, use a "PauseOnError = No" directive at the beginning of the file.

The *4DOS.INI* file has two sections, which are identified by a name in square brackets on a line by itself. The section names are

[Primary]: Directives in this section will be used when 4DOS is running as the primary shell. The same values will be passed automatically to all secondary shells, unless overridden by a directive with the same name in the [Secondary] section.

[Secondary]: Directives in this section are used in secondary shells only, and override any corresponding primary shell settings. For example, if your *4DOS.INI* file contains the lines:

```
ScreenRows = 25
[Secondary]
ScreenRows = 50
```

then 4DOS will assume that you have 25 rows on the screen in the primary shell and 50 lines in all secondary shells.

Lines that precede a section name are used in both primary and secondary shells.

The SETDOS command can override several of the *4DOS.INI* file directives. For example, the number of rows on the screen can be adjusted with SETDOS /R. The correspondence between SETDOS options and INI file directives is noted under each directive below, and under each option of the SETDOS command.

The *4DOS.INI* file normally should be stored in the root directory of the boot drive. You can specify a different location for the primary 4DOS shell with the "@d:\path\inifile" startup option in your *CONFIG.SYS* file (see page 108).

Secondary shells automatically inherit the configuration settings currently in effect in the previous shell. If values have been changed by SETDOS since 4DOS started, the new values will be passed to the secondary shell. If the previous shell's INI file had a **[Secondary]** section, it will then be read and processed. If not, the previous shell's settings will remain in effect. For example, if you set BatchEcho to Yes in *4DOS.INI*, do not include a **[Secondary]** section, and use SETDOS /V0 to turn off batch file echoing in the primary shell, then secondary shells will inherit the SETDOS setting, i.e. batch files will default to no echo.

If you want to force secondary shells to start with the same value as the primary shell for a particular directive, regardless of any changes made with SETDOS, repeat the directive in the **[Secondary]** section of *4DOS.INI*. You can also place the directive outside any section of *4DOS.INI*, then include an empty **[Secondary]** section. The inclusion of a **[Secondary]** section will force 4DOS to re-read *4DOS.INI*, and the directives outside any section will be reprocessed as they apply to both primary and secondary shells. Because *4DOS.INI* is processed after values from the previous shell are inherited, the value in *4DOS.INI* will override the inherited value. Using the BatchEcho example above, either of the following sets of *4DOS.INI* directives would work to ensure that BatchEcho is set to Yes in secondary shells:

Same directive in both sections

```
[Primary]
BatchEcho = Yes
  ... <other directives>
[Secondary]
BatchEcho = Yes
  ... <other directives>
```

Empty [Secondary]

```
BatchEcho = Yes
  ... <other
  ...  directives>
[Secondary]
<End of File>
```

If you start a secondary shell from a task switching program like Windows, DESQView, or Back & Forth, you can specify an alternate location and name for *4DOS.INI* by passing the "@d:\path\inifile" option to 4DOS as a command-line parameter (see page 109). In this case, the configuration settings in the alternate *4DOS.INI* file will supersede any settings inherited from the previous shell. Any values which are not explicitly set

in the alternate file will retain the value they had in the previous shell. Any **[Primary]** section in the alternate file will be ignored because it is being run from a secondary shell.

The first section below lists the different types of directives. Subsequent sections list all the individual directives, divided by function and then alphabetically by directive name. At the end we have included a few examples of how to use *4DOS.INI* directives (see page 133).

Types of Directives

There are 8 types of directives in *4DOS.INI*. When you look at the descriptions below, you can tell the directive type by the way in which the allowable values are shown.

- **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 can not 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. The choices can be abbreviated as long as the abbreviation is unique and can not cause ambiguity. For example, if the choices were shown as "YES | No" then "YES" is the default. You could enter "Y", "N", "y", or "n" for the value. But if the choices are shown as "YES | No | Never" then you could not use "N" as an abbreviation because it would be ambiguous.

- **Name = color**: This directive takes a color specification in the form:

  ```
  [BRIght] [BLInk] fg ON bg [BORder bc]
  ```

  Where:

  fg is the foreground color.

bg is the background color.
bc is the border color.

The allowable color names are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

The color names and the keywords **BRIGHT**, **BLINK**, and **BORDER** can be shortened to three letters.

- **Name = Key**: This directive takes a key specification in the form:

```
[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, Bksp, Enter, Left, Right, Home, End, PgUp, or PgDn
Shift	followed by F1 - F10 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, and can be abbreviated as long as the abbreviation is unique and unambiguous. Alphabetic keys can be specified in upper-case or lower-case. You cannot specify a punctuation key.

The prefix and key name can be together or separated by a dash, but cannot be separated by any other characters. For example,

```
AddFile = AltF10     ; This is okay
AddFile = Alt-F10    ; This is also okay
AddFile = 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. In general, you will find it easier to use the names described above rather than key numbers.

- **Name = Path:** This directive takes a path specification, but not a filename. The path should include both a drive and path (*e.g., C:\4DOS*) 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 = Anything else**: 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.

Initialization Directives

The directives in this section control how 4DOS starts and where it looks for its files.

4StartPath = Path: Sets the drive and directory where 4DOS should look for the *4START* and *4EXIT* batch files.

Alias = nnnn (1024): Sets the amount of memory in bytes allocated for the alias list. The allowable range of values is 256 to 32767 bytes.

AutoExecPath = Path: Sets the path used to find *AUTOEXEC.BAT* if 4DOS is started as a primary shell with the /P option in *CONFIG.SYS* (see page 109). The default is the root directory of the boot drive.

Environment = nnnn (512): Sets the amount of memory allocated to the environment in bytes. The allowable range of values is 256 to 32000 bytes.

EnvFree = nnnn (128): Sets the minimum amount of memory in bytes that will be available in the environment for secondary shells. 4DOS will enlarge the environment for each secondary shell, if necessary, so that there is at least this much free environment space when the shell starts. The allowable range of values is 128 to 32000 bytes.

HelpOptions = [/M]: Sets default options for the 4DOS help system. At present the only available option is **/M**, which forces HELP to use monochrome (black and white) colors even on color systems. You may find this useful on laptop or portable computers with color video boards and LCD or similar screens.

HelpPath = Path: Sets the path used to find *4HELP.EXE* when F1 is pressed or the HELP command is used. If this directive is not used, 4DOS will search the current directory and each of the directories in the PATH.

History = nnnn (1024): Sets the amount of memory allocated to the history list in bytes. The allowable range of values is 512 to 8192 bytes.

PauseOnError = YES | No: "Yes" tells 4DOS to pause with the message "Error in 4DOS.INI, press any key to continue processing" after displaying any error message related to a specific line in the *4DOS.INI* file. "No" continues processing with no pause after an error message is displayed.

Swapping = swap type [, swap type] ...: Sets the type of swapping 4DOS should use. 4DOS runs in two parts, a resident portion that is always in memory and a transient portion that can be stored in EMS memory, in XMS memory, on a RAM disk, or on your hard disk while application programs are running. The resident portion uses about 3K of memory in the primary shell and about 1.5K in secondary shells. The transient portion uses about 88K of memory. 4DOS will perform most quickly if the transient portion is swapped to the fastest memory or device available. See page 49 for explanations of XMS and EMS memory.

Swapping for the primary shell normally requires about 96K of EMS memory, or 88K of XMS memory or disk space. Secondary shells normally require 32K of EMS, or 22K of XMS or disk space. If you have a large environment, alias list, or history list, more memory will be required in both primary and secondary shells. The EMS requirements are larger because EMS memory is allocated in 16K increments; 4DOS does not actually use more memory when swapping to EMS.

The *swap type* may be:

EMS: 4DOS will swap to EMS expanded memory if it is available. You must have expanded memory and an EMS memory manager (version 3.2 or later) for this option.

XMS: 4DOS will swap to XMS extended memory if it is available (you must have an 80286, 386, or 486 computer for this option, and an XMS memory manager).

d:\path: 4DOS will create a swap file in the drive and directory specified. The file will be called *4DOSSWAP.NNN* where "NNN" is the shell number. This swap file is created as a hidden system file to avoid accidental deletion and will not be visible with a normal DIR command. Swapping to a RAM disk will generally be somewhat faster than swapping to a hard disk. Do not use a floppy disk for swapping because its performance is likely to be unacceptably slow.

None: No swapping. The transient portion of 4DOS will remain in memory at all times. This option will reduce memory available for application programs by about 90K compared to the other swap types, and should be used only when no other swapping options are available.

You can specify multiple swap types and 4DOS will try them in the order listed. Swap type "None" is always appended to your list of possible swap types as a "last resort", even if you don't include it explicitly. This allows 4DOS to start even if the other swap types you specify don't work.

For example, if your system has EMS memory and a RAM disk set up as drive D, the directive:

```
Swapping = EMS, D:\, C:\SWAP
```

will tell 4DOS to try EMS memory first, then the RAM disk, and finally the \SWAP directory on drive C. If all of these options fail (because there isn't enough free space available), the transient portion of 4DOS will remain in memory (swap type "None").

The default Swapping specification is:

```
Swapping = EMS, XMS, x:\, None
```

where "x" is the boot drive (for the primary shell) or the COMSPEC drive (for secondary shells). The default is usually adequate for most systems.

UMBEnvironment = Yes | NO: "Yes" attempts to load the master environment into a UMB (Upper Memory Block). This reduces 4DOS's base memory requirements but may cause problems with some programs that try to access the master environment directly. This option requires an 80286, 386, or 486 computer and appropriate support software. (See pages 50 and 136 for more information on UMBs.)

UMBLoad = Yes | NO: "Yes" attempts to load the resident portion of 4DOS into a UMB (Upper Memory Block). This reduces the size of the resident portion in base memory from about 3K bytes to 256 bytes, plus the environment size (unless you have also enabled UMBEnvironment). This option requires an 80286, 386, or 486 computer and appropriate support software. (See pages 50 and 136 for more information on UMBs.)

Configuration Directives

These directives control the way that 4DOS operates. Some can be changed with the SETDOS command while 4DOS is running. Any corresponding SETDOS command is listed in the description of each directive; information on SETDOS is on page 280.

ANSI = AUTO | Yes | No: Tells 4DOS whether an ANSI driver is installed and should be used for the CLS and COLOR commands. 4DOS normally determines this itself, but if you are using a non-standard ANSI driver or your loading sequence is unusual, you may need to explicitly inform 4DOS. Also see SETDOS /A.

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 4DOS "error" beeps (if you press an illegal key, for example). To disable all 4DOS 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 4DOS "error" beeps (if you press an illegal key, for example).

CursorIns = nnnn (100): This is the shape of the cursor for insert mode during command line editing and all commands which accept line input (DESCRIBE, ESET, etc.). The size is a percentage of the total character cell size, between 0% and 100%. Because of the way video BIOSes map the cursor shape, you may not get a smooth progression in cursor shapes as **CursorIns** and **CursorOver** change. Also see SETDOS /S.

CursorOver = nnnn (10): This is the shape of the cursor for overtype mode during command line editing and all commands which accept line input. The size is a percentage of the total character cell size, between 0% and 100%. Also see SETDOS /S.

CommandSep = c (^): This is the character used to separate multiple commands on the same line. You cannot use any of the redirection characters (| > <) or any of the whitespace characters (space, tab, comma, or equal sign). Also see SETDOS /C.

EditMode = Insert | OVERSTRIKE: This directive lets you start the command line editor in either insert or overstrike mode. Also see SETDOS /M.

EscapeChar = c (Ctrl-X [↑]): Sets the character used to suppress the normal meaning of the following character. See page 91 for a description of the escape character and special escape sequences.

You cannot use any of the redirection characters (| > <) or the whitespace characters (space, tab, comma, or equal sign) as the escape character. Also see SETDOS /E.

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 (save everything) to 256 (disable all command history saves).

HistWinColor = Color: Sets the default colors for the command line history window. If this directive is not used the colors will be reversed from the current colors on the screen.

HistWinHeight = nn (10): Sets the height of the command line history window in lines, including the border. Legal values range from 5 to the height of your screen; any value which would cause the bottom of the window to be off the screen will be adjusted so that the entire window remains on the screen.

HistWinLeft = nn (50): Sets the horizontal position of the left side of the command line history window. Legal values range from 0 (the left edge of the screen) to the number of columns on your screen minus 10. Any value which would cause the right side of a minimum-width window to be off the screen will be adjusted so that the entire window remains on the screen.

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

HistWinWidth = nn (30): Sets the width of the command line history window in characters, including the border. Legal values range from 10 to the width of your screen; any value which would cause the right side of the window to be off the screen will be adjusted so that the entire window remains on the screen.

LineInput = Yes | NO: This directive controls how 4DOS gets its input from the command line. "Yes" forces 4DOS to use line input via the DOS service INT 21H 0AH "Get Line" which is the way that

COMMAND.COM gets input. This will disable command-line editing, history recall, and filename completion, and is normally used only for rare memory-resident programs (TSRs) which do not work properly unless the command processor uses line input. If you have a particular program that requires line input, you can use SETDOS /L to temporarily change modes. See *APPNOTES.DOC* for any specific programs which require this option.

NoClobber = Yes | NO: If set to Yes, will prevent standard output redirection (see page *66*) from overwriting an existing file, and will require that the output file already exist for append redirection. Also see SETDOS /N.

ScreenRows = nnnn: Sets the number of screen rows used by the video display. Normally, 4DOS detects the screen size automatically, but if you have a non-standard display you may need to set it explicitly. This value does not affect screen scrolling, which is controlled by your video BIOS or ANSI driver. ScreenRows is used only by the LIST and SELECT commands, the paged output options of other commands (*e.g.*, TYPE /P), and error checking in the screen output commands. Also see SETDOS /R.

UpperCase = Yes | NO: "Yes" specifies that filenames should be displayed in the traditional upper-case by internal commands like COPY and DIR. "No" allows the normal 4DOS lower-case style. Also see SETDOS /U.

Color Directives

These directives control the colors that 4DOS uses for its displays.

ListColors = Color: Sets the colors used by the LIST and SELECT commands. If this directive is not used, LIST and SELECT will use the current default colors set by the CLS or COLOR command or by the StdColors directive, below.

StdColors = Color: Sets the standard colors to be used when CLS is used without a color specification, and for LIST and SELECT if ListColors is not used. Using this directive is equivalent to placing a COLOR command in AUTOEXEC.BAT. StdColors takes effect once the transient portion of 4DOS starts (i.e., when *4START* is run), but will not affect the color of error or other messages displayed during

the 4DOS loading and initialization process. If *ANSI.SYS* or a compatible driver is not loaded, the colors will not be "sticky" -- you may lose them when you run an application.

Key Mapping Directives

The directives in this group allow you to change the keys used for 4DOS command line editing and other internal functions. They take effect only inside 4DOS itself and do not affect other programs (including 4DOS's external help program, *4HELP.EXE*).

The description of each directive below explains the function of the corresponding key. Using the directive allows you to assign a different or additional key to perform the function described. For example, to assign function key F3 to invoke the 4DOS HELP facility:

```
Help = F3
```

Any directive can be used multiple times to assign multiple keys to the same function. For example:

```
ListFind = F          ;F does a find in LIST
ListFind = F5         ;F5 also does a find in LIST
```

Use some care when you reassign keystrokes. If you assign a default key to a different function, it will no longer be available for its original use. For example, if you assign F1 to the AddFile directive (a part of filename completion), the F1 key will no longer invoke 4DOS's HELP facility, so you will probably want to assign a different key to HELP.

4DOS will prohibit the mapping of standard alphabetic and numeric keys to any functions except those used by the LIST command (the directives that begin with "List" below). Otherwise, if (for example) you assigned the "E" key to scroll through the history list, you would not be able to type an "E" as part of any command name or filename.

4DOS processes all command line editing key assignments 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

standard 4DOS keys without assigning it to another function, use the NormalKey directive described below.

General Input Keys

This first set of Key Mapping Directives applies to all input. These directives are effective whenever 4DOS requests input from the keyboard, including command line editing and the DESCRIBE, ESET, INPUT, LIST, and SELECT commands. (Scrolling through the command history list is controlled by NextHist and PrevHist (see page 130), not by the Up and Down directives below.) See page 56 for more information about command line editing.

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.

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.

Down = Key (Down): Scrolls the display down one line in LIST; moves the cursor down one line in SELECT and in the command history window.

EndLine = Key (End): Moves the cursor to the end of the line.

EraseLine = Key (Esc): Deletes the entire line.

Ins = Key (Ins): Toggles between insert and overtype mode.

Left = Key (Left): Moves the cursor left one character; moves the display left 8 columns in LIST.

NormalKey = Key: Deassigns a key which has a specific meaning to 4DOS, so that it is treated like a "normal" key with no special function. For example,

```
NormalKey = F1
```

will disable the 4DOS HELP key. This directive can be used to deassign command line editing or other keys in order to make them available for keystroke aliases (see page 167).

Right = Key (Right): Moves the cursor right one character; scrolls the display right 8 columns in LIST.

Up = Key (Up): Scrolls the display up one line in LIST; moves the cursor up one line in SELECT and in the command history 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

The following directives apply only to command line editing. They are only effective at the 4DOS prompt.

AddFile = Key (F10): Keeps the current filename completion entry and inserts the next matching name.

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

DelHistory = Key (Ctrl-D): Deletes the displayed history list entry and displays the previous entry.

EndHistory = Key (Ctrl-E): Displays the last entry in the history list.

Help = Key (F1): Invokes the 4DOS HELP facility.

NextFile = Key (F9, Tab): Gets the next matching filename.

NextHistory = Key (Down): Recalls the next command from the command history.

PrevFile = Key (F8, Shift-Tab): Gets the previous matching filename.

PrevHistory = Key (Up): Recalls the previous command from the command history.

SaveHistory = Key (Ctrl-K): Saves the command line in the history list without executing it.

History Window Keys

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

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

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

LIST Keys

The keys in the last group of Key Mapping Directives are effective only inside the LIST command. Unlike the other key directives, these key assignments cannot be disabled with the NormalKey directive; however, the directives below can be used to assign additional keys to LIST functions.

ListFind = Key (F): Prompts and searches for a string.

ListHighBit = Key (H): Toggles LIST's "strip high bit" option, which can aid in displaying files from certain word processors.

ListNext = Key (N): Finds the next matching string.

ListPrint = Key (P): Prints the file on LPT1.

ListWrap = Key (W): Toggles LIST's wrap option on and off. The wrap option wraps text at the right margin.

Advanced Directives

These directives are used for unusual circumstances or for diagnosing problems. They are not needed in normal use.

CritFail = Yes | NO: This is the same as /F on the SHELL = line in *CONFIG.SYS*. It intercepts all DOS critical errors and returns a Fail to each. We do **not** recommend this on a normal system, because you will not have a chance to react to a critical error and correct the problem that caused it. It is intended for use on bulletin boards or other systems where unattended operation is required without user prompts.

Inherit = YES | No: Aliases and the history list are normally passed to secondary shells automatically. "No" disables this feature.

MessageServer = YES | No: For compatibility with *COMMAND.COM* in MS-DOS 4.x and 5.x, 4DOS includes a "message server" that retrieves error message text for DOS external commands like DISKCOPY and FORMAT. The message server increases the size of the resident portion of 4DOS by about 200 bytes. "No" disables the message server and saves this space, but will cause more cryptic error messages such as "Parse error 3" or "Extended error 7" from some DOS external commands. The message server is automatically disabled by 4DOS except in the primary 4DOS shell loaded from *CONFIG.SYS* when running under DOS 4.x or 5.x.

Reduce = YES | No: Set to "No" to disable the smaller swap size used by 4DOS secondary shells. For diagnosing unusual swapping problems only.

ReserveTPA = YES | No: Set to "No" to prevent 4DOS from reserving memory for its transient portion while at the command prompt. For diagnosing unusual TSR or swapping problems only.

StackSize= nnnn (3584): Set the 4DOS internal stack size. The allowable range of values is 3584 to 8192. You may need to increase the stack size if you are using extremely complex combinations of batch files and nested "prefix" commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line. Under such circumstances 4DOS may not work properly; you should only use this directive if you are actually experiencing trouble under such conditions. For virtually all users the default stack size will be sufficient. Increasing this value also increases the size of 4DOS's transient portion and the size of the 4DOS swap area.

SwapReopen = Yes | NO: Set to "Yes" to enable reopening of the 4DOS swap file if it is closed by another program. This is required when swapping 4DOS to Novell Netware drives. In all other circumstances, it is only useful for diagnostic purposes.

Examples

The following examples will give you an idea of the types of things that can be done with the *4DOS.INI* file. The comments on each directive explain what it does.

First, a very simple example that just sets up swapping and environment size, leaving everything else at its default value:

```
Swapping = ems, c:\              ;try EMS, then C: root
Environment = 1024               ;set environment size
```

Here's something a little fancier that changes a number of the default settings:

```
Swapping = xms, h:\, c:\         ;try XMS, then RAM disk,
                                 ;  then C: root
Environment = 1792               ;set env size
Alias = 6144                     ;set alias size
History = 1024                   ;set history size
UmbEnv=Y                         ;master environment in UMB
BatchEcho = No                   ;default is ECHO OFF
EditMode = Insert                ;editor in insert mode
CursorO = 100                    ;overstrike cursor 100%
CursorI = 10                     ;insert cursor 10%
```

The final example is similar to the second, but includes key reassignments and a **[Secondary]** section to vary a couple of settings in secondary shells:

```
; ALL SHELLS
PauseOnError = No                ;don't stop on an error
Swapping = xms, c:\              ;try XMS, then C: root
Environment = 1792               ;set env size
Alias = 6144                     ;set alias size
History = 1024                   ;set history size
UmbEnv=Y                         ;master environment in UMB
BatchEcho = No                   ;default is ECHO OFF
EditMode = Insert                ;editor in insert mode
CursorO = 100                    ;overstrike cursor 100%
ListFind = F5                    ;set LIST find to F5
ListNext = F6                    ;find next is F6
ListPrint = F7                   ;print file is F7
;     PRIMARY SHELL ONLY
```

```
[Primary]                          ;set primaries
CursorI = 10                       ;insert cursor 10%
StdColors = bri whi on blu         ;set primary's colors
;     SECONDARY SHELL ONLY
[Secondary]                        ;set secondaries
CursorI = 30                       ;insert cursor 30%
StdColors = bri whi on cya         ;set secondary's colors
```

CHAPTER 7 / USING 4DOS WITH YOUR HARDWARE AND SOFTWARE

This section of the manual explains how to get the most from 4DOS with your particular system. A few general techniques and concepts are explained here, but if you have questions about how to make 4DOS run most efficiently with a particular hardware setup or a specific application, you should also see the file called *APPNOTES.DOC* which comes with 4DOS.

If you have questions about some of the terms and concepts here, see Chapter 4 / General Concepts on page 41, the Glossary on page 325, and the Index.

This section begins with a discussion of hardware considerations and includes some tips for installing 4DOS most effectively to run with several types of software, as well as certain popular programs. It concludes with some techniques for resolving unintended interactions between programs, even if those interactions do not include 4DOS itself.

Hardware

The CPU

The CPU or "Central Processing Unit" is the chip which performs or directs all of the work done by your computer. All PC CPU chips are part of or compatible with Intel's "80x86" family. These include the 8088, 8086, 80188, 80186, 80286, 386, 486, NEC V20, and NEC V30, plus "SX" versions and other variations of some of those chips. 4DOS is compatible with and will run equally well on all of these chips.

Some systems have a numeric coprocessor as a companion to the CPU. The numeric coprocessor performs many arithmetic calculations faster than the CPU. 4DOS does not use or access the numeric coprocessor in any way.

You can determine which CPU chip your system has by using 4DOS's _CPU internal variable:

```
c:\> echo %_cpu
```

Similarly, you can find out if you have a numeric coprocessor with _NDP:

```
c:\> echo %_ndp
```

See page 82 for details about _CPU and _NDP.

Memory

4DOS does its best to detect and properly access all types of memory that your computer can have: Base memory, Expanded (EMS) Memory, Extended (XMS) Memory, and Upper Memory Blocks (UMBs). 4DOS always uses standard, documented methods to use the memory that you have installed.

4DOS uses base memory (the area from 0 to 640K on most machines, which is sometimes called "low memory" or "DOS memory") for its resident portion and the master environment, and to hold its transient portion while your system is at the command prompt or executing a 4DOS command or batch file. 4DOS may use EMS memory or an XMS Extended Memory Block (EMB) to swap its transient portion, according to the Swapping directive in your *4DOS.INI* file (see page 122).

4DOS uses UMBs for several purposes:

- to move the 4DOS resident portion out of base memory, if you specify "UMBLoad = Yes" in your *4DOS.INI* file.
- to move the master environment out of base memory, if you specify "UMBEnvironment = Yes" in your *4DOS.INI* file.
- to load memory-resident programs (TSRs) "high" using the LOADHIGH or LH command under MS-DOS 5.0.

To load 4DOS or the master environment into a UMB, you **must** be using a memory manager or XMS driver which provides **both** the ability to remap memory into the area between 640K and 1MB (to create the UMBs) and XMS or DOS 5.0 UMB support (to manage the UMBs). These are generally the same requirements which must be met to load TSRs "high."

To give 4DOS access to UMBs, you need hardware and software combinations like the following:

386 and 486 systems (including 386SX computers):

Hardware: Sufficient installed RAM.

Software: Qualitas' 386MAX or Blue Max, Quarterdeck's QEMM 5.0 or later, DOS 5.0's *EMM386.SYS*, or a similar 386 memory manager. *HIMEM.SYS* alone is **not** sufficient.

80286 systems:

Hardware: Chips and Technologies NEAT chip set, or an EMS 4.0 or EEMS memory board, plus sufficient installed RAM.

Software: Qualitas' MOVE-EM 1.02 or later with Microsoft's *HIMEM.SYS*, or Quarterdeck's QRAM and QEXT.

Other memory-management software may also work. The lists above are examples only. On the 386 systems here at JP Software, we use 386MAX almost exclusively, but we also have had good results with QEMM. Look for specific information about your memory management program in *APPNOTES.DOC*.

If you want to use the 4DOS LOADHIGH command as well as put 4DOS and the master environment in high memory, you must also be running MS-DOS 5.0 or above.

4DOS never accesses extended memory directly. It always uses an XMS driver like *HIMEM.SYS*, *EMM386.SYS*, 386MAX, QEXT, or QEMM. 4DOS can also access any RAM disk you create in extended memory by using a program like *VDISK.SYS* or *RAMDRIVE.SYS*. 4DOS does not use the XMS "High Memory Area" (HMA), a 64K byte area just above 1 MB on 80286, 386, and 486 systems.

If you want to know whether 4DOS sees your system's memory accurately, check the output of the MEMORY command. It should correspond to your computer's memory configuration.

The MEMORY command's output depends to some extent on your memory manager. If you are using Quarterdeck's QEMM, the report may not be the same as you expect. Because QEMM turns your extended memory into either XMS or EMS memory as required, the same memory is shown both ways in the MEMORY report. If 1 MB of extended memory managed by QEMM is available, MEMORY will report 1 MB of free XMS memory and

1 MB of free EMS memory as well, even though it is all the same memory. This is not a bug but a result of QEMM's flexibility.

Memory-related problems with 4DOS are usually due to programs which overwrite the extended memory block (EMB) that 4DOS uses for swapping its transient portion. When you exit from such a program, your system will hang, because 4DOS tried to swap itself back into base memory but its code and data in XMS have been destroyed by the program. The same problem can occur with EMS swapping but is less common because EMS memory is generally better defended against wayward programs. You can diagnose this kind of problem easily by changing to disk swapping with the *4DOS.INI* Swapping directive, and rebooting. If the problem goes away with disk swapping, then the program in question is probably destroying 4DOS's swap area in XMS or EMS memory.

4DOS EMS swapping sometimes has difficulty with EMS drivers which do not fully meet the EMS 3.2 specification (4DOS supports, but does not require, EMS 4.0 drivers). If you have trouble accessing EMS for swapping, check *APPNOTES.DOC* to see if there are any known problems with your EMS board or the associated driver software.

Video

4DOS is compatible with most display adapters and monitors. Although 4DOS can normally detect your video parameters automatically, you may have to configure it to use the system most efficiently.

4DOS uses two methods of displaying text on the screen:

- 4DOS calls DOS to write the text of prompts and normal messages. If you use an ANSI driver, DOS will transmit the calls to it. Otherwise, DOS will use your BIOS to display text on the screen. DOS text display calls will work on all DOS systems, regardless of video type.

- The DRAWBOX, DRAWHLINE, DRAWVLINE, LIST, SELECT, SCRPUT, and VSCRPUT commands bypass DOS, the BIOS, and any ANSI driver. They write directly to video memory. These commands will only work on systems with 100% IBM-compatible video systems. On other computers, results will be unpredictable at best. If you have such a system you probably know it already, because most application programs have similar problems.

EGA and VGA systems can display text in standard 25-line mode, plus modes with 43, 50, or more lines. 4DOS normally detects the number of lines automatically. If it doesn't, you can use the *4DOS.INI* ScreenRows directive or the SETDOS /R command to set the 4DOS screen length. 4DOS uses ScreenRows or SETDOS /R to set the display length that it uses for the LIST and SELECT command, as well as commands that have a "pause" option (TYPE /P, DIR /P, etc.).

4DOS never attempts to manipulate your video hardware in order to set the number of rows actually displayed on the screen (the "video mode"); to do so, you must use the software that came with your video board or other software tailored to your system.

4DOS does not handle display scrolling at the command prompt. If you put the screen in 43-line or 50-line mode and find that it still scrolls at the 25th line, your ANSI driver is probably not properly supporting your extended screen length. This is not a bug in 4DOS.

The video cursor shape that 4DOS uses is defined as a percentage of a character cell height. You can set the height independently for insert and overstrike mode with the CursorIns and CursorOver directives in *4DOS.INI* or with the SETDOS /S command. If you don't use either, 4DOS sets the height to 10% of the character cell height for overstrike mode and 100% (a block cursor) for insert mode.

If you have trouble with the cursor, use SETDOS /S to find the values that work for your system. Some video boards may not give a "smooth" response to varying SETDOS /S values. For example, a value of 20% may generate a very small cursor while a value of 30% may generate a half-height cursor. 4DOS can't do anything about this behavior, so you will have to experiment to find the cursor values that you want to use.

If the cursor disappears and you can't fix it with SETDOS /S, you probably have a screen color problem. The cursor is shown in the color of the underlying character cell. If that cell has (for example) the color attribute of black on black, the cursor will be invisible. If you are using an ANSI driver, you can fix this problem easily by clearing the screen to a known color with the CLS command. If the cursor still doesn't reappear, you will have to determine what is setting your screen attributes to an invisible color. For example, you might be using an ANSI driver that assumes a 25-line video mode on an EGA/VGA system running in 43-line or 50-line

mode. This can cause the driver to set portions of the screen to an invisible color when the screen is cleared.

If you are using an EGA or VGA adapter, we encourage you to try UltraVision from Personics Corp. It gives you excellent control over your video system, includes a wide variety of text-mode screen fonts, has its own ANSI driver, and works superbly with 4DOS. We use it and wouldn't be without it. A special version for laptop and notebook computers dramatically improves the readability of their smaller displays.

Most versions of DOS include a copy of *ANSI.SYS*, a device driver that is normally installed with a DEVICE= line in your *CONFIG.SYS* file. There are a number of more powerful and faster versions available as public-domain, shareware, and retail products. We use PC Magazine's free utility *ANSI.COM* because it can be enabled, disabled, loaded, and unloaded without rebooting, and because it is small and fast and works well inside windows of multitasking systems. It is available on most bulletin boards and online systems. Another excellent choice is *ANSI-UV.SYS* which is included with UltraVision.

4DOS normally detects automatically whether an ANSI driver is installed. If you have an ANSI driver installed and 4DOS doesn't recognize it, try the command SETDOS /A1 which forces 4DOS to use ANSI commands. Use SETDOS /A2 to tell 4DOS you do **not** have an ANSI driver installed. These options can also be set with the ANSI directive in *4DOS.INI*.

Hard Drives and Floppy Disks

4DOS uses your disks for a wide variety of purposes, and many 4DOS commands are designed to help you create, move, delete, view, and otherwise manage disk files. 4DOS never tries to manipulate the structure of your hard disk directly. It never modifies the FAT, root directory, or other system areas of the disk directly, and it doesn't read or write data on your disk itself. It always calls on DOS to perform these actions, just like most application programs do. As a result, 4DOS is compatible with all disk sizes, formats, and structures that your DOS version supports.

The most common question that we're asked about 4DOS and disks is whether 4DOS will handle a hard disk larger than 32 Megabytes. The answer depends on how your system is configured. Early versions of DOS

do not support hard disks over 32 Megabytes unless a disk partitioning driver like SpeedStor, Vfeature Deluxe, or Disk Manager is used. Later versions of DOS support large hard disks directly, without a partitioning driver. If your system supports large hard disks, either directly through DOS or with a partitioning driver, 4DOS will support them also. If your system doesn't support large hard disks, neither will 4DOS.

4DOS will generally access your disk very quickly, but the speed depends on what you are trying to do. If you find that 4DOS is slower at performing a particular function than you are used to, you may have asked it to do more than you ask of traditional DOS commands. In particular, if you use file descriptions, remember that 4DOS has to access the description file as well as the actual files that you are manipulating.

Some users notice that the common commands DEL and DIR appear slower with 4DOS under certain circumstances. With DEL, this slowdown may be because 4DOS uses a newer method of file deletion instead of an older method that is no longer recommended (but commonly used). The new method is necessary to enable 4DOS to display the names of the files you are deleting, and to support 4DOS's "extended wildcards" (see page 71). You can force 4DOS to use the older method with DEL's /Q option as long as you don't use extended wildcards.

For DIR, any perceived speed decrease is because of 4DOS's directory sorting. 4DOS must read all filenames before it can display any of them. The sort itself is quite fast, but DOS is relatively slow at retrieving the entire list of file names and passing them on to 4DOS. Once the 4DOS DIR display starts, it should go as fast as or faster than the traditional DIR display.

Laptop and Notebook Computers

4DOS makes a great addition to any laptop or notebook computer, but some of these systems have unusual characteristics which you must consider when you install 4DOS or make any change to your *CONFIG.SYS* file.

Many of these computers boot from a floppy drive or a hard disk just like a desktop computer. On these machines, you can generally install 4DOS just as you would on any other computer. However, some laptop and notebook computers, including many Tandy laptops, boot from ROM

("Read Only Memory" chips inside the computer). You need to take some precautions with these machines.

First, systems which boot from ROM often can be configured to use either the ROM boot feature or a standard floppy or hard drive boot up. The following comments only apply when you use the ROM boot feature.

! If your system allows you to boot from ROM but load *CONFIG.SYS* and *AUTOEXEC.BAT* from a hard disk or floppy, you need to be cautious. If you make a mistake in a hard-disk based *CONFIG.SYS* that keeps your system from booting, there may be no way to tell the ROM boot program to ignore the bad *CONFIG.SYS* file. You may have to take drastic measures like opening the case and disconnecting the hard drive to get the system to ignore your mistaken *CONFIG.SYS* and boot properly.

If you have a system that boots from ROM and reads *CONFIG.SYS* from the hard disk, we strongly recommend that you change the configuration to boot from the hard disk or a floppy before you make **any** change to your *CONFIG.SYS* file, whether it is related to 4DOS or not. This will allow you to boot from a boot-up floppy disk if you make an error in the *CONFIG.SYS* on your hard drive. Once you are satisfied that everything is working properly, you can switch back to ROM bootup.

The second, and related, issue with systems that boot from ROM is that they may consider the ROM to be a disk drive of sorts. A system with drive C as the hard disk may view the ROM as drive D. When you boot from ROM, the ROM drive is the current drive, and that is where 4DOS will look for the *4START* and *AUTOEXEC.BAT* files. But they will be on your hard drive or floppy diskette, and 4DOS won't be able to find them. (This doesn't happen with *COMMAND.COM* on these systems because the manufacturer has modified it to get around the problem.)

If you run into this problem, you can fix it easily with some changes to your *CONFIG.SYS* and *4DOS.INI* files. First, modify the SHELL= line in *CONFIG.SYS* so that it tells 4DOS where to find *4DOS.INI*. You can do this by placing the full path to *4DOS.INI* on the line, like this (see page 109 for more details):

```
shell=c:\4dos\4dos.com @c:\4dos.ini /p
```

(change the drive and path shown if *4DOS.COM* is not in the directory *C:\4DOS* on your system). This tells 4DOS to look for *4DOS.INI* on drive

C, even though the boot drive might be (for example) drive D. Then add two lines to the *4DOS.INI* file (see page 121 for details on these directives):

```
4StartPath=c:\
AutoExecPath=c:\
```

These tell 4DOS to look for *4START* and *AUTOEXEC.BAT* in the root directory of drive C, even though the boot drive may be different.

If you have a system like this, 4DOS may set the COMSPEC to the ROM drive. You can avoid this by setting the COMSPEC yourself on the SHELL= line in *CONFIG.SYS* (see page 108). For example, the SHELL= line above could be modified to read:

```
shell=c:\4dos\4dos.com c:\4dos @c:\4dos.ini /p
```

The second "C:\4DOS" tells 4DOS to use this directory as the COMSPEC path.

Finally, if you have a laptop or notebook computer with a color (CGA, EGA, or VGA) video board and a monochrome screen, you may need to use the HELP /M option, the HelpOptions directive in *4DOS.INI*, or run *HELPCFG* to adjust the HELP colors.

Critical Errors

A "critical" error is an error that gives you the "Abort, Retry, Fail" message. With 4DOS running, this message appears as follows:

```
[Error message]
R(etry), I(gnore), F(ail), or A(bort)?
```

The error message on the first line explains the error that has occurred, and the device on which it occurred. The second line prompts for your choice of action (the Fail choice will not be displayed under DOS 2).

A critical error usually indicates a hardware malfunction. The error may be that the device doesn't exist, there is no disk in the drive, the network has gone down, or a data error occurred. In most cases you will choose R to retry the operation, or A to abort the operation.

! Choosing I(gnore) can be risky: it will cause 4DOS to proceed as if the error had not occurred. This can produce additional errors, and may lead

the command which generated the original error to perform its functions improperly.

F(ail) will tell 4DOS that the operation it was attempting has failed, which will generally produce another error message. For example, if you attempt to do a directory on drive A with no disk in the drive, and answer F to the resulting critical error, you will get an additional error message, "Invalid drive A".

Many programs install their own critical error handlers. If you get a critical error message within an application and the second line does not read as shown above, the message did not come from 4DOS. Any problem in handling your response properly is due to the application, not to 4DOS.

Choosing A(bort) within an application may abort the entire application and not just the operation being performed. The specific action depends on which critical error handler is in use (4DOS's or the application's), and how the application's critical error handler (if any) is designed.

If you are using 4DOS on a system that must run unattended (for example, a bulletin board), you can use the /F startup option in *CONFIG.SYS* or the CritFail directive in *4DOS.INI* (see pages 108 and 131) to provide an automatic F(ail) response to all critical errors. However, we do **not** recommend this on a normal system, because you will not have a chance to react to a critical error and correct the problem that caused it.

Software

You should find that 4DOS is compatible with all your PC software. We have designed it carefully so that it uses only standard, documented methods to do its job. It works properly with application software, utilities, networks, multitaskers and task switchers, memory-resident (TSR) programs, and system software like disk caches, memory managers, and device drivers. We test 4DOS regularly with hundreds of popular software products in order to catch and correct compatibility problems before you encounter them.

The following sections discuss using 4DOS with two major kinds of software: multitasking and task switching programs, and networks. For specific information about any individual software package, including the latest information about products mentioned here, see the

APPNOTES.DOC file distributed with 4DOS. It contains the latest information we have available when your copy of 4DOS was shipped.

If you need to diagnose a problem that isn't covered below or in *APPNOTES.DOC*, see page 7.

4DOS and DOS

4DOS is compatible with all versions of MS-DOS and PC-DOS from 2.0 through 5.0 and above, and with DR-DOS 3.4, 5.0, and above. For MS-DOS and PC-DOS users, we recommend the use of DOS 3.1 or above. Some specific considerations for DR-DOS users are discussed in *APPNOTES.DOC*. MS-DOS users using the APPEND command may need to set up some aliases to invoke APPEND; see *APPNOTES.DOC* for details.

If you use the FORMAT /S command from MS-DOS or PC-DOS, version 4.0 or above, FORMAT will copy the file pointed to by the COMSPEC environment variable (see page 113) and name it *COMMAND.COM*. In most cases this means that *4DOS.COM* will be copied to the floppy disk, but with the name *COMMAND.COM*. Such a disk should boot properly and start 4DOS, but its contents is sure to be confusing to others. If you use FORMAT /S with MS-DOS or PC-DOS 4.0 or above, we recommend that you copy *COMMAND.COM* manually to the floppy disk (you can use an alias or batch file if you format bootable disks frequently), or rename the file that FORMAT copies to *4DOS.COM* and place a proper *CONFIG.SYS* file for 4DOS on the floppy disk.

Using 4DOS with Task Switchers and Multitaskers

Task switchers are programs that allow you to switch quickly among multiple applications, with one application running at a time. Multitaskers are more complex programs which run multiple applications at the same time, with one or more programs executing "in the background" while you work with another program on the screen.

For convenience, in the text below we will refer to both multitaskers and task switchers as "multitaskers," and to each window or partition they use as a "window," even though some do not have windowed displays.

4DOS works well as both the primary shell (loaded when your system boots) and the secondary shell (loaded in a window) with most multitaskers.

Most multitaskers have a pre-configured "DOS" window. In some programs, this window always runs *COMMAND.COM*. Others run whatever program COMSPEC points to, which means they will run 4DOS if you boot up with 4DOS. We recommend that you always set up a 4DOS window explicitly, with the configuration you want, rather that relying on the multitasker's generic "DOS" window.

Many multitaskers also run the command processor when you start certain kinds of windows, such as windows that run a *.BAT* file. In general, this use of the command processor is transparent. The multitasker will run 4DOS for you automatically when it needs to, and you won't need to do anything about it.

If you find that your multitasker is running *COMMAND.COM* when you meant to run 4DOS, check the COMSPEC setting that is in effect when you start the multitasker. You may also need to check the way a particular window is configured.

When you set up a 4DOS window, be sure to specify the full path to *4DOS.COM*, and any command-line options you want (see page 111 for information on command line options). To set parameters (swapping, alias space, etc.) to be used by all 4DOS secondary shells run by your multitasker, use the **[Secondary]** section in *4DOS.INI* (see page 117). To set these parameters separately for a specific window, create a copy of *4DOS.INI* just for that window and use the **@d:\path\inifile** option on your command line for the window to tell 4DOS where to find *4DOS.INI*. To change the configuration of a specific window without creating a separate copy of *4DOS.INI*, use the **//iniline** option on your command line for the window (see example below).

4DOS allows you to place a command to be executed as the last parameter on your 4DOS command line. This command is executed before 4DOS displays its first prompt. You can use this feature to run a batch file (or any other command) each time a 4DOS window is started by your multitasker. For example, if you are setting up 4DOS to run as a DOS application under Windows 3.0, your command line might look this:

```
c:\4dos\4dos.com //swapping=f:\ c:\winstart.btm
```

This tells Windows to load 4DOS, includes a *4DOS.INI* directive to tell 4DOS to swap to drive F, and passes 4DOS the command *C:\WINSTART.BTM*. You can place commands in *C:\WINSTART.BTM* to be executed whenever such a window is started (for example, to change your PROMPT to show that you're in a window, or to load a TSR for just that window). The command to be executed (*C:\WINSTART.BTM* in this example), must be the **last** thing on the 4DOS command line; no 4DOS switches or options can be placed after it because anything after the command will be interpreted as parameters for the command.

This command feature is similar to what's provided by the *4START* batch file, but *4START* is executed **every** time 4DOS loads, whereas a file like *WINSTART* will be executed only when a 4DOS window is started from your multitasker. A batch file started this way will be run after *4START*.

Multitasking and Disk Swapping

When 4DOS is loaded as the primary shell, it acts as a "traffic cop" for copies of the transient portion of 4DOS swapped to disk. Each secondary shell is assigned a unique shell number, which is used as the extension of its disk swap file (*4DOSSWAP.001*, *4DOSSWAP.002*, etc.). These shell numbers avoid file name conflicts between multiple copies of 4DOS running in different windows but creating swap files in the same disk directory.

However, if 4DOS is **not** loaded before the multitasker, this capability will not be available. In this case, the copy of 4DOS in each window will use a swap file called *4DOSSWAP.000*. To avoid a conflict in this situation, you **must** force every copy of 4DOS to place its swap file in its own unique directory by using the "Swapping=d:\path" directive in *4DOS.INI*. If you don't follow this rule, your system will hang when you switch windows or when you exit from an application.

This problem will occur only in those rare situations where 4DOS is loaded within a window but is **not** loaded as your primary shell, **and** if 4DOS disk swapping is used in more than one window at a time. The problem will not occur if 4DOS is loaded as the primary shell (the usual case), or if 4DOS can use EMS or XMS swapping for all simultaneous shells. Note that since the default swapping option uses disk swapping if insufficient EMS or XMS memory is available, you can be invoking disk swapping in your multitasker's windows without specifically requesting it.

4DOS and Microsoft Windows 3.0

4DOS works well as both the primary shell, loaded before Windows 3, and as a secondary shell loaded inside any window. It works in any Windows mode (Real, Standard, or Enhanced). The previous general information about multitaskers applies to Windows as well. You should read it before continuing with this section.

To run 4DOS as a secondary shell from within Windows, you will need to create a Program Manager icon to run 4DOS. The generic "DOS" icon supplied by Microsoft will only run *COMMAND.COM*. You can set up a 4DOS icon from the Program Manager's File / New menu selection.

First, create a new program item and set the command line to "C:\4DOS\4DOS.COM" (use the appropriate drive and path for your system). As discussed above, you can put the name of a batch file at the end of the command line.

To install a special 4DOS icon, use the Program Manager's File / Properties menu selection. Click on the Change Icon button and type in the full path name of your new 4DOS icon file. We supply two Windows icons with 4DOS: *4DOS.ICO* for color displays, and *4DOSM.ICO* for monochrome displays. Of course, you can create your own with any icon editor.

For more flexibility, you can use the Windows PIF editor to create a *4DOS.PIF* file. We have included a sample *.PIF* file on the distribution diskette. You must edit this file and make it correspond to your system before you use it to run 4DOS.

If you run Windows in 386 Enhanced mode, 4DOS will work properly in either a full-screen or a windowed session. The *.PIF* file determines the mode that 4DOS will start in. If you don't use a *.PIF* file, 4DOS will always start in full-screen mode.

Your batch files can determine whether they are running in a secondary shell under Windows, and the current Windows mode, with the 4DOS _WIN environment variable (see page 83).

You can easily set up the Windows File Manager so that it will consider *.BTM* files to be "executable". Open your *WIN.INI* file with any editor and

find the section labeled "**[extensions]**". Add the following line to the end of the section:

```
btm=c:\4dos\4dos.com /c ^.btm
```

(adjust this to show the proper path for 4DOS.COM on your system). It is **not** possible to execute *.BTM* files from the Program Manager by modifying the Programs= setting in *WIN.INI*; if you try to do so, the system will hang when you attempt to actually invoke a BTM file.

4DOS and DESQView

4DOS works well as both the primary shell loaded before DESQView, and as the secondary shell loaded inside any DESQView window. The previous general information about multitaskers applies to DESQView as well. You should read it before continuing with this section.

To use 4DOS as a secondary shell with DESQview, you must add it to your DESQview "Open Window" menu. To do this, select the Add a Program option, then press the "O" key (for Other Program). Press Enter and you will get an Add a Program window. You'll need to modify settings on the standard first screen, and on the second "advanced options" screen. Set the Program Name to *C:\4DOS\4DOS.COM* (adjust the drive and path for your own computer). Set the Parameters to whatever 4DOS startup options you want, but do not use /C or /P. For other DESQView parameters, the defaults are workable with the following changes:

To run 4DOS in a full-screen window:

Writes Text Directly to Screen:	Y	(screen 1)
Virtualize Text / Graphics:	N	(screen 1)
Close on Exit to DOS:	Y	(screen 2)
Uses its Own Colors:	Y	(screen 2)

To run 4DOS in a window smaller than the full screen:

Writes Text Directly to Screen:	N	(screen 1)
Virtualize Text / Graphics:	Y or T	(screen 1)
Close on Exit to DOS:	Y	(screen 2)
Uses its Own Colors:	Y	(screen 2)

4DOS is written to be "DESQView-aware", and will not "bleed through" to other windows when running full-screen commands such as HELP, LIST, or SELECT.

If you use 4DOS commands that work with an ANSI driver (CLS, COLOR, and the COLORDIR environment variable), you will need to load the ANSI driver in your 4DOS window. Drivers like PC Magazine's *ANSI.COM* may work in full-screen windows, but we've found that the only ANSI driver which works properly in a window smaller than the full screen is Quarterdeck's *DVANSI.COM*.

You can set up a startup batch file to load your ANSI driver or take other actions when a DESQView 4DOS window is opened, as discussed in the general section on multitaskers above. Just place the batch file name (with drive and path if necessary) as the last thing on the Parameters line.

DESQView will work properly with the UMBLoad and UMBEnvironment directives set to Yes for the primary shell in *4DOS.INI*, but may not work properly if these directives are also active in secondary shells. 4DOS will turn these directives off by default when loading a secondary shell under DESQView, but you can override this default action with any directives you explicitly place in *4DOS.INI*. If you have trouble with secondary shells and are using either of these directives, try placing the following lines in the **[Secondary]** section of *4DOS.INI*:

```
UMBLoad = No
UMBEnvironment = No
```

If you want to use DESQView's DOS Services program, check *APPNOTES.DOC* for the details on how to set it up for use with 4DOS.

Using 4DOS on a Network

4DOS works well with DOS-compatible networks. This section will give you some tips on using 4DOS properly on a network, and on the locations to use for 4DOS files on a network.

In general, you'll find that you can load and run your network software normally under 4DOS. Network drives will be accessible as normal drives once the network is loaded, and files on the network will be accessible just as if they were on a local hard disk.

Some networks support file and directory names beginning with a double backslash, or with a server name followed by a colon, to identify files by their location on the network. 4DOS detects such names and passes them through to the network unaltered, allowing the network software to process them properly.

In rare situations, you may have trouble loading network software under 4DOS. To the best of our knowledge, all DOS-compatible networks do work with 4DOS. If yours doesn't, our experience suggests that the most common cause is a network bug, an old version of your network software, or a conflict in the way 4DOS and your network are configured. Most bugs have now been corrected by network vendors, and should not appear on your system. If you have any questions about compatibility with your particular network, first check for a listing in *APPNOTES.DOC*; then of course feel free to contact our technical support department for assistance.

If you need to boot a diskless workstation from a network drive, you must have the network drive accessible at boot time. If this condition is satisfied (so 4DOS can find its files on the network drives), the normal approach can be used to start 4DOS from the network.

Some networks with large server disk drives (256 MB or more) may report values that are too small if the FREEcommand and the %@DISKFREE, %@DISKTOTAL, and %@DISKUSED variable functions are used for the server drive. If this occurs, it is because the network software does not provide a way to return larger values to 4DOS.

! When you use 4DOS on a network, pay attention to where files are stored in order to ensure that two 4DOS users do not attempt to access the same 4DOS file at the same time. You will need to pay particular attention to disk swapping and pipes.

If 4DOS uses disk swapping (either because of an explicit directive in *4DOS.INI* or because default swapping is used and no EMS or XMS memory is available), you should be sure that two users don't use the same directory simultaneously for their disk swap files. If they do, the filenames (*4DOSSWAP.000*, *4DOSSWAP.001*, etc.) will conflict and each user will write over the other's files, possibly causing one or both systems to hang. To take care of this, use the Swapping directive in *4DOS.INI* to assign each user's swap files to a different directory on the network drive.

Pipes are a method of passing information from one program to another, and are invoked with the pipe symbol [|] on the command line (see page 66). Pipes work by taking the output from one program, storing it in a temporary file, then telling a second program to obtain its input from that file. The temporary files, with names like P1.$00 and P2.$00, are placed by default in the root directory of the boot drive, but can be placed on a different drive and directory by setting the TEMP or TEMP4DOS environment variable (see page 113).

The same cautions given for disk swapping must be followed for pipe temporary files; that is, you must ensure that each user's temporary pipe files go to a separate directory. To do this, just be sure that each network user running 4DOS has TEMP or TEMP4DOS set to a unique directory. If you boot 4DOS from a local hard disk, the pipe temporary files will go to that disk and the environment variable setting will not be necessary.

Solving Software Compatibility Problems

Any DOS program running on your computer can potentially interact with any other program running at the same time. Of course, most program interactions are ones you want: your print spooler intercepts printer output and saves it to print later, or your disk cache intercepts disk requests and speeds them up by retrieving data from memory.

If you've used the PC for any length of time, however, you'll know that you can also get interactions you don't want. If you load just the wrong combination of TSRs and device drivers, your system may slow to a crawl. Perhaps you can't load your favorite Personal Information Manager with Windows running. And so on.

As publishers of a product that replaces part of the operating system, we're very familiar with these issues -- not because 4DOS is more likely to cause problems, but because it sometimes gets blamed first when a problem appears. Our technical support department has developed a set of reliable techniques for finding out what's causing an apparent compatibility problem with 4DOS and other software.

We are presenting these techniques here as a series of things to try when there seems to be a compatibility problem. Some may not make sense for the particular problem you're investigating. Others may not yield useful

results. But as a group, they'll help you resolve many of the common software interactions that do appear, whether with 4DOS or anything else.

Some of our suggestions help you figure out what's going on, but aren't intended to help you fix it. For example, when we suggest that you remove all your TSRs to look for the problem, we aren't suggesting that as a permanent solution, but only as a diagnostic test. Before you get started, be sure to check *APPNOTES.DOC* to see if we've already solved the problem you're facing.

The first thing to consider is whether the particular combination of software that's not working used to work together. If so, think carefully about what you have changed and see if reversing the change solves the problem. If it does, then you can narrow your search, using the following techniques to find out what it is about that specific change that is causing the problem.

Second, make sure that your problem can be reproduced relatively easily, and make sure you know exactly what sequence of commands or other steps reproduces it. Most interactions are very easy to reproduce, but if you think there's an interaction and it occurs once every 10 days, it's going to be difficult to know when you have fixed it. Also, the process of carefully documenting how to reproduce a problem often helps you realize what the problem is without further effort.

Third, if you have a problem with a specific application hanging or working improperly, try cleaning up the "atmosphere" in which that program runs. This is the single most useful tool we know for finding compatibility problems. By "cleaning up the atmosphere" we mean all of the following, and any other similar things you may be able to think of about your particular system after reading our suggestions below:

> Check the length of your PATH variable. 4DOS lets you make it longer than the standard limit of 123 characters. Some programs can't handle long PATHs and may behave strangely. If your PATH is over the normal limit, reduce its size using the PATH or ESET command and see if the application starts working. If so, use a batch file or alias to set up an alternate path for running that one program, for example:
>
> ```
> setlocal
> path d:\myprog
> d:\myprog\myprog.exe
> ```

```
endlocal
```

The SETLOCAL / ENDLOCAL pair saves and restores the environment; when you're done, the old PATH will be restored automatically (see page 284).

Next, check how much environment space is in use in your system. The 4DOS MEMORY command reports the total environment space and the amount free; a simple subtraction tells you how much is in use. Some programs use outdated techniques and simply don't work right if there's a lot of information in the environment (these programs don't usually care how big the total environment space is, only how much of it is actually in use). In most cases, these problems show up when the amount of space in use gets up to around 1K (1024) bytes or so, but they can occur at any point. To test for this, use the following simple batch file:

```
setlocal
unset var1 var2 var3 ...
[command to run the program in question]
endlocal
```

where VAR1, VAR2, etc. are variables you can remove from the environment to decrease the space in use before running the program. If reducing the environment space in use makes things work, contact the program's manufacturer and report the problem. You have found a legitimate bug. DOS allows an environment of up to 32K and all programs should be able to work with an environment that large. Until the manufacturer fixes the bug, use the batch file above as a workaround.

Next, look for a multi-program interaction. Remove all the device drivers and TSRs you possibly can and still have enough software present to demonstrate the problem. For example, you can't look for a network problem if you don't load the network, but you probably can check it without your disk cache running. When you do this, and any time you modify your boot configuration, be sure you have a bootable floppy disk handy.

If you run a partitioning disk driver like SpeedStor, Vfeature, or Disk Manager, you probably can't remove it for diagnostic purposes without temporarily losing access to some or all of your hard disk. The same may be true of disk compression programs like Stacker,

depending on the mode in which they are installed. Most other device drivers and TSRs can be removed without causing trouble. Check your system and software manuals if you are unsure of which programs can safely be removed.

Once you know what you can take out, don't skimp or guess where the interaction might be. Take out **everything** you possibly can from *CONFIG.SYS*, *4START*, and *AUTOEXEC.BAT* that loads or accesses another program . In *CONFIG.SYS*, remove all possible DEVICE and INSTALL statements. In *AUTOEXEC.BAT*, remove all the lines you can that load memory-resident programs (and remember that some DOS utilities, like MODE, can be memory-resident). Of course, save copies of your configuration files before you delete anything. Better yet, use the REM command to remove lines temporarily without deleting them. REM can be used on any line in *AUTOEXEC.BAT*, in *4START*, and in *CONFIG.SYS* if you are running DOS 4.0 or above. In earlier versions, REM will work in *CONFIG.SYS* but will also generate a harmless "unrecognized command" error message during bootup. If you want to remove everything in *AUTOEXEC.BAT* you can simply rename it to any other name, and rename it back when you are done testing.

Clean out your configuration files all at once, not one line at a time. If that solves the problem, you're on the right track, and you can put the lines back one at a time until you find the culprit. If it doesn't solve the problem, you won't waste time removing lines one by one.

If you do find a suspect program, first try booting your system with *COMMAND.COM*, without changing anything else about your configuration. If the problem remains, then it's not related to an interaction with 4DOS.

If the problem isn't there under *COMMAND.COM*, try fiddling with the program's configuration. If you were loading it high, try loading it low. If you can change the way it uses memory, try doing so. If it's a driver that's used by other programs (like your mouse driver) and is quite old, consider obtaining an update from the manufacturer. All of these techniques will help you narrow down what it is about the program that's causing a problem. Once you have done that, you may have a simple workaround. If not, contact

our technical support crew and we'll try to verify the problem, then resolve it with the manufacturer of the other software.

Rarely, some problems can be resolved by modifying the order in which you load drivers and TSRs. If you've found a problem with a particular driver or TSR, try loading it earlier or later than you were and see if the problem goes away.

Next, try modifying the atmosphere in another way: change the way 4DOS is configured. In particular, try changing the 4DOS swapping type using the Swapping directive in *4DOS.INI* (see page 122). This technique is especially appropriate if the system hangs every time you exit a particular application. If that solves the problem, there's probably a memory conflict. Another program is trying to use the same memory space 4DOS uses for swapping. See if you can control the other program's memory usage.

If you can configure 4DOS and the other program to work together, you're all set. If they work together only in a useless combination (for example, with 4DOS swapping turned off), contact us. We'll try to figure out what the other program is doing to damage 4DOS's swap space and get the manufacturer to take care of the problem.

You can also change the UMB-related configuration settings (UMBLoad and UMBEnvironment) to help diagnose compatibility problems. Problems with these directives are rare, but if you're at an impasse, try setting both of these values to "No" in *4DOS.INI*.

Some of the advanced directives in *4DOS.INI* (see page 131) may help solve very rare configuration problems, but unless you are an experienced DOS user and understand the side effects of each directive, they should be used only as diagnostic tools, and not as a workaround or fix. Any of the following can be tried:

```
Inherit = No
LineInput = Yes       (or SETDOS /L1)
Reduce = No
StackSize = nnnn      (increase value to 4096 or more)
```

If you've tried all these techniques and haven't found the problem, contact our technical support department (see page 7). We have more tricks up our sleeve, and a very high success rate at resolving software interactions.

CHAPTER 8 / COMMAND REFERENCE GUIDE

The following pages are a complete guide and reference to the 4DOS commands that are available from the command line, in aliases, and in batch files. Nearly all of these commands are *internal* 4DOS commands, which means that 4DOS performs the activity you have requested without running another program. *External* commands require loading and running a separate program, either an executable (*.EXE* or *.COM*) program or a batch (*.BTM* or *.BAT*) program. DOS is shipped with a number of external utility programs (such as FORMAT and DISKCOPY), and any program or application you install on your system becomes a new external command.

The advantage of internal commands is that they run almost instantly. When you give 4DOS an internal command, it interprets the command line and carries out the necessary activities without having to look for, load, and run another program.

The advantage of external commands is that they can be large, varied, and complex without taking space inside the system command processor. External commands can also be renamed or replaced easily. If you want to rename the external DOS command XCOPY to MYCOPY, for example, all you need to do is find the file called *XCOPY.EXE* on your DOS disk or directory and change its name to *MYCOPY.EXE*. If you want to replace XCOPY with a more efficient program, you can do so. 4DOS adds this flexibility to internal commands. You can rename or replace any internal command by using an ALIAS, and you can enable or disable internal commands whenever you wish.

4DOS Commands

4DOS has over 80 internal commands, many more than any version of DOS. 4DOS neither replaces nor interferes with external DOS commands like ASSIGN, BACKUP, CHKDSK, DISKCOPY, SUBST, or XCOPY. Once 4DOS is installed, you can continue to use those utilities like you always have. Also, 4DOS has been designed so that it is compatible with virtually all traditional internal commands, even though it enhances many of those commands with additional options and capabilities. Once you have installed 4DOS, you can continue using the commands that you already know and get the same results.

A few of the 4DOS commands are the same as traditional commands, some are enhanced with new features, and many are unique to 4DOS. The best way to learn the 4DOS commands is to use them and experiment with them. The following lists categorize the available commands by topic and will help you find the ones that you need. Most of this chapter is an alphabetic list of the commands and how to use each one. We urge you to browse through this chapter occasionally and look for commands that might help simplify your computing life.

In the following summary lists, commands that are unique to 4DOS are marked with a number sign [#]. Those which are enhanced traditional commands are marked with an asterisk [*]. And those which are identical to traditional commands have no marks at all.

System Configuration Commands:

BREAK	CHCP	CTTY	DATE
FREE #	LH / LOADHIGH	MEMORY #	PROMPT *
REBOOT #	SETDOS #	SWAPPING #	TIME
VER	VERIFY	VOL *	

File and directory management:

ATTRIB *	COPY *	DEL / ERASE *	DESCRIBE #
MOVE #	REN / RENAME *	TRUENAME #	

Subdirectory management:

CD / CHDIR *	CDD #	DIR *	DIRS #
MD / MKDIR *	POPD #	PUSHD #	RD / RMDIR *

Commands normally used in batch files and aliases (many are also useful at the command line):

ALIAS #	BEEP #	CALL	CANCEL #
COLOR #	DELAY #	DRAWBOX #	DRAWHLINE #
DRAWVLINE #	ECHO *	ECHOS #	ENDLOCAL #
GOSUB #	GOTO *	FOR *	IF *
IFF #	INKEY #	INPUT #	KEYSTACK #
LOADBTM #	PAUSE *	QUIT #	REM *
RETURN #	SCREEN #	SCRPUT #	SETLOCAL #
SHIFT *	TEXT #	TIMER #	UNALIAS #
VSCRPUT #			

Other commands:

? #	CLS *	ESET #	EXCEPT #
EXIT *	GLOBAL #	HELP #	HISTORY #
LIST #	LOG #	PATH *	SELECT #
SET *	TEE #	TYPE *	UNSET #
Y #			

As you can see, most 4DOS commands are either enhanced traditional commands or are entirely new. If you are comfortable using traditional commands, you can switch to 4DOS without making any changes in your habits. But you will be missing a lot of the power of 4DOS's enhancements and new commands unless you take a few minutes to see what's available here. Make sure you don't skip a section of this reference just because you already know how to use a traditional command with the same name.

We have made no attempt to document external DOS commands in this reference, partly because they are explained in your DOS manual, and partly because the number and name of DOS external commands, and the options available with each command, vary widely from one version of DOS to another and from one computer manufacturer to another. The 4DOS HELP system does include information on standard DOS external commands.

If you come across terms or concepts in this chapter that you are unsure about, please refer to Chapter 4 / General Concepts, the Glossary on page 325, or the Index.

How to Use the Command Descriptions

Each of the 4DOS commands is described in detail on the following pages. The descriptions are arranged alphabetically, and each includes examples that will help you learn to use the commands.

Each description begins with the **name** of the command on the left side of the page and a word in parentheses on the right side. The commands marked "New" are unique to 4DOS. Those marked "Enhanced" are similar to traditional commands but add new features and options. The commands marked "Compatible" follow the syntax and features of the traditional command with the same name.

The name is followed by a sentence or two that briefly describes the command's **purpose** or major function. That sentence should help you determine quickly whether you have found the command you are seeking.

The next part of each description shows the command's **format** or syntax. The format line uses certain conventions to describe how the command should be entered and to create reference points for the text describing the command:

Words in **UPPER CASE** must be spelled exactly as they are shown (although you can type them in using either upper or lower case, or a combination). If a word is shown partly in upper case (for example **BLInk**), only the upper case portion is required, the rest is optional.

Words shown in ***italics*** (for example ***source*** or ***filename***) are meant to be replaced by other words or values. Each of these words is explained directly beneath the format line, and discussed in more detail in the text description of the command. When the word stands for a file name, the name may be a simple file name like *MYFILE.TXT* or it may include a drive letter and/or a full path, like *C:\MYDIR1\MYDIR2\MYFILE.TXT*. If the command can work on multiple files, you can use the 4DOS wildcards, multiple file names, or an include list (see pages 71, 73, and 74).

Anything followed by an ellipsis (three periods **[...]**) may be repeated as often as you wish.

Text shown in **[square brackets]** is optional. Text outside of square brackets must be entered literally (if it is capitalized) or replaced by other words or values (if it is in italics).

Vertical bars | represent a choice; you can pick one option or another but not both. For example, the following format shows that the command may be followed by the word ON or the word OFF, but not both:

```
COMMAND [ ON | OFF ]
```

A slash followed by a letter, like **[/X]**, is an "option" or "switch" which controls the effect of a command. Many commands have several switches, and you are usually free to use none, one, or several to make a command behave as you wish. If you use a single switch,

you must precede it with a slash. If you use several switches, in most cases you can put them together with one slash or use separate slashes. For example, if you wanted to use switches X, Y, and Z for a command, you could type them three different ways:

```
command /x /y /z
command /x/y/z
command /xyz
```

A few switches, particularly in the DIR and SELECT commands, use two or more characters. If you need to follow a multi-letter switch with another switch, the second switch must have its own slash to avoid ambiguity.

Included in the format section is an explanation of each replaceable argument and a one or two word explanation of each switch. Many descriptions also list related commands to help you find the exact command you want.

After the command format, you'll find a description of the command's **usage**. This description normally starts with the basic functions of a command and gradually adds more details. We've also included many examples to help you see the command in action. In the examples, characters in **bold** type represent input from the user. Characters in normal type represent 4DOS prompts or responses, or lines in a batch file.

The last part of each description is a detailed explanation of the **options** or switches available for each command, in alphabetical order. Occasionally, we've included more examples in this section to demonstrate how a switch is used or how multiple switches interact.

Pay careful attention to the information about switch placement. Some switches have different effects based on where they appear in the command line. The effects of switches occasionally vary from one command to another in order to retain compatibility with traditional commands.

In the Usage and Options sections you may see the symbol ❖. This indicates a more in-depth discussion or an advanced topic which you can skip if you are new to the command; come back to this topic later for more details, or if you're having trouble with the command. In most cases the remainder of the section after such a symbol is devoted to similar information.

The ❖ doesn't mean that only advanced users will need the information -- you may find it useful even if you're relatively new to computers or to 4DOS. But it does mean that you can skip the marked section and still understand and use the basic features of the command. If a ❖ appears before the "Usage" heading, it indicates that the entire command is generally used only in unusual situations or by more advanced users.

When you see a ❖ in the list of options, remember that the options are listed alphabetically, so there may be more basic options discussed later in the list, after a more complex or advanced option marked with ❖. Don't stop reading the option list the first time you see the mark.

A ! to the left of a paragraph means that paragraph contains a caution or warning you may need to observe when using the feature it discusses.

? (New)

Purpose: Display a list of 4DOS commands.

Format: ?

Usage: ? displays a list of 4DOS internal commands. For help with these commands, and with external DOS commands, see the HELP command.

When you use the ? command, you will see a display like this:

```
c:\> ?
?           ALIAS       ATTRIB      BEEP
BREAK       CALL        CANCEL      CD
CDD         CHCP        CHDIR       CLS
COLOR       COPY        CTTY        DATE
DEL         DELAY       DESCRIBE    DIR
DIRS        DRAWBOX     DRAWHLINE   DRAWVLINE
ECHO        ECHOS       ENDLOCAL    ERASE
ESET        EXCEPT      EXIT        FOR
FREE        GLOBAL      GOSUB       GOTO
HELP        HISTORY     IF          IFF
INKEY       INPUT       KEYSTACK    LH
LIST        LOADBTM     LOADHIGH    LOG
MD          MEMORY      MKDIR       MOVE
PATH        PAUSE       POPD        PROMPT
PUSHD       QUIT        RD          REBOOT
REM         REN         RENAME      RETURN
RMDIR       SCREEN      SCRPUT      SELECT
SET         SETDOS      SETLOCAL    SHIFT
SWAPPING    TEE         TEXT        TIME
TIMER       TRUENAME    TYPE        UNALIAS
UNSET       VER         VERIFY      VOL
VSCRPUT     Y
```

If you have disabled a command with the SETDOS /I command, it will not appear in the list.

ALIAS

(New)

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 the key to execute the alias.
value: Text (commands, etc.) to be substituted for the alias name.

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

See also: UNALIAS and page 94.

Usage: The ALIAS command lets you create new command names or redefine the 4DOS internal commands. It also lets you assign one or more commands to a single keystroke. An alias is often used to execute a complex series of commands with a few keystrokes or to create "in memory batch files" that run much faster than disk-based batch files.

For example, if you would rather type D instead of DIR /W you would use the command:

```
c:\> alias d = `dir /w`
```

Now when you type a single *d* as a command, 4DOS will translate it into a DIR /W command. The marks around DIR /W in the ALIAS command are back-quotes, below the tilde [~] on most PC keyboards. They are NOT single quotes ['].

An alias can represent more than one command. For example:

```
c:\> alias letters = `cd \letters ^ text`
```

creates a new command called LETTERS. The command first uses CD to change to a subdirectory called *\LETTERS* and then runs a program called *TEXT*. The caret [^] is the 4DOS

command separator and tells 4DOS that the two commands are distinct and should be executed sequentially.

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 or any replaceable parameters (which are discussed below), to prevent premature expansion of the arguments. You MAY use back quotes [`] around other definitions, but they are not required. To avoid confusion, we recommend that you always use back quotes around alias definitions you type at the command line or enter in a batch file.

Aliases may invoke internal 4DOS commands, external commands, or other aliases. (However, an alias may not invoke itself, except in special cases where an IF or IFF command is used to prevent an infinite loop.) The two aliases below demonstrate alias nesting (one alias invoking another). The first line defines an alias which runs a program called *WP.EXE* that is in the *E:\WP51* 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:\wp51\wp.exe`
c:\> alias w = `pushd c:\wp ^ wp ^ popd`
```

The second alias above could have included the full path and name of the *WP.EXE* program instead of calling the WP alias. However, writing two aliases makes the second one easier to read and understand, and makes the first alias available for independent use. If you rename the *WP.EXE* program or move it to a new directory, only the first alias needs to rewritten.

If you put an asterisk [*] immediately before a command in the *value* of an alias definition (the part after the equal sign), it tells 4DOS not to attempt to interpret that command as another (nested) alias. An asterisk used this way must be preceded by a space or caret [^] and followed immediately by an internal or external command name. The asterisk is used to make sure that 4DOS interprets the following word as the name of an internal or external command instead of as an alias which may have the same name. It also allows two popular uses of aliases.

By using an asterisk, you can redefine the default options for any 4DOS command. For example, suppose that you always want to use the DIR command with the **/2** (two column) and **/P** (pause at the end of each page) options. The following line will do just that:

```
c:\> alias dir = `*dir /2/p`
```

If you didn't include the asterisk, 4DOS would interpret the second DIR on the line as the name of the alias itself and attempt to 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 tells 4DOS to interpret the second DIR as a command but not as an alias.

An asterisk also helps you keep the names of internal 4DOS commands from conflicting with the names of external programs. For example, suppose you have a program called *LIST.COM*. Normally, the 4DOS internal command will run anytime you type LIST. But two simple aliases will give you access to both the *LIST.COM* program and the LIST command:

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

The first line above defines LIST as an alias for the *LIST.COM* program. If you stopped there, the external program would run every time you typed LIST and you would not have easy access to the internal 4DOS LIST command. The second line renames the internal LIST command as DISPLAY. The asterisk is needed in the second command to tell 4DOS that the following word means the internal command LIST, not the LIST alias which runs your external program.

Another way to understand the asterisk is to remember that when 4DOS processes a command it always checks for an alias first, then looks for an internal or external command, or a batch file (see page 43). The asterisk at the beginning of a command name simply tells 4DOS to skip over the usual check for aliases when processing that command, and 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, 4DOS won't try to

interpret that command 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 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
```

4DOS will expand the WHEREIS alias and process the command:

```
dir /sp myfile.txt
```

If you want to assign an alias to a keystroke, use the keyname on the left side of the equal sign, preceded by an at-sign [@]. For example, to assign the command DIR /W to the F5 key, type

```
c:\> alias @F5 = `dir /w`
```

Keynames must be in the form

```
[Prefix[-]]Keyname
```

The key prefix can be any one of the following:

No prefix	followed by F1 - F12
Alt	followed by A - Z, 0 - 9, or F1 - F12
Ctrl	followed by F1 - F12, Left, Right, PgUp, PgDn, Home, or End
Shift	followed by F1 - F12

The possible key names are:

A - Z	Left	PgDn
0 - 9	Right	Home
F1 - F12	PgUp	End

All key names must be spelled as shown, and can be abbreviated as long as the abbreviation is unique and unambiguous. Alphabetic keys can be specified in upper-case or lower-case. You cannot specify a punctuation key.

The prefix and key name can be together or separated by a dash, but cannot be separated by any other characters (including spaces). You can also define a keystroke alias by using "@" plus a scan code for one of the permissible keys (see Appendix B on page 311 for a list of scan codes).

When you define keystroke aliases, the assignments will only be in effect at the 4DOS 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 directive in your *4DOS.INI* file. See page 129 for instructions.

If you define a keystroke alias like this:

```
c:\> @f5 = `dir /w`
```

then, when you press the F5 key, 4DOS will enter the value, "dir /w" on the command line for you. You can type additional parameters if you wish and then press Enter to execute the command. With this particular alias, you can define the files that

you want to display after pressing F5 and before pressing Enter to execute the command.

If you want the keystroke alias to take action automatically without waiting for you to edit the command line or press Enter, you must end the *value* with a carriage return. To enter the carriage return into your alias use the 4DOS escape character (normally Ctrl-X, shown here as an up-arrow [↑]) **twice**, followed by an "r." For example, this command will assign an alias to the F6 key that uses the CDD command to take you back to the previous default directory:

```
c:\> alias @f6 = `cdd -↑↑r`
```

If you want to see a list of all current ALIAS commands, type

```
c:\> alias
```

You can also view the definition of a single alias. If you want to see the definition of the alias LIST, you can type

```
c:\> alias list
```

You can save the list 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 from *AUTOEXEC.BAT*, you can edit them with a text editor, reload the edited file with the ALIAS /R command, 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 should **NOT** use back quotes around the value, even if back-quotes would normally be required when defining the same alias at the command line or in a batch file.

To remove an alias, use the UNALIAS command.

- Alias commands can use command-line arguments or replaceable parameters like those in batch files. 4DOS numbers the command line arguments from **%1** to **%127**. It is up to the alias to determine the meaning of each argument. Arguments that are referred to in an alias, but which are missing on the command line, appear as empty strings inside the alias.

  The parameter **%n&** has a special meaning. 4DOS 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. Aliases do not have access to %0, which is used only in batch files.

  For example, the following alias will change directories, perform a command, and return to the original directory:

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

  When this alias is invoked as:

  ```
  c:\> in c:\comm mycomm /xmodem /2400
  ```

  the first replaceable parameter, **%1**, has the value "*c:\comm*." **%2** is "*mycomm*," **%3** is "*/xmodem*," and **%4** is "*/2400*". 4DOS expands the command line into these three separate commands:

  ```
  pushd c:\comm
  mycomm /xmodem /2400
  popd
  ```

- This next example uses the IFF command to redefine the defaults for SET only if no other options are entered on the command line. This ALIAS command should be entered on one line:

  ```
  c:\> alias set = `iff %#==0 then ^ *set /p ^
       else ^ *set %& ^ endiff`
  ```

  This example 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 replaceable parameters, 4DOS will delete command line arguments up to and including the highest referenced argument. For example, if you have an alias which refers only to **%1** and **%4**, then the first and fourth arguments passed to the alias will be used, the second and third arguments will be discarded, and any additional arguments beyond the fourth will be appended to the end of the expanded command after the *value* portion of the alias. If an alias uses no replaceable 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, 4DOS's internal variables, and 4DOS's variable functions. For example, you can create a simple command-line calculator this way (enter this on one line):

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

Now, if you enter

```
c:\> calc 5 * 6
```

4DOS will display

```
The answer is: 30
```

- Aliases created in the primary shell will be inherited automatically in a secondary shell. However, an alias created in a secondary shell will not be passed back to a primary or parent shell.

Options: **/P**(ause): 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.

/R(ead 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 should **NOT** be used 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 one after another on the command line, separated by spaces:

```
c:\> alias /r alias1.lst alias2.lst
```

ATTRIB

(New)

Purpose: Change or view file and subdirectory attributes.

Format: **ATTRIB [/D /Q /S] [+ | -[AHRS]] *files***

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

/D(irectories) **/S**(ubdirectories)
/Q(uiet)

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

Usage: Every file and subdirectory has 4 attributes that can be turned on (set) or turned off (cleared): *Archive*, *Hidden*, *Read-only*, and *System*. DOS sets the *Archive* attribute every time a file is updated or changed. Backup utilities often use this attribute to find files that have changed since the last general backup. DOS prevents programs from altering or erasing files that have the *Read-only* attribute set (but applications can change the attribute first and then update or erase the file). Files with *Hidden* and/or *System* attributes are normally not visible in directory listings.

The **ATTRIB** command lets you set or clear any attribute(s) for any file, group of files, or subdirectory. You can view file attributes by entering ATTRIB without specifying new attributes (i.e. without the **[+ | -[AHRS]]** part of the format), or with the **DIR /T** command.

For example, you can set the read-only and hidden attributes for the file *MEMO*:

```
c:\> attrib +rh memo
```

Attribute options apply to the file(s) that follow the options on the ATTRIB command line. The example below shows how to set different attributes on different files with a single command. It sets the archive attribute for all *.TXT* files, and changes *TEST.COM* to system and not modified:

```
c:\> attrib +a *.txt +s -a test.com
```

❖ You may know that DOS also supports "D" (subdirectory) and "V" (volume label) attributes. These attributes cannot be altered with ATTRIB; they are designed to be controlled only by DOS itself.

Options:

/D(irectories): If you don't use this option, ATTRIB will only modify file attributes. If you use the **/D** option, ATTRIB will also modify the attributes of subdirectories (yes, you can have a hidden subdirectory):

```
c:\> attrib /d +h c:\mydir
```

/Q(uiet): This option turns off ATTRIB's normal screen output. It is most useful in batch files.

/S(ubdirectories): 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

(New)

Purpose: Beep the speaker or play simple music.

Format: **BEEP [*frequency duration ...*]**

frequency is in Hertz (cycles per second).
duration is in 1/18th second intervals

Usage: BEEP generates a sound through your computer's speaker. It is normally used in batch files to signal that an operation has been completed, or that the computer needs attention.

Because BEEP allows you to specify the frequency and duration of the sound, you can also use it to play simple music or to create different kinds of signals for the user.

You can include as many frequency and duration pairs as you wish. No sound will be generated for frequencies less than 20 Hz, allowing you to insert short delays. The default value for *frequency* is 440 Hz; the default value for *duration* is 2.

This batch file fragment runs a program called *DEMO*, then plays a few notes and waits for you to press a key:

```
demo ^ beep 440 4  600 2  1040 6
pause Finished with the demo - hit a key...
```

The following table gives the *frequency* values for a five octave range (middle C is 523 Hz):

C	131	262	523	1046	2093
C#/Db	139	277	554	1108	2217
D	147	294	587	1174	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	1864	3729
B	248	494	988	1976	3951

BREAK

(Compatible)

Purpose: Display, enable, or disable Ctrl-C and Ctrl-BREAK checking.

Format: **BREAK [ON | OFF]**

Usage: The Ctrl-C and Ctrl-Break keys are used by many programs (including 4DOS) as a signal to interrupt the current operation. BREAK controls how often DOS checks to see if you've entered one of these keystrokes.

Normally, BREAK is turned off, and DOS only checks for Ctrl-C and Ctrl-Break keystrokes during normal 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 programs use 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. If you turn BREAK on, programs will run slightly slower than normal (the difference is not usually noticeable), but you will be able to break out of some programs more easily.

Turning BREAK on or off only affects when DOS detects Ctrl-C and Ctrl-Break and notifies the program you're running. Any program can choose to ignore these signals, in which case the BREAK setting won't affect that program's behavior. Also, any external program can change the BREAK setting on its own.

Type BREAK by itself to display the current BREAK status:

```
c:\> break
BREAK is OFF
```

Type BREAK plus ON or OFF to set the BREAK status:

```
c:\> break on
```

BREAK is off by default. You can change the default by adding the following line to your *CONFIG.SYS* file:

```
break=on
```

CALL

(Compatible)

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) without starting a secondary copy of the command processor. 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.

The following batch file fragment compares an input line to "wp" and calls another batch file if it matches:

```
input  Enter your choice:  %%option
if "%option" == "wp" call wp.bat
```

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

The current ECHO state will be inherited by a called batch file.

- A called batch file will return to the calling file after processing the last line in the called file, or when a QUIT command is executed. A called batch file should always return in this way, or terminate all batch files with CANCEL. Restarting (or CALLing) the original batch file from within a called file will prevent 4DOS from detecting that you've left the second file, and it may cause an infinite loop or a stack overflow.

CANCEL

(New)

Purpose: Terminate batch file processing.

Format: **CANCEL**

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.

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
```

CD / CHDIR

(Enhanced)

Purpose: Display or change the current directory.

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

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

See also: CDD, MD, PUSHD, RD, and CDPATH on page 114.

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

CD lets you navigate through the DOS 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 both the drive and directory together.

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. You can move to a sibling directory -- one that branches from the same parent directory as the current subdirectory -- with the command **CD ..\newdir**.

If you enter CD with no argument or with only a disk drive name, it will display the current directory on the default or named drive.

CD saves the current directory before changing to a new directory. You can switch back to the previous directory by entering **CD -**. (There must be a space between the CD command and the hyphen.) You can switch back and forth between two directories by repeatedly entering **CD -**. The saved directory is the same for both the CD and CDD commands.

- 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 default drive and directory (on the second drive) will not be changed.

- If CD can't change to the specified directory, it will look for the CDPATH environment variable. CD will append the specified directory name to each directory in CDPATH and attempt to change to that directory, until the first match or the end of the CDPATH argument. This lets you use CDPATH as a quick way to find commonly used subdirectories which have unique names. For example, if you are currently in the directory C:\WP\LETTERS\JANUARY and you'd like to change to C:\FINANCE\REPORTS, you could enter the command:

```
c:\wp\letters\january> cd \finance\reports
```

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

```
c:\wp\letters\january> cd reports
```

  and 4DOS will change to C:\FINANCE\REPORTS.

- DOS will not accept directory names longer than 64 characters. You must be sure that the complete directory name from the root to your deepest subdirectory fits within the 64 character restriction.

CDD

(New)

Purpose: Change the current disk drive and directory.

Format: **CDD** ***path***

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

See also: CD, MD, PUSHD, RD, and CDPATH on page 114.

Usage: CDD is similar to the CD command, except that it also changes the default disk drive if one is specified. CDD will change to the directory and drive you name. To change from the root directory on drive A to the subdirectory *C:\WP*:

```
a:\> cdd c:\wp
c:\wp>
```

You can change to the parent directory with **CDD ..**; you can also go up one additional directory level with each additional **[.]**. For example, **CDD ....** will go up three levels in the directory tree.

CDD saves the current drive and directory before changing to a new directory. You can switch back to the previous drive and directory by entering **CDD -**. (There must be a space between the CDD command and the hyphen.) You can switch back and forth between two drives and directories by repeatedly entering **CDD -**. The saved directory is the same for both the CD and CDD commands.

- If CDD can't change to the specified directory, it will look for the CDPATH environment variable. CDD will append the specified directory name to each directory in CDPATH and attempt to change to that drive and directory, until the first match or the end of the CDPATH argument. This allows you to use CDPATH as a quick way to find commonly used subdirectories which have unique names. For example, if you are currently in the directory C:\WP\LETTERS\JANUARY and you'd like to change to D:\SOFTWARE\UTIL, you could enter the command:

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

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

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

and 4DOS will change to D:\SOFTWARE\UTIL.

- DOS will not accept directory names longer than 64 characters. You must be sure that the complete directory name from the root to your deepest subdirectory fits within the 64 character restriction.

CHCP

(Compatible)

Purpose: Display or change the current system code page.

Format: **CHCP [*n*]**

n: A system code page number.

❖ Usage: Code page switching allows you to select different character sets for language support. To use code page switching, you must have an EGA or VGA display and MS-DOS or PC-DOS 3.3 or above.

If you enter CHCP without a number, the current code page is displayed.

```
c:\> chcp
Active code page: 437
```

If you enter CHCP plus a code page number, the system code page is changed. For example, to set the code page to multilingual:

```
c:\> chcp 850
```

Before using CHCP, you must first load the device drivers (in *CONFIG.SYS*); make sure the information file (*COUNTRY.SYS*) is available; load national language support (using the NLSFUNC command); and prepare the specified code page for the devices (using the MODE command with the CODEPAGE PREPARE option).

CHCP accepts one of the two prepared system code pages. An error message is displayed if a code page is selected that has not been prepared for the system.

See your DOS manual for more information on CHCP.

CLS

(Enhanced)

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

Format: **CLS [[BRIght] [BLInk] *fg* ON *bg*] [BORder *bc*]**

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

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

See also: COLOR.

Usage: CLS can be used to clear the screen without changing colors, or to clear the screen and change the screen colors simultaneously. These three examples show how to clear the screen to the default colors, to bright white letters on a blue background, and to bright yellow letters on a magenta background with a blue border:

```
c:\> cls
c:\> cls bright white on blue
c:\> cls bri yel on mag bor blu
```

As shown in the last example, color names plus the words BRIGHT, BLINK, and BORDER may be shortened to the first 3 letters.

If you use BRIGHT and/or BLINK, 4DOS will apply the appropriate attribute to the foreground characters (PC displays do not allow a bright or blinking background).

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

- If *ANSI.SYS* or a compatible driver is not loaded, the colors will not be "sticky" -- you may lose them after you run an application. If 4DOS thinks you have an ANSI driver loaded, it first tries an ANSI clear screen. If that doesn't work, 4DOS will call the BIOS to clear the screen. You can force 4DOS to recognize the ANSI

state with the SETDOS /A option (see page 280) or the ANSI directive in *4DOS.INI* (see page 124).

If your display accommodates more than 25 rows by 80 columns and CLS doesn't clear the whole screen, your ANSI driver probably does not support the large display size properly.

COLOR

(New)

Purpose: Change the default display colors.

Format: **COLOR [BRIght] [BLInk] *fg* ON *bg* [BORder *bc*]**

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

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

See also: CLS.

Usage: COLOR is normally used in batch files before displaying text. To set screen colors to bright white on blue, you can use either of these commands:

```
c:\> color bright white on blue
c:\> color bri whi on blu
```

As the examples show, you may shorten color names plus the words BRIGHT, BLINK, and BORDER to the first 3 letters.

If you have an ANSI driver (such as *ANSI.SYS*) installed, COLOR will not change anything on the screen. It will only set the default colors for subsequent screen displays.

If you are not using an ANSI driver, COLOR will change the display colors of every character on the screen. However, the colors will not be "sticky" -- you may lose them after you run an application.

❖ If you see odd characters like "[44;37m" when you try to set the screen colors, 4DOS probably thinks you have an ANSI driver loaded when you don't. Use SETDOS /A2, or ANSI = No in *4DOS.INI*, to tell 4DOS you have no ANSI driver.

COPY

(Enhanced)

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

Format: **COPY [/C /H /M /N/ /P /Q /S /R /U /V]** ***source*****[+] ... [/A /B]**
destination **[/A /B]**

source: A file or list of files or a device to copy **from**.
destination: A file, directory, or device to copy **to**.

/A(SCII)	**/P**(rompt)
/B(inary)	**/Q**(uiet)
/C(hanged)	**/R**(eplace)
/H(idden)	**/S**(ubdirectories)
/M(odified)	**/U**(pdate)
/N(othing)	**/V**(erify)

See also: ATTRIB, MOVE, and REN.

Usage: The 4DOS COPY command accepts all traditional syntax and options and adds several new features.

The simplest use of COPY is to make a copy of a file, like this example which makes a copy of a file called *file1.abc*:

```
c:\> copy file1.abc file2.def
```

You can also copy a file to another drive and/or directory. The following command copies *file1* to the *\MYDIR* directory on drive E:

```
c:\> copy file1 e:\mydir
```

You can copy several files at once by using wildcards:

```
c:\> copy *.txt e:\mydir
```

(See page 71 for an explanation of how 4DOS interprets the wildcard characters [*] and [?].)

4DOS also lets you copy several files at one time. The following command copies 3 files from the current directory to the *\MYDIR* directory on drive E:

```
c:\> copy file1 file2 file3 e:\mydir
```

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

If there is only one argument on the line, COPY assumes it is the *source*, and uses the current drive and directory as the *destination*. For example, the following command copies all the *.DAT* files on drive A to the current directory on drive C:

```
c:\> copy a:*.dat
```

If there are two or more arguments on the line and [+] signs are not used, then COPY assumes that the last argument is the *destination* and copies all *source* files to this new location. If the *destination* is a drive, directory, or device name then the *source* files are copied individually to the new location. If the *destination* is a file name, the first *source* file is copied to the *destination*, and any additional *source* files are then appended to the new *destination* file.

For example, the first of these commands copies the *.DAT* files from the current directory on drive A individually to *C:\MYDIR* (which must already exist as a directory); the second appends all the *.DAT* files together into one large file called *C:\DATA* (assuming *C:\DATA* is not a directory):

```
c:>\ copy a:*.dat c:\mydir\
c:>\ copy a:*.dat c:\data
```

When you copy to a directory, if you add a backslash [\] to the end of the name as shown in the first example above, COPY will display an error message if the name does not refer to an existing directory. You can use this feature to keep COPY from treating a mistyped *destination* directory name as a file name and attempting to append all your *source* files to a *destination* **file,** when you really meant to copy them individually to a *destination* **directory**.

- A plus [+] tells COPY to append two or more files to a single *destination* file. If you list several *source* files separated with [+]

and don't specify a *destination*, COPY will use the name of the first *source* file as the destination, and append each subsequent file to the first file. For example, the following command will append the contents of *MEMO2* and *MEMO3* to *MEMO1* and leave the combined contents in the file named *MEMO1*:

```
c:\> copy memo1+memo2+memo3
```

To append the same three files but store the result in *BIGMEMO:*

```
c:\> copy memo1+memo2+memo3 bigmemo
```

- You cannot append files to a device (such as a printer); if you try to do so, COPY will ignore the [+] signs and copy the files individually. If you attempt to append several *source* files to a *destination* directory or disk, COPY will append the files and place the copy in the new location with the same name as the first *source* file.

- If your *destination* has wildcards in it, COPY will attempt to match them with the *source* names. For example, this command copies the *.DAT* files from drive A to *C:\MYDIR* and gives the new copies the extension *.DX*:

```
c:\> copy a:*.dat c:\mydir\*.dx
```

This feature can give you unexpected results if you use it with multiple *source* file names. For example, suppose that drive A contains *XYZ.DAT* and *XYZ.TXT*. The command

```
c:\> copy a:\*.dat a:\*.txt c:\mydir\*.dx
```

will copy *A:XYZ.DAT* to *C:\MYDIR\XYZ.DX*. Then it will copy *A:XYZ.TXT* to *C:\MYDIR\XYZ.DX*, overwriting the first file it copied.

- COPY also understands include lists (see page 74), 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:\
```

❖ COPY does not change a file's attributes. The *destination* file will have the same attributes as the *source* file.

Options: The **/A**(SCII) and **/B**(inary) options apply to the preceding filename and to all subsequent filenames on the command line until another **/A** or **/B** is entered. The other options (**/C**, **/H**, **/M**, **/N**, **/P**, **/Q**, **/R**, **/S**, **/U**, **/V**) apply to all filenames on the command line, no matter where you put them. For example, either of the following commands could be used to copy a font file to the printer in binary mode:

```
c:\> copy /b myfont.dat prn
c:\> copy myfont.dat /b prn
```

Some options do not make sense in certain contexts, in which case COPY will ignore them. For example, you cannot prompt before replacing an existing file when the *destination* is a device such as the printer -- there's no such thing as an "existing file" on the printer. If you use conflicting output options, like /Q and /P, 4DOS will take a "conservative" approach and give priority to the option which generates more prompts or more information.

Options used in less common situations have been marked with ❖ below. Remember that the options are in alphabetical order, so more basic options are interspersed with those marked with ❖.

❖ **/A**(SCII): If you use **/A** with a *source* filename, 4DOS will copy the file up to, but not including, the first Ctrl-Z (Control-Z or ASCII 26) character in the file. If you use **/A** with a *destination* filename, 4DOS will add a Ctrl-Z to the end of the file (some application programs use the Ctrl-Z to mark the end of a file). 4DOS defaults to **/A** when appending files.

❖ **/B**(inary): If you use **/B** with a *source* filename, 4DOS will copy the entire file. Using **/B** with a *destination* filename prevents 4DOS from adding a Ctrl-Z to the end of the *destination* file. 4DOS defaults to **/B** for normal file copies.

/C(hanged files): Copy files only if the *destination* file exists and is older than the *source* (see also **/U**). This option is useful for updating the files in one directory from those in another without copying any newly created files.

- **/H**(idden): Copy all matching files including those with the hidden and/or system attribute set (see ATTRIB).

  /M(odified): Copy only those files with the archive bit set (see ATTRIB), *i.e.* those which have been modified since the last backup. The archive bit will NOT be cleared after copying.

- **/N**(othing): Do everything except actually perform the copy. This option is most useful for testing what the result of a complex COPY command will be.

  /P(rompt): Ask the user to confirm each *source* file by pressing **Y** or **N**. An **N** response will skip that particular file and continue with the rest of the command.

  /Q(uiet): Turn off the display of the files copied. This option is most often used in batch files.

  /R(eplace): Prompt the user before overwriting an existing file.

  /S(ubdirectories): 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.

  /U(pdate): 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**(erify): 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.

CTTY

(Compatible)

Purpose: Change the default console device.

Format: **CTTY** ***device***

device: The new console device.

❖ Usage: Normally, 4DOS uses the keyboard as the standard input device and the display as the standard output device. Together, the keyboard and display are known as the console or CON. The CTTY command allows you to substitute another device that can perform standard character I/O for the console.

For example to change the console to the first serial port:

```
c:\> ctty com1
```

Change the console back to the standard keyboard and display:

```
c:\> ctty con
```

CTTY works only for programs and commands that use standard DOS input and output functions. This includes all 4DOS internal commands except DRAWBOX, DRAWHLINE, DRAWVLINE, LIST, SCREEN, SCRPUT, SELECT, and VSCRPUT.

DATE

(Compatible)

Purpose: Display and optionally change the system date.

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

mm: The month (01 - 12)

dd: The day (01 - 31)

yy: The year (80 - 199 = 1980 to 2099)

See also: TIME.

Usage: If you simply type **DATE** without any parameters, 4DOS will display the current system date and time, and prompt you 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 of each file as it is created or altered:

```
c:\> date
Mon  Sep 16, 1991  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 9/16/91
```

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

The format for the date entry depends on the country code defined in the *CONFIG.SYS* file or by the CHCP command. The default format is U.S. (*mm-dd-yy*). The European format is *dd-mm-yy*; the Japanese is *yy-mm-dd*.

DEL / ERASE

(Enhanced)

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

Format: **DEL [/N /P /Q /S /X /Y /Z]** ***file...***
or
ERASE [/N /P /Q /S /X /Y /Z] ***file...***

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

/N(othing)
/P(rompt)
/Q(uiet)
/S(ubdirectories)
/X (remove empty subdirectories)
/Y(es to all prompts)
/Z(ap hidden and read-only files)

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
```

Like all 4DOS file processing commands, DEL accepts multiple file names, wildcards (see page 71), and include lists (see page 74). For example, to erase all the files in the current directory with a *.BAK* or *.PRN* extension:

```
c:\> del *.bak *.prn
```

If you enter a subdirectory name, or a filename composed only of wildcards (* and/or ?), DEL asks for confirmation (**Y** or **N**) unless you specified the **/Y** option. If you respond with a **Y**, DEL will delete all the files in that subdirectory (except hidden, system, and read-only files, unless you have used the **/Z** option).

Options:
- **/N**(othing): Do everything except actually delete the file(s). This is useful for testing what the result of a DEL would be.

/P(rompt): Ask the user to confirm each erasure by pressing **Y** or **N**. An **N** response will skip that particular erasure.

/Q(uiet): Don't display filenames as they are deleted. DEL will run fastest if you specify the **/Q** option and the filename doesn't use the extended 4DOS wildcards.

/S(ubdirectories): 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!

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

! ❖ **/Y**(es): The reverse of **/P** -- it assumes a **Y** response to everything, including deleting an entire subdirectory tree. 4DOS 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**(ap): 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 extreme caution! Because EXCEPT works by hiding files, **/Z** will override an EXCEPT command.

For example, to delete the entire subdirectory tree starting with C:\UTIL, including hidden and read-only files, without prompting (use this command with CAUTION!):

```
c:\> del /s/x/y/z c:\util\
```

DELAY

(New)

Purpose: Pause for a specified length of time.

Format: **DELAY [*seconds*]**

seconds: the number of seconds to delay.

Usage: DELAY is useful in batch file loops while waiting for something to occur. To wait for 10 seconds:

```
delay 10
```

A simple loop could make a tone with the BEEP command to get the operator's attention and then DELAY for 60 seconds while it waits for the user to respond.

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

You can cancel a delay by pressing Ctrl-C or Ctrl-Break.

DESCRIBE

(New)

Purpose: Create, modify, or delete file and subdirectory descriptions.

Format: **DESCRIBE *file* [*"description"*] ...**

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

Usage: DESCRIBE adds descriptions to DOS files and subdirectories. The descriptions are displayed by DIR in single-column mode and by SELECT. Each description can be up to 40 characters long. Descriptions let you identify your files in much more meaningful ways than DOS allows in an eight-character filename.

You enter a description on the command line by typing the DESCRIBE command, the filename, and the description in quotation marks, like this:

```
c:\> describe memo.txt "Memo to Bob about party"
```

If you don't put a description on the command line, DESCRIBE will prompt you for it:

```
c:\> describe memo.txt
Describe "memo.txt" : Memo to Bob about party
```

If you use wildcards or multiple filenames with the DESCRIBE command and don't include the description itself, 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.

4DOS stores the descriptions in each directory in a hidden file called *DESCRIPT.ION*. Use the ATTRIB command to "unhide" this file if you need to copy or delete it.

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.

DIR

(Enhanced)

Purpose: Display information about files and subdirectories.

Format: **DIR [/1 /2 /4 /A[[:][-]rhsda] /B /C /F /J /K /L /M /N /O[[:][-]deginsu] /P /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(ase -- use upper case)
/F(ull path)
/J(ustify names)
/K (suppress header display)
/L(ower case)
/M (suppress footer display)
/N (reset DIR options)
/O (sort order)
/P(ause)
/S(ubdirectories)
/T (aTtribute display)
/U (sUmmary information)
/V(ertical sort)
/W(ide)

See also: ATTRIB, DESCRIBE, SELECT, and SETDOS.

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; and the file description. You can also display information in 1, 2, 4, or 5 columns, sort the files several different ways, use color to distinguish file types, and pause after each full screen.

The various DIR displays are controlled through options or switches. The best way to learn how to use the many options available with the DIR command is to experiment. You will soon know which options you want to use regularly. You can select those options permanently by using the ALIAS command. You may want to mix several options. For example, to display all the files in the current directory, in 2 columns, sorted vertically (down one column then down the next), and with a pause at the end of each page:

```
c:\> dir /2/p/v
```

To set up this format as the default, using an alias:

```
c:\> alias dir=`*dir /2/p/v`
```

This example displays all the files on all directories of drive *C*, including hidden and system files, pausing after each page:

```
c:\> dir /s/h/p c:\
```

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

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

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 link multiple filenames with a semi-colon **[;]** (an "include list", see page 74), 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
```

If you have an ANSI driver loaded, you can display the file and subdirectory names in color by setting the COLORDIR environment variable. The format for COLORDIR is:

```
ext ... :[BRIght][BLInk] fg [ON bg]; ...
```

where "ext" is a file extension (1 to 3 characters) 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
```

For example, to display the .COM and .EXE files in red, the .C and .ASM files in bright cyan, and the read-only files in blinking green (this should be entered on one line):

```
c:\> set colordir=com exe:red; c asm:bright cyan;
     rdonly:blink green
```

If you don't specify a background color, DIR will use the current screen background color. COLORDIR will **not** work properly unless you have an ANSI driver loaded.

If you have COLORDIR set and attempt to redirect the output of DIR to a character device, such as a serial port or the printer, non-colorized file names will be displayed on the device but colorized names will still be displayed on the screen. This will not occur if the output of DIR is redirected to a disk file. You can send the output of DIR to a character device by redirecting it to a file, then copying the file to the device, for example:

```
c:\> dir > $dtemp ^ copy $dtemp prn ^ del $dtemp
```

- If a country code was defined in the *CONFIG.SYS* file or by the CHCP command, DIR will display the date in the format for that country. The default date format is U.S. (mm-dd-yy). The separator character in the file time will also be affected by the country code.

- DIR can handle directories of any size, limited only by available memory. Each filename requires 32 bytes of free base memory plus the size of the description (if any); a system with 128K of free base memory can display up to 4,000 files per directory.

- Options on the command line apply only to the filenames which follow the option, and options at the end of the line apply to the preceding filename only. This allows you to specify different options for different groups of files, yet retains compatibility with the traditional DIR command when a single filename is specified.

Options: **/1**: Single column display -- display the filename, size, date, time, and description. This is the default.

/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).

- **/A**(ttributes): Display only those files that have the specified attribute(s) set. Preceding the attribute character with a hyphen [-] will display those files that DON'T have that attribute set. The attributes are:

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

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. See page 173 for more information on file attributes.

- **/B**(are): 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.

/C(ase): Display filenames in the traditional upper case; also see SETDOS /U (page 280) and the UpperCase directive in *4DOS.INI* (page 127).

- **/F**(ull path): Display each filename with its drive letter and path in a single column, without other information.

/J(ustify): Justify (align) filename extensions and display them in the traditional format.

- **/K**: Suppress the header (disk and directory name) display.

/L(ower case): Display filenames in lower case; also see SETDOS /U (page 280) and the UpperCase directive in *4DOS.INI* (page 127).

- **/M**: Suppress the footer (file and byte count total) display.

/N: 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(rder): 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
d	Sort by date and time (oldest first)
e	Sort by extension
g	Group subdirectories together
i	Sort by the file description
n	Sort by filename (this is the default)
s	Sort by size
u	Unsorted

/P(ause): Wait for a key to be pressed after each screen page before continuing the display.

/S(ubdirectories): Display file information from the current directory and all of its subdirectories. DIR will only display headers and summaries for those directories with files that match the filename(s) and attributes (if **/A** is used) that you specify on the command line.

❖ **/T** (aTtributes): Display the filenames and attributes only. The attributes are displayed in the format RHSA, where:

R	Read-only
H	Hidden
S	System
A	Archive

/U (sUmmary information): Only display the number of files, the total file size, and the total amount of disk space used.

/V(ertical sort): Display the filenames sorted vertically rather than horizontally (used with the **/2**, **/4** or **/W** options).

/W(ide): Display filenames only, horizontally across the screen (5 columns on an 80-character wide display).

DIRS

(New)

Purpose: Display the current directory stack.

Format: **DIRS**

See also: PUSHD and POPD.

Usage: The PUSHD command adds the current default drive and directory to the directory stack, a list that 4DOS 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 255 characters, enough for about 10 to 20 typical drive and directory entires.

DRAWBOX

(New)

Purpose: Draw a box on the screen.

Format: **DRAWBOX *ulrow ulcol lrrow lrcol style* [BRIght] [BLInk] fg ON bg [FILl *bgfill*] [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)

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

See also: DRAWHLINE and DRAWVLINE.

Usage: DRAWBOX is useful for creating attractive screen displays in batch files. DRAWBOX 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 box around the entire screen with bright white lines on a blue background (enter this on one line):

```
drawbox 0 0 24 79 1 bri whi on blu fill blu
```

Only the first three characters of the color name and the keywords BRIGHT, BLINK, FILL, and SHADOW are required.

If you use SHADOW, a drop shadow is created by changing the characters in the row under the box and the 2 columns to the right of the box to normal intensity text with a black background (this will make characters displayed in black disappear entirely).

The row and column values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79.

DRAWBOX checks for valid row and column values, and displays a "Usage" error message if any values are out of range.

DRAWHLINE

(New)

Purpose: Draw a horizontal line on the screen.

Format: **DRAWHLINE *row column len style* [BRIght] [BLInk] fg ON 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

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

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
```

Only the first three characters of the color name and the attributes BRIGHT and BLINK are required.

The row and column values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79.

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

DRAWVLINE

(New)

Purpose: Draw a vertical line on the screen.

Format: **DRAWVLINE *row column len style* [BRIght][BLInk]**
fg* ON *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

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

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
```

Only the first three characters of the color name and the attributes BRIGHT and BLINK are required.

The row and column values are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79.

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

ECHO

(Enhanced)

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

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

message: Text to display.

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

Usage: 4DOS has a separate echo capability for batch files and for the command line.

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 4DOS redirection symbols (| > <) in an ECHO message, unless you enclose them in quotes or precede them with the escape character (see page 91).

4DOS defaults to ECHO ON in batch files. The current ECHO state is inherited by called batch files. You can change the default setting to ECHO OFF with the SETDOS /V0 command or the BatchEcho directive in *4DOS.INI* (see page 125).

If you turn the command line ECHO on, 4DOS will display each command before it is executed. This will let you see the command line after 4DOS has expanded all aliases and variables. The command line ECHO is most useful when you are learning how to use the advanced features of 4DOS. This example will turn command line echoing on:

```
c:\> echo on
```

4DOS defaults to ECHO OFF during keyboard input. The keyboard 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.

ECHOS

(New)

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

Format: **ECHOS message**

See also: ECHO, SCREEN, SCRPUT, TEXT, and VSCRPUT.

Usage: ECHOS is useful for outputting text when you don't want 4DOS to add a carriage return / linefeed pair. For example, you can use ECHOS when you need to redirect control sequences to your printer:

```
c:\> echos ↑eP > lpt1:
```

You cannot use the command separator character [^] or the 4DOS redirection symbols (|><) in an ECHOS message, unless you enclose them in quotes or preceded them with the escape character (see page 91).

ENDLOCAL (New)

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

Format: **ENDLOCAL**

See also: SETLOCAL.

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

For example, this batch file fragment saves the drive, current working directory, and environment variables, changes the drive and directory, sets some environment variables, runs the program *TEST1*, and then restores the original values:

```
setlocal
cdd d:\test
set path=c:\;c:\dos;c:\util
set lib=d:\lib
test1
endlocal
```

SETLOCAL and ENDLOCAL can only be used in batch files, not in aliases or from the command line.

ESET (New)

Purpose: Edit environment variables and aliases.

Format: **ESET [/M]** ***variable name...***

variable name: The name of an environment variable or alias to edit.

/M(aster environment)

See also: ALIAS, UNALIAS, SET, and UNSET.

Usage: ESET allows you to edit environment variables and aliases using the 4DOS line editing commands (see page 56).

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 only be able to edit the environment variable.

4DOS limits environment variable and alias names to 80 characters, and their arguments to 255 characters.

Option:
- **/M**(aster environment): Edit an environment variable in the master environment rather than the local environment. This option is only useful from a secondary command shell (for example, when an application has "shelled to DOS").

EXCEPT

(New)

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.

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 a 4DOS 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 ending in *.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
```

- If you use EXCEPT with filename completion (see page 60) to get 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.

- 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 "unhide" (-H) the file(s).

- EXCEPT will not work with programs or commands that ignore the hidden attribute or which work explicitly with hidden files, including DEL /Z and the /H (process hidden files) switch available in some 4DOS file processing commands.

- You can use command grouping (see page 90) 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*. This example should be entered on one line:

```
c:\data> except (*.dat) (copy *.da* d:\save ^
         ren *.da* *.sa*)
```

- You may need to increase 4DOS's internal stack size using the StackSize directive in *4DOS.INI* if you use extremely complex combinations of commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line, or use complex combinations of these commands in nested batch files or nested GOSUBs. See the StackSize directive on page 132 for more information.

EXIT

(Enhanced)

Purpose: Return from a secondary command processor.

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

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

Usage: Some application programs will start a secondary copy of the command processor to allow you to execute DOS commands. To return to the application again, type:

```
c:\> exit
```

- If you specify a value, EXIT will return that value to the program that started 4DOS. For example:

```
c:\> exit 255
```

- The value is a number you can use to inform the program of some result, such as the success or failure of a batch file. This feature is most useful for systems which use batch files to automate their operation, such as bulletin boards, or custom application programs like databases that shell to 4DOS to perform certain tasks.

- You cannot EXIT from the primary command processor unless you are running inside an OS/2 2.0 DOS compatibility box.

FOR

(Enhanced)

Purpose: Repeat a command for several values of a variable.

Format: **FOR *%var* IN ([@]*set*) [DO] *command* ...**

%var: The variable to be used in the command ("FOR variable").
set: A set of values for the variable.
command: A command or group of commands to be executed for each value of the variable.

Usage: 4DOS begins the FOR command by creating a set. It then executes a command for every member of the set. The command can be a 4DOS internal command or alias, an external command, or a batch file.

Normally, the set is a list of files specified with wildcards. For example, if you use this line in a batch file:

```
for %x in (*.txt) do list %x
```

4DOS will create a list of all files in the current directory with the extension *.TXT*. It sets the FOR variable %x equal to each of the file names in turn, and executes the LIST command for each of the files.

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

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

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

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

❖ You can also set the FOR variable equal to each line in a file by placing an [@] in front of the file name. If you have a file called *DRIVES.TXT* that contains a list of drives on your computer, one drive name per line (with a ":" after each drive letter), you can print the free space on each drive this way:

```
for %d in (@drives.txt) do free %d > prn
```

- Because the [@] is also a valid filename character, FOR first checks to see if the file exists with the [@] in its name (*e.g.*, *@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.

- 4DOS will accept either % or %% in front of the variable name. You can use either form whether the FOR command is typed from the command line or is part of an alias or 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 4DOS will give that name priority over any environment variable which starts with the same letter, in order to maintain compatibility with the traditional FOR command. For example, the following command tries to add a: and b: to the end of the PATH, but will not work as intended:

```
c:\> for %p in (a: b:) do path %path;%p
```

  The "%p" in "%path" will be interpreted as the FOR variable %p followed by the text "ath", which is not what was intended. To get around this, use a different letter or a longer name for the FOR variable, or use square brackets around the variable name (see page 78).

- 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):

```
c:\docs> for %file in (*.txt) do if exist
         %@name[%file].txt del %@name[%file].bak
```

- You can use command grouping (see page 90) to execute multiple commands for each element in the list. For example, the following command copies each *.WKQ* file in the current directory to the *D:\WKSAVE* directory, and then changes the extension of each file in the current directory to *.SAV*. This example should be entered on one line:

```
c:\text> for %file in (*.wkq) do (copy %file
         d:\wksave\ ^ ren %file *.sav)
```

- 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 (this should all be entered on one line):

```
c:\> for %x in (filetest fileform fileprnt)
     do %x datafile
```

  4DOS will expand this to three separate commands:

```
filetest datafile
fileform datafile
fileprnt datafile
```

- The variable that FOR uses (the **%X** in the example above) is created in the environment and then erased when the FOR command is done. Because of this, you must be careful not to use the name of one of your environment variables as a FOR variable. For example, a command that begins

```
c:\> for %path in (...
```

  will write over your current path setting and then erase the path variable completely.

- FOR statements can be nested. The permissible nesting level depends on the amount of free space in 4DOS's internal stack.

- You may need to increase 4DOS's internal stack size using the StackSize directive in *4DOS.INI* if you use extremely complex combinations of commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line, or use complex combinations of these commands in nested batch files or nested GOSUBs. See the StackSize directive on page 132 for more information.

FREE

(New)

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 DOS command CHKDSK, but without the wait, since it does not check the integrity of the file and directory structure of the disk.

A colon [:] is required after each drive letter. This example displays the status of drives *A* and *C*:

```
c:\> free a: c:
 Volume in drive A: is unlabeled
  1,213,952 bytes total disk space
  1,115,136 bytes used
     98,816 bytes free
 Volume in drive C: is DEVELOPMENT
 42,496,000 bytes total disk space
 36,851,712 bytes used
  5,644,288 bytes free
```

If you are using DOS 4.0 or later, the disk serial number will appear after the drive label or name.

Some networks with large server disk drives (256 MB or more) may report disk space values that are too small when FREE is used. If this occurs, it is because the network software does not provide a way to return larger values to 4DOS.

GLOBAL

(New)

Purpose: Execute a command in the current directory and its subdirectories.

Format: **GLOBAL [/H /I /Q]** ***command***

command: The command to execute, including arguments and switches.

/H(idden directories) **/Q**(uiet)
/I(gnore exit codes)

Usage: GLOBAL performs the command first in the current directory and then in every subdirectory under the current directory. The command can be a 4DOS internal command or alias, an external command, or a batch file.

The first example erases all the files with a *.BAK* extension in every directory on *C*. The second example copies the files in every directory on drive A to the directory *C:\TEMP*:

```
c:\> global erase *.bak
a:\> global copy *.* c:\temp
```

- ❖ You can use command grouping (see page 90) 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)
```

- ❖ You may need to increase 4DOS's internal stack size using the StackSize directive in *4DOS.INI* if you use extremely complex combinations of commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line, or use complex combinations of these commands in nested batch files or nested GOSUBs. See the StackSize directive on page 132 for more information.

Options: ❖ **/H**(idden directories): Forces GLOBAL to look for hidden directories. If you don't use this switch, hidden directories are ignored.

- **/I**(gnore exit codes): If this option is not specified, GLOBAL will terminate if the command returns a non-zero exit code. Use **/I** if you want *command* to continue in additional subdirectories even if it returns an error in a previous subdirectory.

  /Q(uiet): Do not display the directory names as each directory is processed.

GOSUB

(New)

Purpose: Execute a subroutine in the current batch file.

Format: **GOSUB *label***

label: The batch file line label at the beginning of the subroutine.

See also: CALL, GOTO and RETURN.

❖ Usage: 4DOS allows subroutines in batch files. A subroutine begins with a label (a colon followed by a word) and ends with a RETURN command. The subroutine is invoked with a GOSUB command from another part of the batch file. The RETURN command ends a subroutine; execution of the batch file will continue with the command following the original GOSUB. GOSUB allows you to create subroutines within a batch file (to call other batch files, see CALL.)

The subroutine must start with a *label* that begins with a colon [:] and which appears on a line by itself. 4DOS ignores case differences when matching labels.

The subroutine must end with a RETURN statement. After the RETURN, 4DOS will continue processing the batch file with the command following the GOSUB command.

The following batch file fragment calls a subroutine which displays the directory and returns:

```
echo Calling a subroutine
gosub subr1
echo Returned from the subroutine
quit
:subr1
dir /a/w
return
```

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

GOSUB saves the IFF state, so IFF statements inside a subroutine won't interfere with IFF statements in the part of the batch file from which the subroutine was called.

Subroutines can be nested. The permissible nesting level depends on the amount of free space in 4DOS's internal stack.

- You may need to increase 4DOS's internal stack size using the StackSize directive in *4DOS.INI* if you use extremely complex combinations of commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line, or use complex combinations of these commands in nested batch files or nested GOSUBs. See the StackSize directive on page 132 for more information.

GOTO

(Compatible)

Purpose: Branch to a specified line inside the current batch file.

Format: **GOTO [/I] *label***

label: The batch file line label to branch to.

/I(FF continues)

See also: GOSUB.

Usage: 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. 4DOS ignores case differences when matching labels.

This batch file fragment checks for the existence of the file *CONFIG.SYS*. If the file exists, 4DOS jumps to C_EXISTS and copies all the files from the current directory to the root directory on A:. Otherwise, 4DOS prints an error message and exits.

```
if exist config.sys goto C_EXISTS
echo CONFIG.SYS doesn't exist - exiting.
quit
:C_EXISTS
copy *.* a:\
```

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

To avoid errors in the processing of nested IFF statements, if **/I** is not used GOTO cancels all active IFF statements. This means that a normal GOTO (without **/I**) inside an IFF statement must branch outside all IFF statements, and may not branch to any label that is between an IFF and the corresponding ENDIFF. This includes branches inside the current IFF statement.

Options: ❖ **/I**(FF continues): Prevents GOTO from cancelling IFF statements. Use this option only if you are absolutely certain that your GOTO command is branching entirely within the current active IFF statement, and not into another IFF statement or a different IFF nesting level. Using **/I** under any other conditions will cause an error later in your batch file.

HELP

(New, External command)

Purpose: Display help for 4DOS and DOS commands.

Format: **HELP [/M] [*topic*]**

topic: A help topic, 4DOS internal command, or DOS external command.

/M(onochrome)

Usage: HELP displays a brief description and the proper syntax of both 4DOS and DOS commands. If you simply type

```
c:\> help
```

you will see a list of all help topics. You can select a topic by using the cursor keys and then pressing Enter.

If you want to avoid the opening list, use a topic or command name. For example, if you want help with the 4DOS COPY command, type

```
c:\> help copy
```

You can also start HELP by pressing F1 at the 4DOS prompt (see page 63).

4HELP.EXE and *DOS.HLP* must be in the current directory or one of the directories specified in the current PATH setting. If you keep the help files in a directory which is not on your PATH, you must set the full path for the help program with the HelpPath directive in *4DOS.INI* (see page 122). If you use the HelpPath directive, the HELP command will generally respond more quickly, because 4DOS won't have to search the directories in your PATH setting to find the help files.

The HELPCFG program included with 4DOS allows you to customize the HELP colors. To use it, just change to your 4DOS directory, run HELPCFG, and follow the instructions it displays.

If you use another program named HELP, you can use two alias commands to rename the 4DOS help command as 4HELP:

```
c:\> alias 4help=`*help`
c:\> alias help=`c:\util\help.exe`
```

If you want to customize the help text (for example, to add help for your own commands or aliases), you will need the 4DOS Utility Disk which contains the source text and hypertext compiler.

Options: /M(onochrome): forces HELP to use a monochrome display mode on color systems. This is useful on any system where HELP may be "fooled" into thinking you have a color display when you don't, including portable computers with LCD screens.

HISTORY

(New)

Purpose: Display, add to, clear, or read the history list.

Format: **HISTORY [/A *command*] [/F] [/P] [/R *filename*]**

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

See also: LOG

Usage: 4DOS keeps a list of the commands you have entered on the command line. See page 57 for information on command recall, which allows you to use the history list to repeat or edit commands you have typed.

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 add new commands to the list without executing them, save the list in a file, or read a new list from a file.

The number of commands saved in the history list depends on the length of each command line. The history list size can be specified at startup from 256 to 8192 characters (see page 122). The default size is 1024 characters.

- You can use the HISTORY command as an aid in writing batch files. Any time you have executed a series of commands that you'd like to save as the basis for a batch file, simply redirect the output of HISTORY to a file:

```
c:\> history > newbatch
```

Then edit the output file to contain only the commands you want in the batch file, and save it under the appropriate name. If you know you're going to use HISTORY this way, you may want to use the **/F** switch to clear the history first, so that your output file isn't cluttered with too many extraneous commands.

- You can disable the history list or specify a minimum command line length to save with the HistMin directive in the *4DOS.INI* file.

Options:

- **/A**(dd): Add a command to the history list. This performs the same function as the Ctrl-K key at the command line (see page 57).

  /F(ree): Erase all entries in the command history list.

  /P(rompt): Wait for a key after displaying each page of the list.

- **/R**(ead): Read the command history from the specified file and append it to the history list currently held in memory. 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 *HISTFLE* and reads it back again immediately. If you leave out the second line, 4DOS will append the contents of the file to the current history list instead of replacing the current history list with the file copy:

```
c:\> history > histfile
c:\> history /f
c:\> history /r histfile
```

  If you need to save your history at the end of each day's work, you might use commands like this in your *AUTOEXEC.BAT* file:

```
if exist c:\histfile history /r c:\histfile
alias shut*down `history > c:\histfile`
```

  This restores the previous history list if it exists, then defines an alias which will save the history before shutting off the system.

IF

(Enhanced)

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.

Usage: IF is normally used only in aliases and batch files. It is always followed by one or more *conditions* and then a *command*. 4DOS first evaluates the *conditions*. If they are true, 4DOS executes the *command*. Otherwise, the *command* is ignored. If you add a NOT before a *condition*, the *command* is executed only when the *condition* is false.

You can link *conditions* with **.AND.**, **.OR.**, or **.XOR.**, and you can nest IF statements. The *conditions* can test strings, numbers, the existence of a file or subdirectory, the errorlevel returned from the preceding external command, and the existence of alias names and internal commands.

The *command* can be an alias, a 4DOS internal command, an external command, or a batch file. The entire IF statement, including all *conditions* and the *command*, must fit on one line.

- You can use command grouping (see page 90) 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)
```

(The IFF command provides a more structured method of executing multiple commands if a condition or set of conditions is true.)

- You may need to increase 4DOS's internal stack size using the StackSize directive in *4DOS.INI* if you use extremely complex

combinations of commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line, or use complex combinations of these commands in nested batch files or nested GOSUBs. See the StackSize directive on page 132 for more information.

Conditions: The following conditional tests are available in both the IF and IFF commands. They fit into three categories: string tests, numeric tests, and system tests. The tests can use environment variables, 4DOS internal-variables and variable functions (see pages 80 and 84), file names, and literal text and values as their arguments.

If you use one of the string or numeric tests like == or **GE**, 4DOS decides whether to compare the values as numbers or as strings by examining the first character of each value. If both values begin with a digit, 4DOS performs a numeric comparison. If either value does not begin with a digit, a string comparison is done. To force a string comparison, use double quotes around the values you are testing.

The difference between numeric and string comparisons is best explained by looking at the way values with and without blanks are tested. For example, consider comparing the values 2 and 19. Numerically, 2 is smaller, but as a string it is "larger" because its first digit is larger than the first digit of 19. So the first of these *condition*s will be true, and the second will be false:

```
if 2 lt 19 ...
if "2" lt "19" ...
```

String Tests:

For the string tests, 4DOS ignores case differences. When you compare 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.

string1 == string2
string1 **EQ** string2

If *string1* is **equal to** *string2*, then the condition is true.

string1 != string2
string1 **NE** string2

If *string1* is **not equal to** *string 2*, then the condition is true.

string1 **LT** string2

If *string1* is **less than** *string2*, then the condition is true.

string1 **LE** string2

If *string1* is **less than or equal to** *string2*, then the condition is true.

string1 **GT** string2

If *string1* is **greater than** *string2*, then the condition is true.

string1 **GE** string2

If *string1* is **greater than or equal to** *string2*, then the condition is true.

This first batch file fragment runs a program called *MONOPROG* if a monochrome monitor is attached to the system:

```
if "%_monitor" == "mono" monoprog
```

The second 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 of the file *MYFILE* contain the string "GO" (enter this example on one line):

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

Numeric Tests:

The numeric tests available are exactly the same as the string tests (**==, EQ, !=, NE, LT, LE, GT, and GE**), but compare values as numbers instead of strings. Both values must begin with a digit or 4DOS will perform a string test instead.

The first example below tests whether there is more than 500 KBytes of free base memory; the second tests for more than 2 MBytes of free EMS memory:

```
c:\> if %@dosmem[k] gt 500 echo Over 500K free
c:\> if %@ems[m] gt 2 echo Over 2 MB EMS free
```

This example shows how to implement a simple loop inside a batch file. The lines between the labels **:loop** and **:loopdone** will be executed 100 times:

```
set limit=100
set counter=1
:loop
if %counter gt %limit goto loopdone
  rem DO SOME WORK HERE
  set counter=%@eval[%counter + 1]
  goto loop
:loopdone
```

System Tests:

The final conditions test the system status. You can also use 4DOS pseudo-variables and variable functions to test other parts of the system status.

ERRORLEVEL [condition] 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 numeric operators listed above (EQ, !=, GT, etc.). If no relational operator is specified, the default is GE. The comparison between the exit code and *n* is done numerically.

Not all programs return an explicit exit code. For programs which do not, the behavior of ERRORLEVEL is undefined and may be erratic.

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 wildcards exists.

ISALIAS aliasname

If the specified name is an alias, the condition is true.

ISDIR path

If the subdirectory exists, the condition is true.

ISINTERNAL command

If the specified command is an active, internal 4DOS command, the condition is true. Commands can be activated and deactivated with the SETDOS /I command.

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

```
if exist a:\jan.doc copy a:\jan.doc c:\
```

This example tests the exit code of the previous program and stops 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 sort with **.AND.**, **.OR.**, and **.XOR.** Test conditions are always scanned from left to right -- there is no implied order of precedence, as there is in some programming languages.

When two tests are combined with **.AND.**, the result is true if both of the individual tests are true. When two tests are

combined with **.OR.**, the result is true if either (or both) of the 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 *HIGHRES* if either an EGA or VGA video adapter is in use:

```
if "%_video"=="EGA" .or. "%_video"==vga highres
```

IFF

(New)

Purpose: Allow 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 carets or else be placed on separate lines of a batch file.

See also: IF

Usage: IFF is similar to the IF command, except that it can perform one set of *commands* when a condition or set of *conditions* is true and different *commands* when the *conditions* are false.

IFF can execute multiple commands when the *conditions* are true or false; IF normally executes only one command. IFF imposes no limit on the number of commands and is generally a "cleaner" and more structured command than IF.

IFF is always followed by one or more *conditions*, which 4DOS evaluates. If they are true, 4DOS executes the *commands* that follow the word THEN. Additional *conditions* can be tested with ELSEIFF. If none of these *conditions* are true, 4DOS executes the *commands* that follow the word ELSE. In both cases, after the selected *commands* are executed, processing continues after the word ENDIFF.

If you add a NOT before the condition, the THEN *commands* are executed only when the *condition* is false and the ELSE *commands* are executed only when the *condition* is true.

You can link *conditions* with **.AND.**, **.OR.**, or **.XOR.**, and you can nest IFF statements up to 15 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 4DOS internal command or alias, external command, or batch file.

The following batch file fragment tests the monitor type (monochrome or color), and sets the appropriate colors and prompt (enter the "prompt" lines on one line of the batch file):

```
iff "%_monitor" == "color" then
  color bright white on blue ^ cls
  prompt=$e[s$e[1;1f$e[41;1;37m$e[K  Path:
     $p$e[u$e[44;37m$n$g
else
  prompt=$e[s$e[1;1f$e[0;7m$e[K  Path:
     $p$e[u$e[0m$n$g
endiff
```

The alias in this second 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 ^ rd %1 ^ else ^
     echo Not a directory!^endiff`
```

❖ If you do a GOTO inside an IFF, 4DOS normally assumes you are jumping outside of all active IFF statements. If you attempt to GOTO another part of the same IFF, the middle of another IFF, or a different IFF nesting level, you will eventually receive an "unknown command" error on a subsequent ELSE, ELSEIFF, or ENDIFF statement. You can override this restriction with GOTO /I, but do so only if you are absolutely certain that your GOTO command is branching entirely within the current active IFF statement. Using /I under any other conditions will cause an error later in your batch file.

INKEY

(New)

Purpose: Get a single keystroke from the user and store it in an environment variable.

Format: **INKEY [/Wn] [*prompt*] *%%varname***

prompt: Optional text that is displayed as a prompt.
varname: The variable that will hold the user's keystroke.

/W(ait)

See also: INPUT and KEYSTACK.

Usage: INKEY optionally displays a prompt for user input. 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 from the user. Along with the INPUT command, INKEY allows great flexibility in reading user 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.

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). See Appendix B (page 311) for a list of the ASCII and extended key codes.

If you press Ctrl-C or Ctrl-BREAK while INKEY is waiting for a key, execution of an alias will be terminated, and execution of a

batch file will be suspended while you are asked whether to cancel the batch job.

Option: **/W**(ait): Timeout period, in seconds, to wait for a response. If no keystroke is entered by the end of the timeout period, INKEY returns with the variable unchanged. You can specify **/W0** to return immediately if there are no keys waiting in the keyboard buffer.

For example, the following batch file fragment waits up to 10 seconds for a character, then tests to see if a "Y" was entered:

```
set net=N
inkey /w10 Load the network (Y/N)?  %%net
iff "%net" == "Y" then
  rem Commands to load the network go here
endiff
```

INPUT

(New)

Purpose: Get a string from the keyboard and save it in an environment variable.

Format: **INPUT [/Wn] [*prompt*] %%*varname***

prompt: Optional text that is displayed as a prompt.
varname: The variable that will hold the user's input.

/W(ait)

See also: INKEY and KEYSTACK.

Usage: INPUT optionally displays a prompt for user input. Then it waits for a specified time or indefinitely for the user's entry. It places any characters typed by the user into an environment variable. INPUT is normally used in batch files and aliases to get multi-key input from the user. 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 4DOS 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.

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.

Option: **/W**(ait): Timeout period, in seconds, to wait for a response. If no keystroke is entered by the end of the timeout period, INPUT returns with the variable unchanged. If you enter a key before the timeout 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.

KEYSTACK

(New)

Purpose: Feed keystrokes to a program or command automatically.

Format: **KEYSTACK [/Wn] [*"abc"*] [*keyname*] [*!*] ...**

"abc": Literal characters to be placed in the Keystack.
keyname: Name or code for a key to be placed in the Keystack.
!: Signal to clear the Keystack and the keyboard buffer.

/W(ait)

Usage: KEYSTACK takes a series of keystrokes and feeds them to a program or command as if they were typed at the keyboard. When the program has used all of the keystrokes in the keystack buffer, it will begin to read the keyboard for input, as it normally would.

KEYSTACK will only work if the memory-resident program *KSTACK.COM* has been loaded. KSTACK is usually loaded from the *AUTOEXEC.BAT* file (see page 112). If KSTACK is not loaded the KEYSTACK command will display an error message.

Characters entered within double quotes (*"abc"*) will be stored "as is" in the keyboard buffer. The only items allowed outside double quotes are key names, key codes, and the **/W** option.

Key names are entered in the form:

```
[Prefix[-]]Keyname
```

The key prefix can be left out, in which case any of the key names can be used **except** A - Z or 0 - 9. If the key prefix is used 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, Bksp, Enter, Left, Right, Home, End, PgUp, or PgDn
Shift	followed by F1 - F10 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, and can be abbreviated as long as the abbreviation is unique and unambiguous.

The prefix and key name can be together or separated by a dash, but cannot be separated by any other characters. For example,

```
AltF10          This is okay
Alt-F10         This is also 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. In general, you will find it easier to use the names described above rather than key numbers. See Appendix B (page 311) for an explanation and list of keyboard codes.

An exclamation mark [!] will clear all pending keystrokes, both in the KEYSTACK buffer and in the keyboard buffer.

For example, to start ProComm Plus and skip the opening screen you could use the command:

```
c:\comm> keystack 32 ^ pcplus
```

This places a space (ASCII code 32) in the buffer, then runs ProComm Plus. When ProComm looks for a keystroke to end the display of the opening screen the keystroke is already in the buffer, so the opening screen is removed immediately.

The KEYSTACK command must be executed **before** running the program which is going to receive the stacked keystrokes. This places the keystrokes into the buffer first, so the program can find them when it runs.

You can store a maximum of 255 characters in the KEYSTACK buffer. 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 a keystroke sequence that works for any particular program.

Programs that bypass DOS and the BIOS for keyboard input cannot read keystrokes entered with KEYSTACK. If you use KEYSTACK then run such a program, the keystrokes will not appear in the program, but may appear at the prompt when you exit the program and return to 4DOS.

❖ KEYSTACK treats the number 0 as a special case; it is used with programs that flush the keyboard buffer. When KEYSTACK processes a key value of 0, it tells the program the buffer is clear, so subsequent keystrokes will be accepted normally. Some programs will require several "0"s before they will accept input; you may need to experiment to determine the correct number.

For example, the following batch file starts Lotus 1-2-3 and loads the file specified on the command line when the batch file is invoked (the KEYSTACK command should be entered on one line):

```
pushd c:\123
keystack 0 Enter 0 Enter 0 Enter 0 Enter 0 Enter
  "/FR" 0 "%1" Enter
123
popd
```

The sequence of "0 Enter" pairs tells 1-2-3 that the keyboard buffer is empty, then passes 1-2-3 a carriage return, repeating this sequence five times. This gets 1-2-3 to a point where an empty spreadsheet is displayed. The rest of the KEYSTACK line issues a File Retrieve command (/FR), simulates an empty keyboard buffer once more, enters the file name passed on the batch command line (%1), and finally enters a carriage return to end the file name.

- Here's the same command defined as an alias (enter this on one line):

```
alias 321 `pushd c:\123 ^ keystack 0 Enter 0
Enter 0 Enter 0 Enter 0 Enter "/FR" 0 "%1" 13 ^
123 ^ popd`
```

- Some programs require both the keyboard "scan code" and the ASCII value to be stacked. To stack both codes, calculate the value ((256 * scan code) + ASCII code) and enter that numeric value as an argument for KEYSTACK. For example, for the Enter key, the scan code is 28 and the ASCII code is 13, so to stack both values use ((256 * 28) + 13) or KEYSTACK 7181. Try this approach if a "normal" KEYSTACK command does not work (for example, if you use KEYSTACK 13 for the Enter key and the program doesn't see the correct character). To stack combined key codes you must use the numeric value, not the key name (see Appendix B on page 311 for keyboard codes).

Options:

- **/W**(ait): 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 65535 (65535 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. Sometimes a combination of a delay and an "empty buffer" signal (a 0) are required. For example, to start the program CADX and send it an F7, a delay of one second, an indication that the keyboard buffer is empty, and a carriage return:

```
c:\> keystack F7 /W18 0 Enter ^ cadx
```

LH / LOADHIGH

(Compatible)

Purpose: Load a memory resident program into an Upper Memory Block (UMB).

Format: **LH *filename***
or
LOADHIGH *filename*

filename: The name of the program to load into high memory.

Usage: LH and LOADHIGH are synonyms. You can use either one.

LOADHIGH requires MS-DOS 5.0 or above.

If you load memory-resident programs into UMBs, you will have more room in base memory for application programs. If your system has no UMBs, or if the program is larger than the largest UMB, then LOADHIGH will load the program into conventional base memory.

For example, to load the program *C:\UTIL\CACHE.EXE* into high memory:

```
c:\> loadhigh c:\util\cache.exe
```

In addition to MS-DOS 5.0 or above, LOADHIGH requires the DOS=UMB command in your *CONFIG.SYS* file. It can only be used to load programs into UMBs managed by MS-DOS 5.0 and above.

If you use a memory manager like 386MAX or QEMM to manage your UMBs, rather than the MS-DOS DOS=UMB directive, then LOADHIGH will not work, and you must use the equivalent command supplied with your memory manager in order to load programs high.

LIST

(New)

Purpose: Display a file, with forward and backward paging and scrolling.

Format: **LIST [/H /S /W]** ***file...***

file: A file or list of files to display.

/H(igh bit off)
/S(tandard input)
/W(ord wrap)

See also: TYPE.

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

LIST is normally used for displaying ASCII text files. Most other files contain non-alphabetic characters and may be unreadable.

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

```
c:\> list memo.doc
```

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

Home	Display the first page of the file.
End	Display the last page of the file.
Esc	Exit the current file.
Ctrl-C	Quit LIST.
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- Left Arrow	Scroll left 40 columns.
Ctrl-Right Arrow	Scroll right 40 columns.
F1	Call the 4DOS online help
F	Prompt and search for a string (case is ignored).
H	Toggle the "strip high bit" (**/H**) option.
N	Find next matching string (case is ignored).

P	Print the entire file on LPT1.
W	Toggle the "line wrap" (**/W**) option.

LIST saves the search string used by **F** and **N**, so you can LIST multiple files and search for the same string simply by pressing N in each file, or repeat your search the next time you use LIST.

- Most of the LIST keystrokes can be reassigned with *4DOS.INI* file directives (see pages 128 and 131).

- You can set the default colors used by LIST (and SELECT) using the ListColors directive in *4DOS.INI*. If ListColors is not used the LIST display will use the current screen colors.

Options:

- **/H**(igh 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. If you are displaying a word processor text file and see unusual characters mixed in with the text, try this option.

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

```
c:\> dir | list /s
```

  To redefine the DIR command to always display its output via LIST, use this alias:

```
c:\> alias dir `*dir %& | list /s`
```

  /W(ord wrap): Wrap the text at the right margin. This option is useful when displaying files that don't have a carriage return at the end of each line. The horizontal scrolling keys are intended for use when the display is not wrapped. If you wrap the display with **/W** (or with the W key from within LIST), then use the scrolling keys, the wrapping will change as each line is scrolled. This may produce an unusual display, depending on the structure of the file you are viewing.

LOADBTM

(New)

Purpose: Switch a batch file to or from BTM mode.

Format: **LOADBTM [ON | OFF]**

Usage: 4DOS recognizes two kinds of batch files: *.BAT* and *.BTM*. Batch files executing in BTM mode run two to five times faster than in BAT mode. However, BTM mode should not be used to load memory-resident programs, nor should BTM mode be used for self-modifying batch files. Batch files automatically start in the mode indicated by their file extension, *.BAT* or *.BTM*.

The LOADBTM command turns BTM mode on and off. It can be used to switch modes in either a *.BAT* or *.BTM* file. It can also be used to display the current batch mode from inside a batch file when it is used with no argument.

LOADBTM can only be used within a *.BAT* or *.BTM* file. It is most often used to switch a *.BAT* file into BTM mode after memory-resident programs are loaded, to convert a *.BAT* file to BTM mode without changing its extension, or to switch a *.BTM* file into BAT mode in order to load memory-resident programs.

Using LOADBTM to repeatedly switch modes within a *.BAT* or *.BTM* 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.

The following *.BAT* file fragment loads some memory resident programs (TSRs), and then switches to BTM mode:

```
rem   Because this file has a .BAT extension,
rem   the initial default state is LOADBTM OFF
rem   Loading TSRs...
ansi.com
mouse.com
rem   Switch to high-speed (BTM) mode now that
rem   TSRs are loaded
loadbtm on
path c:\;c:\util;c:\dos
alias /r c:\aliases
```

For more information on *.BTM* and *.BAT* files, see page 97.

LOG

(New)

Purpose: Save a log of commands to a disk file.

Format: **LOG [/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.

/W(rite to).

See also: HISTORY.

Usage: LOG keeps a record of all internal and external commands you use. Each entry includes the current system date and time, along with the actual command after any alias or variable expansion. You can use the log file as a record of your daily activities.

By default, LOG writes commands to the file *4DOSLOG* in the root directory of the drive specified in your COMSPEC environment variable (see page 113).

Entering LOG with no parameters displays the log status (ON or OFF):

```
c:\> log
LOG is OFF
```

To enable or disable logging, add the word "ON" or "OFF" after the LOG command:

```
c:\> log on
```

Entering LOG with *text* writes a message to the log file, even if LOG 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:

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

The LOG output can be used as the basis for writing batch files, but you will probably find HISTORY more effective for this purpose.

Options: **/W**(rite): This switch specifies a different filename for the LOG 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
```

MD / MKDIR

(Compatible)

Purpose: Create a subdirectory.

Format: **MD *pathname...***
or
MKDIR *pathname...*

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

See also: RD.

Usage: MD and MKDIR are synonyms. You can use either one.

MD creates a subdirectory anywhere in the directory tree. To create a subdirectory from the root, start the pathname with a backslash [\]. For example, this command creates a subdirectory called *MYDIR* in the root directory:

```
c:\> md \mydir
```

If no path is given, the new subdirectory is created in the current directory. This example creates a subdirectory called *DIRTWO* in the current directory:

```
c:\mydir> md dirtwo
```

To create a directory from the parent of the current directory (that is, to create a sibling of the current directory), start the pathname with two periods and a backslash [..\].

MD creates one directory at a time. If you need to create the directory *C:\ONE\TWO\THREE* and none of the named directories exist, you must create each directory separately. However, because MD accepts multiple arguments, you can still create all three directories in sequence with one command:

```
c:\> md \one \one\two \one\two\three
```

- DOS will not accept directory names longer than 64 characters. You must be sure that the complete directory name from the root to your deepest subdirectory fits within the 64 character restriction.

MEMORY

(New)

Purpose: Display the amount and status of system RAM.

Format: **MEMORY**

Usage: MEMORY displays information about the RAM in your system. It lists the amount of total RAM in your system and the amount available for applications after DOS, 4DOS, and memory-resident programs have been loaded; the amount of EMS expanded memory, XMS extended memory, and non-XMS extended memory; the HMA status; and the amount of memory 4DOS is using for environment variable space, alias space, and history space:

```
c:\> memory
    655,360 bytes total RAM
    534,464 bytes free

  1,687,552 bytes total EMS memory
  1,097,728 bytes free

    914,432 bytes total XMS memory (HMA in use)

        512 bytes total environment
        195 bytes free

      1,024 bytes total alias
        452 bytes free

      1,024 bytes total history
```

You can use the information from the MEMORY display to fine tune your system, to aid in setting the proper alias and environment sizes in *4DOS.INI*, and to be sure that you have sufficient memory for your largest applications.

If you compare the free RAM displayed by MEMORY with the free RAM displayed by CHKDSK and some memory map programs, MEMORY will usually show a slightly higher value. The difference is the size of the environment passed to these external programs; most memory mapping programs do not count the passed environment as free space, but MEMORY does.

MOVE

(New)

Purpose: Move files to a new directory and drive.

Format: **MOVE [/C /D /H /N /P /Q /R /S /U]** ***source... destination***

source: A file or list of files to move.
destination: The new location for the files.

/C(hanged)
/D(irectory)
/H(idden and system)
/N(othing)
/P(rompt)
/Q(uiet)
/R(eplace)
/S(ubdirectory tree)
/U(pdate)

See also: COPY and RENAME.

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. Like those commands, MOVE never changes the attributes of the files that it operates on.

The simplest MOVE command moves a single *source* file to a new location and, optionally, gives it a new name. These two examples both move one file from drive *C:* to the root directory on drive *A:*

```
c:\> move myfile.dat a:\
c:\> move myfile.dat a:\savefile.dat
```

In both cases, *MYFILE.DAT* is removed from drive *C:* after it has been copied to drive *A:*. If a file called *MYFILE.DAT* in the first example, or *SAVEFILE.DAT* in the second example, already existed on drive *A:*, it would be overwritten. (This demonstrates the difference between MOVE and RENAME. MOVE will move files between drives and will overwrite the destination file if it exists; RENAME will not.)

If you MOVE multiple files, the *destination* must be a directory name. MOVE will move each file into the *destination* directory

with its original name (if the target is not a directory, MOVE will display an error message and exit):

```
c:\> move *.wks *.txt c:\finance\myfiles
```

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

! Be careful when you use MOVE with the SELECT command. If you SELECT multiple files and if the target is not a directory (because of a misspelling, for example) each file will be moved in turn to the target file, overwriting the previous file, and then the original will be erased before the next file is moved. At the end of the command, all of the original files will have been erased and only the last file will exist as the target file. You can avoid this problem by using square brackets with SELECT instead of parentheses (be sure that you don't allow the command line to get too long -- watch the character count in the upper left corner while you're selecting files). MOVE will then receive one list of files to move instead of a series of individual filenames, and it will detect the error and halt. You can also add a backslash [\] to the end of the *destination* name to ensure that it is the name of a subdirectory (see below).

When you move files to another directory, if you add a backslash [\] to the end of the *destination* name MOVE will display an error message if the name does not refer to an existing directory. You can use this feature to keep MOVE from treating a mistyped *destination* directory name as a file name, and attempting to move all *source* files to that name. The **/D** option performs the same function but will also prompt to see if you want to create the *destination* directory if it doesn't exist. The **/S** option always tries to create the *destination* directory if necessary, so this feature will not be effective when **/S** is used.

❖ MOVE first attempts to rename the file(s), which is the fastest way to move files between subdirectories on the same drive. If that fails (the destination is on a different drive or already exists), MOVE will copy the file(s) and then delete the originals.

Options: **/C**(hanged 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(irectory): Requires that the *destination* be a directory. If the *destination* does not exist, MOVE will prompt to see if you want to create it. If the *destination* exists as a file, MOVE will fail with an "Access denied" error. Use this option to avoid having MOVE accidentally interpret your *destination* name as a file name when it's really a mistyped directory name.

- **/H**(idden): Move all files, including hidden and system files.

- **/N**(othing): Do everything except actually move the file(s) This option is most useful for testing what a complex MOVE command will do.

  /P(rompt): Ask the user to confirm each move by pressing **Y** or **N**. An **N** response will skip that particular file.

  /Q(uiet): Don't display filenames as they are moved.

  /R(eplace): Prompt for a **Y** or **N** response before overwriting an existing *destination* file.

- **/S**(ubdirectories): 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. For example, the following alias will "graft" a directory and all of its subdirectories into a new place in the directory tree, and -- because **/D** is used -- will prompt to see if you want to create the first *destination* directory it if it doesn't exist. The **/H** in this alias ensures that any hidden files and subdirectories are also moved:

```
alias graft `move /s/d/h %1\*.* %2`
```

  /U(pdate): 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 files from one directory to another.

PATH

(Enhanced)

Purpose: Display or alter the list of directories that 4DOS will search for executable and batch files 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: SET and ESET.

Usage: When 4DOS is asked to execute an external command (a *.COM*, *.EXE*, *.BTM*, or *.BAT* file or executable extension), it first looks for the file in the current directory. If it fails to find an executable file there, it then searches each of the *directories* specified in the path setting, in the order that they are included.

For example, the following PATH command directs 4DOS to search subdirectories for an executable file in the following order: the current directory, the root directory on *C*, the *DOS* subdirectory on *C*, and the *UTIL* subdirectory on *C*:

```
c:\> path c:\;c:\dos;c:\util
```

The list of *directories* to search can be set or viewed with the PATH command. The list is stored as an environment string, and can also be set or viewed with the SET command and edited with the ESET command.

Directory names in the path must be separated by semicolons [;]. 4DOS shifts each directory name to upper case. This maintains compatibility with programs which can only recognize upper case directory names in the path variable.

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, 4DOS displays the current search path:

```
c:\> path
PATH=C:\;C:\DOS;C:\UTIL
```

If you enter PATH and a semicolon, 4DOS clears the search path and will search only the current directory (this is the default at system startup).

Some applications also use the PATH variable to search for their data files.

- If you include an explicit file extension on the external command name (for example, *WP.EXE*), 4DOS will only look for files with that name and extension in the current directory and every directory in the path setting. It will not look for other executable files with the same base name.

- If you have a directory of a single period [.] in the path, 4DOS will not search the current directory first, but wait until it reaches that point in the path. In rare cases, this feature may not be compatible with applications which use the path to search for their files; if you experience a problem, you will have to remove the "." as a path entry while using any such application.

- 4DOS can create a PATH as long as 250 characters (the command line is limited to 255 characters, and "PATH " takes five). Some applications are written to expect a PATH no longer than the traditional limit of 123 characters. If you have extended your path beyond the traditional limit and experience unusual problems with application programs, see page 153 for tips on resolving the difficulty.

- If you specify an invalid directory in the path, 4DOS will skip that directory and continue searching with the next directory in the path.

PAUSE

(Enhanced)

Purpose: Suspend batch file or alias execution.

Format: **PAUSE [*text*]**

text: The message to be displayed as a user prompt.

Usage: A PAUSE command will suspend execution of a batch file or alias, giving you the opportunity to change disks, turn on the printer, etc.

PAUSE waits for any key to be pressed and then continues execution. You can specify the *text* that PAUSE displays while it waits for a keystroke, or let 4DOS use the default message:

```
Press any key when ready...
```

For example, the following batch file fragment prompts the user before erasing files (the PAUSE command should be entered on one line):

```
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 you want to cancel the batch job.

POPD

(New)

Purpose: Return to the disk drive and directory at the top of the directory stack

Format: **POPD [*]**

See also: DIRS and PUSHD.

Usage: Each time you use the PUSHD command, 4DOS saves the current disk drive and directory on its 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.

This example saves and changes the current disk drive and directory with PUSHD, and then restores it. The command line prompt is set to show the current directory:

```
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.

PROMPT

(New)

Purpose: Change the 4DOS command line prompt.

Format: **PROMPT** [*text*]

text: Text to be used as the 4DOS 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 command sets the command line prompt. The special characters that can be included in a PROMPT command are listed below. For example, to set the prompt to the current date and time, with a ">" at the end:

```
c:\> prompt $d $t $g
Mon  Dec 2, 1991 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
Mon  Dec 2, 1991 10:29:19
C:\>
```

You can include the PROMPT command in your *AUTOEXEC.BAT* file to set the prompt whenever your system is rebooted.

The default prompt is **ng** (drive name plus ">") on drives A and B, and **pg** (current drive and directory plus ">") on all other drives .

If you enter PROMPT with no arguments, 4DOS resets the prompt to the default. 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 4DOS will not use the PROMPT environment variable, in which case the SET command above will give a "Not in environment" error.

- Along with literal text, special characters, and ANSI sequences, you can include the text of any environment variable, 4DOS internal variable, or variable function (see pages 80 and 84) in a prompt. For example, if you want to include the amount of free base memory in the command prompt, plus the current drive and directory, you could use this command:

```
c:\> prompt (%%@dosmem[K]K) $p$g
(601K) c:\data>
```

Notice that the @DOSMEM function is shown with two leading percent signs [%]. If you used only one per cent sign, the @DOSMEM function would be expanded once when the PROMPT command was executed, instead of every time 4DOS displayed the prompt. The result would be that the amount of memory would never change from the value it had when you entered the PROMPT command. You could also use back quotes to delay expanding the variable function until the prompt is displayed:

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

- If you have an ANSI-compatible driver installed, you can include ANSI escape sequences in the PROMPT *text*. 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
```

- Traditionally, it was possible to use the PROMPT command in a batch file to transmit *ANSI.SYS* control sequences to the screen (for example, to redefine function keys). This technique will not work with 4DOS, which doesn't display a prompt within batch files; hence the characters in the PROMPT string are never sent to *ANSI.SYS*. To send ANSI sequences in 4DOS, use the ECHO

command, substituting a Ctrl-X followed by an e for $e in the PROMPT string.

- You may find it helpful to define a different prompt in secondary shells, by using **$z** to display the shell level, or by placing a PROMPT command in your *4START* file and using IF or IFF statements to set the appropriate prompt for different shells.

Prompt Characters:

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: *Tue Jan 1, 1991*
d	Current date, in the format: *Tue 1-01-91*
e	The ASCII ESC character (decimal 27)
f	The close parenthesis [)]
g	The > character
h	BACKSPACE over the previous character
l	The < character
n	Current disk letter
P	Current disk and directory (upper case)
p	Current disk and directory (lower case)
q	The = character
s	The space character
t	Current time, in the format *hh:mm:ss*
v	DOS version number, in the format *3.10*
Xd:	Current directory on drive *d:*, in upper case
xd:	Current directory on drive *d:*, in lower case
z	Current 4DOS shell nesting level; the primary command processor is shell 0.
$	The $ character
_	CR/LF (go to beginning of a new line)

- The format for the date (**$d**) depends on the country code defined in the *CONFIG.SYS* file or by the CHCP command. The default format is U.S. (*mm-dd-yy*). The European format is *dd-mm-yy*; the Japanese is *yy-mm-dd*. The separator used for the time (**$t**) will also be changed based on the country definition.

PUSHD

(New)

Purpose: Save the current disk drive and directory, optionally moving to a new drive and directory.

Format: **PUSHD [*pathname*]**

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

See also: DIRS, POPD and CDPATH on page 114.

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:

```
c:\> pushd
c:\>
```

If a *pathname* is specified as part of the PUSHD command, the current drive and directory are saved and the specified pathname becomes the new current drive and directory. If the *pathname* includes a drive letter, PUSHD changes to the specified directory on the new drive without changing the default current directory on the original drive.

This example saves the current directory and changes to *C:\WORDP\MEMOS*, then returns to the original directory with POPD:

```
c:\> pushd \wordp\memos
c:\wordp\memos> popd
c:\>
```

- The directory stack can hold up to 255 characters or about 10 to 20 entries depending on the length of the directory names. If you exceed the directory stack size, the oldest directory is removed before adding the current directory.

- If PUSHD can't change to the specified directory, it will look for the CDPATH environment variable. PUSHD will append the specified directory name to each directory in CDPATH and attempt to change to that drive and directory, until the first match or the end of the CDPATH argument. This allows you to use CDPATH as a quick way to find commonly used subdirectories which have unique names. For example, if you are currently in the directory C:\WP\LETTERS\JANUARY and you'd like to change to D:\SOFTWARE\UTIL, you could enter the command:

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

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

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

  and 4DOS will change to D:\SOFTWARE\UTIL.

QUIT

(New)

Purpose: Terminate the current batch file.

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

value: The exit code from 0 to 255 to return to 4DOS or to the previous batch file.

See also: CANCEL.

Usage: QUIT provides a simple way to exit a batch file before reaching the end of the file. If you QUIT a batch file called from another batch file, you will be returned to the previous file at the line following the original call.

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
```

To end all batch file processing, use the CANCEL command.

- If you specify a *value*, QUIT will set the ERRORLEVEL or exit code (see the IF command, and the %? variable on page 81) to that value.

- You can also use QUIT in 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 4DOS command prompt or to the calling batch file.

RD / RMDIR (Enhanced)

Purpose: Remove one or more subdirectories.

Format: **RD *pathname...***
or
RMDIR *pathname...*

pathname: The name of a subdirectory to remove.

See also: MD.

Usage: RD and RMDIR are synonyms. You can use either one.

RD removes directories from the directory tree. For example, to remove the subdirectory *MEMOS* from the subdirectory *WP*, you can use this command:

```
c:\> rd \wp\memos
```

Before using RD, you must delete all files and subdirectories (and their files) in the *pathname* you want to remove. Remember to remove hidden and read-only files as well as normal files.

- To use a single command to remove entire subdirectory trees including all files, see the /S and /X options of the DEL command.
- You cannot remove the root directory, the current directory (.), or any directory above the current directory in the directory tree.
- You can use wildcards in the *pathname*.

REBOOT

(New)

Purpose: Do a warm or cold system reboot.

Format: **REBOOT [/C /V]**

/C(old reboot) **/V**(erify)

❖ Usage: REBOOT will completely restart your computer. It is comparable to pressing Ctrl-Alt-Delete (a warm reboot) or to turning the power off and back on or pressing the reset button (a cold reboot). A reboot is necessary to activate any changes to your *CONFIG.SYS* file, and may also be used if you wish to restart the system with an altered *4START* or *AUTOEXEC.BAT* file.

The following example prompts you to verify the reboot, then does a cold boot:

```
c:\> reboot /c/v
```

REBOOT defaults to performing a warm boot, with no prompting.

REBOOT flushes the disk buffers, resets the drives, and waits one second before rebooting, to allow disk caching programs to finish writing any cached data .

! Some system BIOSes, memory managers, multitaskers, or memory-resident programs (TSRs) may intercept attempts to reboot your system, and defeat them entirely, convert a cold boot request to a warm boot or vice versa, or in very rare cases, hang the system -- requiring a reboot! As a result you may need to experiment with which reboot options work best for your system hardware and software configuration, and under rare circumstances REBOOT may not be useable on your system.

Options: **/C**(old): Do a "cold" reboot. This is similar to turning the power off and back on, and may be necessary to properly initialize the system. REBOOT /C may not physically reset all hardware devices as thoroughly as actually turning off the power; its effect depends on the internal design of each hardware device and on your system configuration.

/V(erify): Prompt for confirmation (**Y** or **N**) before rebooting.

REM

(Compatible)

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. Comments can be up to 255 characters long. 4DOS will ignore everything on the line after the REM command (including quote characters, redirection symbols, and other commands).

If ECHO is ON, 4DOS will display the comment. Otherwise, 4DOS will ignore it.

If ECHO is ON and you don't want to display the line, preface the REM command with the @ character.

REN / RENAME (Enhanced)

Purpose: Rename files or subdirectories.

Format: **REN [/N /P /Q /S]** ***old_name... new_name***
or
RENAME [/N /P /Q /S] ***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.

/N(othing) **/Q**(uiet)
/P(rompt) **/S**(ubdirectory)

See Also: COPY and MOVE.

Usage: REN and RENAME are synonyms. You may use either one.

REN lets you change the name of a file or a subdirectory. You can also use REN to move one or more files to a new subdirectory on the same drive. (If you want to move files to a different drive, use MOVE.)

In its simplest form, you simply give REN the *old_name* of an existing file or subdirectory and then a *new_name*. The *new_name* must not already exist -- you can't give two files the same name (unless they are in different directories). The first example renames the file *MEMO.TXT* to *OFFICE.TXT*. The second example changes the name of the *\WORDPROC* directory to *\WP*:

```
c:\> rename memo.txt office.txt
c:\> rename \wordproc \wp
```

You can also use REN to rename a group of files that you specify with wildcards, as multiple files, or in an include list. When you do, the *new_name* must use one or more wildcards to show what part of each filename to change. Both of the next two examples change the extensions of multiple files to *.SAV*:

```
c:\> ren config.sys autoexec.bat 4start.btm *.sav
c:\> ren *.txt *.sav
```

REN can move one or more 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 c:\wp\memos\
c:\> ren oct.dat nov.dat c:\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. 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 c:\data\save\*.sav
```

When *new_name* refers to a file or files (rather than a directory), the file(s) must not already exist. Also, you cannot rename a subdirectory to a new location on the directory tree.

- REN does not change a file's attributes. The *new_name* file(s) will have the same attributes as *old_name*.

Options:

/N(othing): Do everything except actually rename the file(s). This option is useful for testing what a REN command will actually do.

/P(rompt): Ask the user to confirm each move by pressing **Y** or **N**. An **N** response will skip that particular file.

/Q(uiet): Don't display filenames as they are renamed. This option is most often used in batch files.

/S(ubdirectory): Normally, you can rename a subdirectory only if you do not use any wildcards in the *old_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.

RETURN

(New)

Purpose: Return from a GOSUB (subroutine) in a batch file.

Format: **RETURN**

See also: GOSUB.

❖ Usage: 4DOS allows subroutines in batch files. A subroutine begins with a label (a colon followed by a word) and ends with a RETURN command. The subroutine is invoked with a GOSUB command from another part of the batch file. The RETURN command ends a subroutine; execution of the batch file will continue on the line following the original GOSUB.

The following batch file fragment calls a subroutine which displays the current directory:

```
echo Calling a subroutine
gosub subr1
echo Returned from the subroutine
quit

:subr1
dir /a/w
return
```

SCREEN

(New)

Purpose: Position the cursor on the screen and optionally display a message.

Format: **SCREEN *row column* [*message*]**

row: The new row location for the cursor.
column: The new column location for the cursor.
message: Optional text to display at the new cursor location.

See also: ECHO, SCRPUT, TEXT, and VSCRPUT.

Usage: SCREEN allows you to create attractive screen displays in batch files. You use it to specify where a message will appear on the screen. You can use SCREEN to create menu displays, logos, etc. The following batch file fragment displays a menu:

```
@echo off ^ cls
screen 3 10  Select a number from 1 to 4:
screen 6 20  1 - Word Processing
screen 7 20  2 - Spreadsheet
screen 8 20  3 - Telecommunications
screen 9 20  4 - Quit
```

SCREEN does not change the screen colors. If you have *ANSI.SYS* installed and have set colors with CLS or COLOR, those colors will be used for the display. To display text in specific colors, use SCRPUT or VSCRPUT.

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.

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

SCRPUT

(New)

Purpose: Position the cursor on the screen and display a message in color.

Format: **SCRPUT *row col* [BRIght] [BLInk] *fg* ON *bg text***

row: Starting row
col: Starting column
fg: Foreground character color
bg: Background character color
text: The text to display

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

See also: CLS, ECHO, SCREEN, TEXT, and VSCRPUT.

Usage: SCRPUT allows you to create attractive screen displays in batch files. You use it to specify where a message will appear on the screen and what colors will be used to display the message text. You can use SCRPUT to create menu displays, logos, etc.

SCRPUT works like SCREEN, but allows you to specify the display colors. It writes directly to the screen and does not require an ANSI driver.

Only the first three characters of the color name and the attributes BRIGHT and BLINK are required. The row and column are zero-based, so on a standard 25 line by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79.

The following batch file fragment displays a menu in color (each SCRPUT command should be entered on one line):

```
@echo off ^ cls white on blue
scrput 3 10 bri whi on blue Select a number from
  1 to 4:
scrput 6 20 bri red on blue 1 - Word Processing
scrput 7 20 bri yel on blue 2 - Spreadsheet
scrput 8 20 bri gre on blue 3 -
Telecommunications
scrput 9 20 bri mag on blue 4 - Quit
```

SELECT

(New)

Purpose: Interactively select files for a command.

Format: **SELECT [/A[:][-]rhsda /C /O[:][-]deginsu] [*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) **/O**(rder)
/C(ase -- use upper case)

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 a 4DOS internal command or alias, an external command, or a batch file.

If you use parentheses around the *files*, SELECT executes the *command* once for each file you have selected. During each execution, one of the selected files is passed to the *command* as an argument. If you use square brackets around *files*, the SELECTed files are combined into a single list, separated by spaces. The command is then executed once with the entire list presented as its command-line arguments.

SELECT uses the cursor up, cursor down, PgUp, and PgDn keys to scroll through the file list. Use the + key or the **spacebar** to select a file (or unselect a marked file), and the - key to unselect a file. The * key will reverse all of the current marks (excluding subdirectories), and the / key will unmark everything. After marking the files, press **Enter** to execute the command.

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

To skip the files listed in the current display and go on to the next file specification inside the parentheses or brackets (if any), press

the Escape key. To cancel the current SELECT command entirely, press Ctrl-C or Ctrl-Break.

In the simplest form of SELECT, you merely specify the command and then the list of files from which you will make your selection(s). For example:

```
c:\> select copy (*.com *.exe) a:\
```

will let you select from among the *.COM* and *.EXE* files on the current drive. It will then invoke the COPY command to copy each file you select to drive A:. You will be able to select first from a list of all *.COM* files in the current directory, and then from a list of all *.EXE* files.

If you want to select from a list of all the *.COM* and *.EXE* files mixed together, create an include list inside the parentheses by inserting a semi-colon (see page 74 for information on include lists):

```
c:\> select copy (*.com;*.exe) a:\
```

Finally, if you want the SELECT command to send a single list of files to COPY, instead of invoking COPY once for each file you select, put the file names in square brackets instead of parentheses:

```
c:\> select copy [*.com;*.exe] a:\
```

If you use brackets, you have to be sure that the resulting command (the word COPY, the list of files, and the destination drive in this example) is no more than 127 characters long for external commands and no more than 255 characters long for internal 4DOS 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.

- 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 (enter the alias on one line):

```
c:\> alias utils select
     c:\util\*.com;*.exe;*.btm;*.bat)
```

- If you want to use filename completion (see page 60) to enter the filenames inside the parentheses, type a space after the opening parenthesis. Otherwise the command line editor will treat the open parenthesis as the first character of the filename.

- You can set the default colors used by SELECT (and LIST) using the ListColors directive in *4DOS.INI* (see page 127). If ListColors is not used, the default colors will be set by the StdColors directive (page 127) or by the last CLS or COLOR command.

- If you have an ANSI driver loaded, you can display the filenames in color by setting the COLORDIR environment variable (the same colors will be used by the DIR command). The format for COLORDIR is:

```
ext ... :[BRIght][BLInk] fg [ON bg]; ...
```

where *ext* is the file extension, or one of the following file attributes:

```
DIRS      directory
RDONLY    read-only file
HIDDEN    hidden file
SYSTEM    system file
ARCHIVE   file modified since last backup.
```

For example, to display the *.COM* and *.EXE* files in red, the .C and .ASM files in bright cyan, and the read-only files in blinking green (enter this on one line):

```
c:\> set colordir=com exe:red; c asm:bright cyan;
     rdonly:blink green
```

If you don't specify a background color, SELECT will use the current screen background color from the ListColors or StdColors directive or the last CLS or COLOR command (as described

above). COLORDIR will not work properly unless you have an ANSI driver loaded.

❖ You may need to increase 4DOS's internal stack size using the StackSize directive in *4DOS.INI* if you use extremely complex combinations of commands like EXCEPT, FOR, GLOBAL, IF, and SELECT on the same command line, or use complex combinations of these commands in nested batch files or nested GOSUBs. See the StackSize directive on page 132 for more information.

Options: ❖ **/A**(ttribute): Display only those files that have the specified attribute set. Preceding the attribute character with a minus **[-]** will display those files that DON'T have that attribute set. Attributes can also be combined. The attributes are:

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

/C(ase): Display filenames in the traditional upper case format; also see SETDOS /U and the UpperCase directive in *4DOS.INI*.

/O(rder): Set the sort order for the files. The order can be any combination of the following options:

-	Reverse the sort order
d	Sort by date and time (oldest first)
e	Sort by extension
g	Group subdirectories together
i	Sort by the file description
n	Sort by filename (this is the default)
s	Sort by size
u	Unsorted

SET

(Enhanced)

Purpose: Display, create, modify, or delete environment variables.

Format: **SET [/M /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.

/M(aster)
/P(ause)
/R(ead from file)

See also: UNSET and ESET.

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. 4DOS itself uses several environment variables (see page 113). See page 77 for more information on the environment.

If you simply type the SET command with no options or arguments, it will display all the names and values currently stored in the environment. Typically, you will see an entry called COMSPEC, an entry called PATH, an entry called CMDLINE, and whatever other environment variables you and your programs have established:

```
c:\> set
COMSPEC=C:\4DOS.COM
PATH=C:\;C:\DOS;C:\UTIL
CMDLINE=E:\UTIL\MAPMEM.EXE
```

To add a variable to the environment, type SET plus the variable name, an equal sign, and the text:

```
c:\> set mine=c:\finance\myfiles
```

4DOS will convert the variable name to upper case but leave the text after the equal sign 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*.

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=
```

4DOS limits the variable *name* to a maximum of 80 characters, and the name plus the *value* to a maximum of 255 characters.

❖ Unless you use **/M**, SET only affects the environment of the current command processor and the programs it executes. If you EXIT to a parent command processor, the original environment will be unchanged.

The size of the environment is specified by the Environment and EnvFree directives in *4DOS.INI* (see page 122) or by the **/E:** startup switch (see page 109).

Options: ❖ **/M**(aster): Display or modify the master environment rather than the local environment. This option only makes sense in a secondary command processor.

/P(ause): Pause after displaying each page of environment entries. Press Ctrl-C to quit, or any other key to display the next page.

❖ **/R**(ead): Read environment variables from a file. This is much faster than loading variables from a batch file with multiple SET commands. The file is in the same format as the SET display, so SET /R can accept as input a file generated by redirecting SET output. For example, the following commands will save the environment variables to a file, and then reload them from that file:

```
set > varlist
set /r varlist
```

You can load variables from multiple files by listing the filenames individually after the /R. You can add comments to a variable file by starting the comment line with a colon [:].

SETDOS

(New)

Purpose: Display or set the 4DOS configuration.

Format: **SETDOS [/A? /C? /E? /I+ | - command /L? /M? /N? /R? /S?:? /U? /V?]**

/A(NSI)
/C(ompound)
/E(scape character)
/I(nternal commands)
/L(ine)
/M(ode for editing)
/N(o clobber)
/R(ows)
/S(hape of cursor)
/U(pper case)
/V(erbose)

Usage: SETDOS allows you to customize certain aspects of 4DOS to suit your personal tastes or the configuration of your system. Each of these options is described below.

You can display the value of all SETDOS options by entering the SETDOS command with no parameters.

Most of the SETDOS options can be initialized when 4DOS starts through directives in the *4DOS.INI* file (see page 124). The name of the corresponding directive is listed in square brackets [ ] with each option; if none is listed, that option cannot be set from the *4DOS.INI* file. You can also define the SETDOS options in your *AUTOEXEC.BAT* or 4START file, in aliases, or at the command line.

Secondary shells automatically inherit the configuration settings currently in effect in the previous shell. If values have been changed by SETDOS since 4DOS started, the new values will be passed to the secondary shell. For details on inheritance of SETDOS values by secondary shells and their relationship to *4DOS.INI*, see page 118.

Many of the options below are marked with ❖. If you are a new user, skip these and read the **/M**, **/S**, and **/U** options, which are more common.

Options: ❖ **/A**(NSI) [ANSI]: The ANSI option determines whether 4DOS will attempt to use ANSI escape sequences for the CLS and

COLOR commands. 4DOS normally determines this itself, but if you are using a non-standard ANSI driver or your loading sequence is unusual, you may need to explicitly inform 4DOS. **/A0** allows 4DOS to determine whether an ANSI driver is installed (the default). **/A1** forces 4DOS to assume an ANSI driver is installed. **/A2** forces 4DOS to assume an ANSI driver is not installed.

- **/C**(ompound character) [CommandSep]: The COMPOUND 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 any of the whitespace characters (blank, tab, comma, or equal sign) as the command separator. This example changes the COMPOUND character to a tilde [~]:

```
c:\> setdos /c~
```

- **/E**(scape character) [EscapeChar]: The ESCAPE option sets the character used to suppress the normal meaning of the following character. Any character following the escape character will be passed unmodified to the command line. For example, you could include a redirection character such as > as part of a command-line argument if the character was preceded by the escape character. The default escape character is Ctrl-X (ASCII 24; appears on screen as an up-arrow [↑]). You cannot use any of the redirection characters (| > <) or the whitespace characters (blank, tab, comma, or equal sign) as the escape character. Certain characters (**b**, **e**, **f**, **n**, **r**, and **t**) have special meanings when immediately preceded by the escape character. See page 91 for additional details.

- **/I**(nternal): The INTERNAL option allows you to disable or enable internal 4DOS commands. To disable a command, precede the command name with a minus [-]. To re-enable a command, precede it with a plus [+]. For example, to disable the internal LIST command to force 4DOS to use an external command:

```
c:\> setdos /i-list
```

- **/L**(ine) [LineInput]: The LINE option controls how 4DOS gets its input from the command line. **/L0** tells 4DOS to use character

input (the default). **/L1** tells 4DOS to use line input (via DOS service INT 21h function 0Ah, like *COMMAND.COM*). **/L1** will disable command line editing, history recall, and filename completion; it should only be used if it is needed for compatibility with a specific program. If you have a program that requires line input, you can use the following line in an alias or batch file to change the line input option just for that single program:

```
setdos /L1 ^ program %& ^ setdos /L0
```

See the file *APPNOTES.DOC* for information on programs which require this option.

/M(ode) [EditMode]: The MODE option controls the initial line editing mode. **/M0** forces 4DOS to start editing in overstrike mode (the default). **/M1** forces 4DOS to start editing in insert mode.

- **/N**(o clobber) [NoClobber]: The NOCLOBBER option controls output redirection (see page 66). **/N0** means existing files will be overwritten by simple 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 simple output redirection, and that when appending the output file must exist. A **/N1** setting can be overridden with the **[!]** character. If you use **/N1**, you may have problems with a few unusual programs that shell to DOS to run a command with redirection, and expect to be able to overwrite an existing file.

- **/R**(ows) [ScreenRows]: The ROWS option sets the number of screen rows used by the video display. Normally 4DOS detects the screen size, but if you have a non-standard display you may need to set it explicitly. This option does not affect screen scrolling (that is controlled by your video BIOS or *ANSI.SYS*); it is used only for LIST, SELECT, the paged output options (i.e., TYPE /P), and error checking in the screen output commands.

  /S(hape) [CursorOver, CursorIns]: The SHAPE option sets the 4DOS cursor shape. The format is ***/So:i*** where ***o*** is the cursor size for overstrike mode, ***i*** the cursor size for insert mode. The size is entered as a percentage of the total character height. The default values are 10:100 (an underscore cursor for overstrike mode, and

a block cursor for insert mode). Because of the way video BIOSes remap the cursor shape, you may not get a smooth progression in the cursor size from 0% - 100%. To disable the cursor, enter **/S0:0**.

/U(pper) [UpperCase]: The UPPER option controls the default case (upper or lower) for filenames displayed by 4DOS internal commands like COPY and DIR. **/U0** displays file names in lower case (the default). **/U1** displays file names in the traditional upper case.

- **/V**(erbose) [BatchEcho]: The VERBOSE option controls the default for command echoing in batch files. **/V0** disables echoing of batch file commands unless ECHO is explicitly set ON. **/V1** enables echoing of batch file commands unless ECHO is explicitly set OFF. **/V1** is the default.

SETLOCAL (New)

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

Format: **SETLOCAL**

See also: ENDLOCAL.

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

For example, this batch file fragment saves everything, changes the disk and directory, modifies some variables, runs a program, and then restores the original values:

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

SETLOCAL and ENDLOCAL are not nestable within a batch file. However, you can have multiple SETLOCAL / ENDLOCAL pairs within a batch file, and nested batch files can each have their own SETLOCAL / ENDLOCAL. You cannot use SETLOCAL in an alias or at the command line.

- 4DOS automatically performs an ENDLOCAL at the end of a batch file if you forget to do so. If you invoke one batch file from another without using CALL, the first batch file is terminated, and an automatic ENDLOCAL is performed. The second batch file inherits the drive, directory, aliases, and environment variables as they were prior to any unterminated SETLOCAL.

- Do not load memory-resident programs (TSRs) from a batch file while SETLOCAL is in effect. If you do, when ENDLOCAL is executed and the memory used by SETLOCAL is released, a "hole" will be left in memory below the TSR. This is not usually harmful, but wastes memory.

SHIFT

(Enhanced)

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

Format: **SHIFT [*n*]**

Usage: SHIFT is provided for compatibility with older batch files, where it was used to access more than 10 replaceable parameters. 4DOS supports 128 replaceable parameters (%0 to %127), so you may not need to use SHIFT for batch files running exclusively under 4DOS.

SHIFT moves each of the batch file replaceable 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 4DOS 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 zero one two three four five six

zero one two three
one two three four
three four five six
two three four five
```

SWAPPING (New)

Purpose: Enable or disable 4DOS swapping, or display the swapping state.

Format: **SWAPPING [ON | OFF]**

❖ Usage: SWAPPING temporarily disables or enables the swapping of the transient portion of 4DOS to EMS expanded memory, to XMS extended memory, or to disk (see page 122).

Setting **SWAPPING OFF** is particularly useful for speeding up batch files (including *AUTOEXEC.BAT*) when 4DOS is using disk swapping. When you are running several small programs from a batch file, disk swapping can sometimes cause a noticeable delay. However, if you disable swapping, there will be about 88K memory available for large application programs.

The following batch file fragment disables swapping, runs several programs, and then re-enables swapping:

```
swapping off
c:\util\mouse
c:\video\ansi.com
cls bright white on blue
c:\bin\cache.com
swapping on
```

If you enter **SWAPPING** with no arguments, 4DOS displays the current swapping type (XMS, EMS, Disk, or None) and state:

```
c:\> swapping
SWAPPING (XMS) is ON
```

Setting SWAPPING OFF does not close the disk swap file or release any reserved EMS or XMS memory.

You may have trouble if you load memory-resident programs (TSRs) with SWAPPING OFF and unload them with SWAPPING ON, or vice versa. Many TSRs expect the system to be in the same state when they unload that it was in when they loaded, and variation from this norm may cause the TSR to unload improperly or hang your system, requiring a reboot.

TEE

(New)

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; redirection options (page 66).

❖ Usage: Many programs get their input from "standard input" and send their output to "standard output," which are normally the keyboard and video display (known collectively as the console or CON). You can redirect both the input and output of such programs, for example, using a file either to provide the input or collect the output.

TEE gets its input from standard input and sends out two copies: one goes to standard output, the other to the *file* or *files* that you specify. It is most often used with a redirection pipe [|] to capture intermediate output before the data is altered by another program or command.

For example, to search the file *DOC* for any lines containing the string "4DOS", make a copy of the matching lines in *4.DAT*, sort the lines, and write them to the output file *4D.DAT*:

```
c:\> find "4DOS" doc | tee 4.dat | sort > 4d.dat
```

If you are typing at the keyboard to produce the input for TEE, you must enter a Ctrl-Z to terminate the input.

Option: **/A**(ppend): Add the output to the file(s) rather than overwriting them.

TEXT

(New)

Purpose: Display a block of text in a batch file.

Format: **TEXT**

.
.
.

ENDTEXT

See also: ECHO, SCREEN, SCRPUT, and VSCRPUT.

Usage: The TEXT command is useful for displaying menus or multi-line messages from a batch file. 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.

If you have an ANSI driver loaded, you can change screen colors by inserting ANSI escape sequences anywhere in the text block. You can also 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 - DOS Utilities
Enter your selection :
endtext
```

TIME

(Compatible)

Purpose: Display or set the current system time.

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

hh hour, 0 - 23
mm minute, 0 - 59
ss second, 0 - 59

See also: CHCP and 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 2, 1991  9:30:10
New time (hh:mm:ss):
```

TIME defaults to 24-hour format, but you can optionally enter the time in 12-hour format by appending an "am" or "pm" to the time you enter.

For example, to enter the time as 9:30 am:

```
c:\> time 9:30 am
```

DOS adds the system time and date to the directory entry of every file you create and modify. If you keep both the time and date accurate, you will have a record of when you last updated each file.

The separator used by TIME depends on the country code defined in the *CONFIG.SYS* file or by the CHCP command.

TIMER

(New)

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 is .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(plit): 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).

TRUENAME (New)

Purpose: Find the full, true path and file name for a file.

Format: **TRUENAME *file***

file: The file whose name TRUENAME will report.

See also: **@truename** variable function on page 87.

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 requires MS-DOS or PC-DOS 3.0 or above.

TYPE

(Enhanced)

Purpose: Display the contents of the specified file(s).

Format: **TYPE [/L /P]** ***file...***

file: The file or list of files that you want to display.

/L(ine numbers) **/P**(ause)

See also: LIST.

Usage: The TYPE command displays a file. It is normally only useful for displaying ASCII text files. Executable files (*.COM* and *.EXE*) and many data files may be unreadable when displayed with TYPE because they include non-alphanumeric characters.

To display the files *MEMO1* and *MEMO2*:

```
c:\> type /p memo1 memo2
```

You can press Ctrl-S to pause TYPE's display and then any key to continue.

You will probably find LIST to be more useful for displaying files. However, the TYPE /L command used with redirection (see page 66) is useful if you want to add line numbers to a file.

Options: **/L**(ine numbers): Print a line number preceding each line of text.

/P(ause): Wait for a keystroke after displaying each page. Press Ctrl-C to quit, or any other key to continue.

UNALIAS

(New)

Purpose: Remove aliases from the alias list.

Format: **UNALIAS *alias...***
or
UNALIAS *

alias: One or more aliases to remove from memory.

See also: ALIAS and ESET.

Usage: 4DOS 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 *
```

UNSET

(New)

Purpose: Remove variables from the environment.

Format: **UNSET [/M]** ***name...***
or
UNSET *

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

/M(aster environment)

See also: SET and ESET.

Usage: See the SET command and page 77 for a discussion of environment variables.

UNSET removes one or more variables from the environment. For example, to remove the variable CMDLINE:

```
c:\> unset cmdline
```

If you use the command **UNSET ***, all of the environment variables will be deleted:

```
c:\> unset *
```

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

! Use caution when removing environment variables, and especially when using UNSET *. Many programs will not work properly without certain environment variables; 4DOS itself depends on PATH and COMSPEC.

Option: ❖ **/M**(aster): Remove the variable from the master environment rather than the local environment. This option only makes sense if used in a secondary command processor.

VER (Enhanced)

Purpose: Display the current 4DOS and DOS versions.

Format: **VER**

Usage: Both the 4DOS and DOS version numbers consist of a one-digit major version number, a period, and a one- or two-digit minor version number. The VER command displays both version numbers:

```
c:\> ver
4DOS 4.0  DOS 5.0
```

VERIFY

(Compatible)

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 written with the data actually placed on disk.

If used without any parameters, VERIFY will display the state of the DOS verify flag:

```
c:\> verify
VERIFY is OFF
```

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.

VOL

(Enhanced)

Purpose: Display a 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 DOS external LABEL command. Also, every floppy disk formatted with DOS version 4.0 or above has a volume serial number.

The VOL command will display the volume label and, if you are using DOS 4.0 or later, 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 you are using DOS 4.0 or later, the disk 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

(New)

Purpose: Display text vertically in the specified color.

Format: **VSCRPUT *row col* [BRIght] [BLInk] *fg* ON *bg text***

row: Starting row number.
col: Starting column number
fg: Foreground text color
bg: Background text color
text: The text to display

The available colors are:

Black	Blue	Green	Red
Magenta	Cyan	Yellow	White

See also: SCRPUT.

Usage: VSCRPUT writes text vertically on the screen rather than horizontally. Like the SCRPUT command, it uses the colors you specify to write the text. VSCRPUT can be used for simple graphs and charts generated by batch files.

The row and column are zero-based, so on a standard 25 row by 80 column display, valid rows are 0 - 24 and valid columns are 0 - 79.

Only the first three characters of the color name and the attributes BRIGHT and BLINK are required.

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
```

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

Y

(New)

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.

If you are typing at the keyboard to produce input text for Y, you must enter a Ctrl-Z to terminate the input.

APPENDIX A / 4DOS ERROR MESSAGES

This appendix lists error messages generated by 4DOS, and includes a recommended course of action for most errors. If you are unable to resolve the problem look through Chapter 7 / Using 4DOS With Your Hardware and Software, beginning on page 135, or contact JP Software for technical support (see page 7).

Error messages relating to files are generally reports of errors returned by DOS. You may find some of these messages (for example, "Access denied") vague enough that they are not always helpful. 4DOS includes the file name in file error messages, but is often unable to determine a more accurate explanation of these DOS errors. The message shown is the best information available based on the error codes returned by DOS.

The following list includes all error messages, in alphabetical order:

4DOS initialization error --: An error occurred during the 4DOS startup process. Look up the rest of the message in this list for a more specific explanation.

4DOS server error --: An error occurred in communication between 4DOS's resident and transient portions. Look up the rest of the message in this list for a more specific explanation.

4DOS swapping failed, loading in memory-resident mode: None of the swapping options worked. Check your **Swapping** specification in *4DOS.INI*, and/or free some XMS or EMS memory or disk space.

4DOS unrecoverable error XX: An error occurred in the resident portion of 4DOS. These errors will terminate secondary shells and require a reboot if they occur during a primary shell or if 4DOS cannot continue.

BI	Bad server function code. Contact JP Software.
DI	Same as Disk swap file corrupted.
DR	Same as Swap file read error.
DS	Same as Swap file seek error.
EI	Same as EMS mapping error.
NS	No number for new shell. You have started too many 4DOS secondary shells without properly exiting some of them, perhaps by closing DESQView windows rather than EXITing. Clean up any work in process and reboot the system.
PT	Illegal process termination. Contact JP Software.
TS	Terminated inactive shell. Contact JP Software.
XI	Same as XMS move failed.

Access denied: You tried to write to or erase a read-only protected file, to rename a file or directory to an existing name, to create a directory that already exists, or to remove a read-only directory or a directory with files or subdirectories still in it.

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.

Ambiguous directive name: The name of a *4DOS.INI* directive was not fully spelled out and was therefore ambiguous. Spell out the directive more fully to make its name unambiguous.

Attempt to exit from root shell: Another program has destroyed a portion of 4DOS's memory. Reboot the system; if the error persists, contact JP Software.

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

Bad environment: The DOS environment has a bad structure, probably because a program destroyed 4DOS's master environment space. Reboot the system.

Batch file missing: 4DOS can't find the batch (*.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 file to itself: 4DOS will not permit you to COPY or MOVE a file to itself. 4DOS performs full path and filename expansion before copying to ensure that files aren't inadvertently destroyed.

Can't create: 4DOS can't create the specified file. The disk may be full or write protected, or the file already exists and is read-only, or the root directory is full.

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

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

Can't make directory entry: 4DOS can't create the filename in the directory. This is usually caused by a full root directory. Create a subdirectory and move some of the files to it.

Can't open: 4DOS 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 DOS does not allow. Change to the parent directory and try again.

Can't set up disk swap file: The disk swap file you specified cannot be opened. The path or drive is invalid, the disk is full, DOS is out of file handles, or there is a hardware problem. Check *4DOS.INI* to be sure your **Swapping** directive is correct.

Command line too long: A single command exceeded 255 characters, or the entire command line exceeded 511 characters, during alias and variable expansion. To address this reduce the complexity of the command, or use a batch file. This error also occurs if you pass an extremely long command line to 4DOS on your SHELL= line in *CONFIG.SYS* and 4DOS does not have room for your command line and the necessary directory information for the COMSPEC environment variable. In this case, place the commands in a batch file and invoke the batch file from your SHELL= line.

Contents lost before copy: COPY was appending files, and found one of the source files is the same as the target. That source file is skipped, and appending continues with the next file.

Data error: DOS can't read or write properly to the device. On a floppy drive, this error is usually caused by a defective floppy disk, dirty disk drive heads, or a misalignment between the heads on your drive and the drive on which the disk was created. On a hard drive, this error may indicate a drive that is too hot or too cold, or a hardware problem. Retry the operation; if it fails again, correct the hardware or diskette problem.

Directory stack empty: POPD or DIRS can't find any entries in the directory stack.

Disk is write protected: The disk cannot be written to. Check the disk and remove the write-protect tab or close the write-protect window if necessary.

Disk swap file corrupted: The 4DOS disk swapping file (*4DOSSWAP.nnn*) has been moved, deleted, or damaged by another program. Reboot the system.

Drive not ready--close door: The floppy disk drive door is open. Close the door and try again.

EMS deallocation failed: 4DOS can't deallocate EMS memory when exiting from a secondary shell. The EMS map has been corrupted or the memory area used by 4DOS or the EMS driver has been destroyed by a program. Clean up any work in process and reboot the system.

EMS map save or restore failed: 4DOS cannot save or restore the EMS page map. The EMS map has been corrupted, memory has been destroyed by a program, or you have an incompatible EMS driver. If this error recurs, try another swapping method, update your EMS driver, or contact JP Software.

EMS mapping failed: 4DOS can't map EMS pages when swapping to or from EMS. The EMS map has been corrupted or the memory area used by the loader or the EMS driver has been destroyed by a program. Reboot the system.

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 a *4DOS.INI* directive on the SHELL= line in *CONFIG.SYS* or on the startup command line for a secondary shell, but the directive is in error. A more specific error message follows.

Error on line N of 4DOS.INI: There is an error in *4DOS.INI*. This message is followed by a more specific message, which can be looked up in this list, and the text of the incorrect directive. Correct the directive and reboot for the change to take effect.

Error reading: DOS 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: DOS 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.

Fatal error -- please reboot: 4DOS cannot continue due to the previous error. Reboot the system.

File Allocation Table bad: DOS can't access the FAT on the specified disk. This can be caused by a bad disk, a hardware error, or an unusual software interaction.

File exists: The requested output file already exists, and 4DOS won't overwrite it.

File not found: 4DOS couldn't find the specified file. Check the spelling or path name.

General failure: This is usually a hardware problem, particularly a disk drive failure or a device not properly connected to a serial or parallel port. Try to correct the problem or reboot and try again.

INI file processing error at line n, remainder of file skipped: An input error (such as a data error) has prevented 4DOS from fully processing your *4DOS.INI* file. Check that the file is readable by another program, or TYPE it to ensure that 4DOS reads the file properly.

Insufficient disk space: COPY or MOVE ran out of room on the destination drive. Remove some files and retry the operation.

Internal DOS error: DOS encountered an internal bug and failed. Reboot the system.

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 *4DOS.INI*.

Invalid color: You gave an invalid value for a color directive in *4DOS.INI*.

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

Invalid drive: A bad or non-existent disk drive was specified.

Invalid DOS version: You need a newer version of DOS to execute the specified command.

Invalid INI file path or name, file not processed: The path or name for the initialization file on the SHELL= line in *CONFIG.SYS* or on the startup command line for a secondary shell. Correct the **@d:\path\inifile** option to name the correct file.

Invalid directive name: 4DOS can't recognize the name of a directive in your *4DOS.INI* file.

Invalid key name: You tried to make an invalid key substitution in *4DOS.INI*, or you used an invalid key name in a keystroke alias or KEYSTACK. Correct the error and retry the operation.

Invalid numeric value: You gave an invalid value for a numeric directive in *4DOS.INI*.

Invalid parameter: 4DOS didn't recognize a parameter. Check the syntax and spelling.

Invalid path: The specified path does not exist. Check the disk specification and/or spelling.

Invalid path specification: You used an invalid path in a path or filename directive in *4DOS.INI*.

Invalid section name: You used an invalid section name in *4DOS.INI*. The only valid section names are [Primary] and [Secondary].

Invalid startup switch, ignored: You passed 4DOS an invalid option on the SHELL= line in *CONFIG.SYS* or on the startup command line for a secondary shell. Correct the switch.

Invalid Swapping option or path: The swap type or disk swap path in the *4DOS.INI* Swapping directive is invalid. 4DOS ignores the bad swap type or path and attempts to scan the rest of the Swapping specification for a valid option. Multiple errors in the Swapping directive will cause this message to repeat. Correct *4DOS.INI* and reboot the system for the corrected swap type to take effect.

Invalid time: An invalid time was entered. Check the syntax and reenter.

Keystroke substitution table full: 4DOS ran out of room to store keystroke substitutions entered in *4DOS.INI*. Reduce the number of key substitutions or contact JP Software for assistance.

KSTACK.COM not loaded: You attempted to execute a KEYSTACK command without loading KSTACK.COM. See the KEYSTACK command for more information.

Label not found: A GOTO or GOSUB referred to a non-existent label. Check your batch file.

Memory deallocation error: 4DOS can't deallocate memory while loading. DOS memory allocation has been corrupted. Reboot the system.

Memory destroyed: The DOS memory control blocks have been corrupted. Reboot the system.

Missing ENDTEXT: A TEXT command is missing a matching ENDTEXT. Check the batch file.

Missing GOSUB: 4DOS 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: 4DOS encountered an ENDLOCAL without a matching SETLOCAL.

No aliases defined: You tried to display aliases but no aliases have been defined.

No closing quote: 4DOS couldn't find a second matching quote (` or ") on the command line.

No file handle available: This is an internal 4DOS disk swapping error. Change to another swapping method if possible, and contact JP Software.

No UMBs; loading low: The LOADHIGH (or LH) command can't find any UMBs for your program. The program is loaded into base memory. LH and LOADHIGH only work with MS-DOS 5.0 and above, when the DOS=UMB directive is included in *CONFIG.SYS* and sufficient upper memory space is available for the program.

No upper memory available, master environment will be placed in low memory: You asked 4DOS to load the master environment into an UMB via the UMBEnvironment directive in *4DOS.INI*, but no UMB was available. Check that your XMS driver is properly installed and / or free up some UMB space in use by another program.

No upper memory available, resident portion will remain in low memory: You requested relocation of the 4DOS resident portion to an UMB via the UMBLoad directive in *4DOS.INI*, but no UMB was available. Check that your XMS driver is properly installed and / or free up some UMB space in use by another program.

Non-DOS disk: DOS can't read the disk. Either the disk is bad, or it has been formatted by a different operating system. Reformat it as a DOS disk.

Not an alias: The specified alias is not in the 4DOS alias list.

Not in environment: The specified variable is not in the environment.

Not in swapping mode: You attempted to turn swapping on or off with the SWAPPING command, but 4DOS is loaded in memory-resident mode and swapping is not active.

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 environment space: 4DOS has run out of environment space. Edit the SHELL line in CONFIG.SYS or the Environment directive in *4DOS.INI* to increase the environment size.

Out of memory: DOS or 4DOS had insufficient free memory to execute the last command, or the DOS memory control blocks have been destroyed. If this error occurs in a secondary shell, return to the primary shell before running the command. Otherwise, try to free some memory by removing memory-resident programs. If the error persists, use the MEMORY command to determine the actual memory available. If the base memory (DOS RAM) figures reported by MEMORY are unreasonable, the memory control blocks have probably been destroyed and you must reboot the system. If you receive this error from DIR when the MEMORY command shows sufficient memory for the directory you are displaying, memory has probably been "fragmented", and contains a free area larger than 8K but not large enough for the entire directory. Use a memory mapping program like PMAP, MAPMEM, or MANIFEST to determine where the fragmentation is, and experiment with your TSRs and applications to determine and remove its cause.

Out of paper: DOS detected an out-of-paper condition on one of the parallel printers (LPT1, LPT2, or LPT3). Check your printer and add paper if necessary.

Overflow: An arithmetic overflow occurred in the %@EVAL variable function. Check the values being passed to %@EVAL. %@EVAL can handle 16 digits to the left of the decimal point and 8 to the right.

Read error: DOS disk read error; usually caused by a bad disk or a non-DOS disk.

Sector not found: BIOS disk error; usually caused by a bad disk or a non-DOS disk.

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

String area overflow: 4DOS ran out of room to store the text from string directives in *4DOS.INI*. Reduce the complexity of *4DOS.INI* or contact JP Software for assistance.

Swap file [seek | read | write] failed: 4DOS encountered an I/O error while accessing the disk swap file (4DOSSWAP.nnn). The disk was changed, the file has been destroyed by a program, or 4DOS's memory area has been overwritten by a program. Reboot the system.

Syntax error: A command or variable function was entered in an improper format. Check the syntax in this manual and correct the error.

Too many open files: DOS has run out of file handles. Try setting FILES=20 or more in your *CONFIG.SYS* file.

Transient memory allocation error: 4DOS couldn't reserve memory for its transient portion (probably in a SWAPPING OFF command). The memory control blocks have been destroyed, or a program has fragmented memory. Reboot the system.

Transient memory deallocation error: 4DOS couldn't release memory for its transient portion (probably in a SWAPPING ON command). The memory control blocks have been destroyed, or a program has fragmented memory. Reboot the system.

Unknown command: A command was entered that 4DOS didn't recognize and couldn't find in the current search path. Check the spelling or PATH specification.

Variable loop: A nested environment variable refers to itself, or variables are nested more than 16 deep. Correct the error and retry the command.

Write error: A DOS disk error, usually caused by a bad disk or a non-DOS disk.

XMS deallocation failed: 4DOS could not deallocate XMS memory when exiting a secondary shell. XMS memory has been destroyed; reboot your system.

XMS move failed: 4DOS could not move data between base memory and XMS memory while swapping itself. XMS memory has been destroyed; reboot your system.

APPENDIX B / ASCII AND KEY CODES

ASCII

To represent the text you type, computers must translate each letter to and from a number. The code used by all PC-compatible computers for this translation is called ASCII (American Standard Code for Information Interchange). ASCII codes are also used for the characters displayed on the screen. Function keys, cursor keys, and **Alt** keys do not generate ASCII codes. For details on these keys and a reference table, see **Keys and Key Codes** later in this Appendix.

The ASCII table on the following pages is in three parts. The first two parts cover the 128 standard ASCII characters; the third part covers the additional 128 extended ASCII characters defined by IBM for use on the IBM PC and compatible computers. All the tables include a **Char** column showing the visual representation of the character, a **Dec** column showing the decimal numeric value of the character in the ASCII set, and a **Hex** column showing the hexadecimal (base-16) value. The tables are divided as follows:

ASCII Control Characters have numeric values between 0 and 31, and include non-printing characters like carriage return and line feed. The ASCII standard does not define a visual representation for control characters, but the IBM PC character set does define one for most control characters as shown in the **Char** column of the table. You can enter these characters using the Ctrl key combination shown in the **Ctrl** column of this table, with a caret **[^]** representing the Ctrl key. For example, character 4 is shown as ^D; to enter it, press Ctrl and D on your keyboard. You can also enter control characters with the **Alt** key and the numeric keypad, like extended ASCII codes (see below). The **Name** column of this table shows a two or three character "name" given to each control character as part of the ASCII standard.

ASCII Printing Characters have numeric values between 32 and 127, and include the entire English-language character set as well as punctuation and other special marks. You enter these characters by pressing the corresponding keyboard key. Character 127 has no corresponding keyboard key, but can be entered on many systems by typing **Ctrl-Backspace**.

Extended ASCII Characters have values between 128 and 255 and include international language characters, line-drawing characters, and other graphics symbols. You can enter extended ASCII codes on the keyboard by holding down the **Alt** key, entering the decimal numeric value of the key on the numeric keypad, and then releasing the Alt key.

ASCII Control Characters

Char	Dec	Hex	Ctrl	Name	Char	Dec	Hex	Ctrl	Name
	000	00	^@	NUL	►	016	10	^P	DLE
☺	001	01	^A	SOH	◄	017	11	^Q	DC1
●	002	02	^B	STX	↕	018	12	^R	DC2
!	003	03	^C	ETX	‼	019	13	^S	DC3
♦	004	04	^D	EOT	¶	020	14	^T	DC4
♣	005	05	^E	ENQ	§	021	15	^U	NAK
♠	006	06	^F	ACK	▬	022	16	^V	SYN
•	007	07	^G	BEL	↨	023	17	^W	ETB
◘	008	08	^H	BS	↑	024	18	^X	CAN
○	009	09	^I	HT	↓	025	19	^Y	EM
◙	010	0A	^J	LF	→	026	1A	^Z	SUB
♂	011	0B	^K	VT	←	027	1B	^[	ESC
♀	012	0C	^L	FF	∟	028	1C	^\	FS
♪	013	0D	^M	CR	↔	029	1D	^]	GS
♫	014	0E	^N	SO	▲	030	1E	^^	RS
☼	015	0F	^O	SI	▼	031	1F	^_	US

ASCII Printing Characters

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
	032	20	8	056	38	P	080	50	h	104	68
!	033	21	9	057	39	Q	081	51	i	105	69
"	034	22	:	058	3A	R	082	52	j	106	6A
#	035	23	;	059	3B	S	083	53	k	107	6B
$	036	24	<	060	3C	T	084	54	l	108	6C
%	037	25	=	061	3D	U	085	55	m	109	6D
&	038	26	>	062	3E	V	086	56	n	110	6E
'	039	27	?	063	3F	W	087	57	o	111	6F
(	040	28	@	064	40	X	088	58	p	112	70
)	041	29	A	065	41	Y	089	59	q	113	71
*	042	2A	B	066	42	Z	090	5A	r	114	72
+	043	2B	C	067	43	[	091	5B	s	115	73
,	044	2C	D	068	44	\	092	5C	t	116	74
-	045	2D	E	069	45	]	093	5D	u	117	75
.	046	2E	F	070	46	^	094	5E	v	118	76
/	047	2F	G	071	47	_	095	5F	w	119	77
0	048	30	H	072	48	`	096	60	x	120	78
1	049	31	I	073	49	a	097	61	y	121	79
2	050	32	J	074	4A	b	098	62	z	122	7A
3	051	33	K	075	4B	c	099	63	{	123	7B
4	052	34	L	076	4C	d	100	64	¦	124	7C
5	053	35	M	077	4D	e	101	65	}	125	7D
6	054	36	N	078	4E	f	102	66	~	126	7E
7	055	37	O	079	4F	g	103	67		127	7F

Extended ASCII Characters

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
Ç	128	80	á	160	A0	└	192	C0	α	224	E0
ü	129	81	í	161	A1	┴	193	C1	ß	225	E1
é	130	82	ó	162	A2	┬	194	C2	Γ	226	E2
â	131	83	ú	163	A3	├	195	C3	π	227	E3
ä	132	84	ñ	164	A4	─	196	C4	Σ	228	E4
à	133	85	Ñ	165	A5	┼	197	C5	σ	229	E5
å	134	86	ª	166	A6	╞	198	C6	µ	230	E6
ç	135	87	º	167	A7	╟	199	C7	τ	231	E7
ê	136	88	¿	168	A8	╚	200	C8	Φ	232	E8
ë	137	89	⌐	169	A9	╔	201	C9	Θ	233	E9
è	138	8A	¬	170	AA	╩	202	CA	Ω	234	EA
ï	139	8B	½	171	AB	╦	203	CB	δ	235	EB
î	140	8C	¼	172	AC	╠	204	CC	∞	236	EC
ì	141	8D	¡	173	AD	═	205	CD	φ	237	ED
Ä	142	8E	«	174	AE	╬	206	CE	ε	238	EE
Å	143	8F	»	175	AF	╧	207	CF	∩	239	EF
É	144	90	▒	176	B0	╨	208	D0	≡	240	F0
æ	145	91	▒	177	B1	╤	209	D1	±	241	F1
Æ	146	92	▒	178	B2	╥	210	D2	≥	242	F2
ô	147	93	│	179	B3	╙	211	D3	≤	243	F3
ö	148	94	┤	180	B4	╘	212	D4	⌠	244	F4
ò	149	95	╡	181	B5	╒	213	D5	⌡	245	F5
û	150	96	╢	182	B6	╓	214	D6	÷	246	F6
ù	151	97	╖	183	B7	╫	215	D7	≈	247	F7
ÿ	152	98	╕	184	B8	╪	216	D8	°	248	F8
Ö	153	99	╣	185	B9	┘	217	D9	∙	249	F9
Ü	154	9A	║	186	BA	┌	218	DA	·	250	FA
¢	155	9B	╗	187	BB	▉	219	DB	√	251	FB
£	156	9C	╝	188	BC	▐	220	DC	n	252	FC
¥	157	9D	╜	189	BD	▌	221	DD	2	253	FD
₧	158	9E	╛	190	BE	▀	222	DE	■	254	FE
ƒ	159	9F	┐	191	BF		223	DF		255	FF

Keys and Key Codes

When you press a single key or a key combination, software built into your computer (the **BIOS** or **B**asic **I**nput / **O**utput **S**ystem) 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. The BIOS returns these numbers the next time a program requests keyboard input. This section explains how key codes work; for information on using them with 4DOS features see *4DOS.INI* (page 128), keystroke aliases (page 167), INKEY (page 237), and KEYSTACK (page 240).

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 and the computer manufacturer who provides the built-in BIOS. As a result, we can't guarantee the accuracy of the information below 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 AT-compatible and PS/2 computers, but virtually all of it is applicable to the 84-key keyboards on older PC and XT systems. The primary difference is that the older keyboards lack a separate cursor pad and only have 10 function keys.

All keys have a scan code. Not all keys have an ASCII code. For example, function keys and cursor keys are not part of the ASCII character set (see above), and have no ASCII value, but they do have a scan code. Some keys have more than one ASCII code. The **A** key, for example, has ASCII code 97 (lower case "a") if you press it by itself. If you press it along with the **Shift** key, 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 the BIOS 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 BIOS 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 that we're discussing, 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 (the spacebar) to four, depending on the particular key, the design of your keyboard, and the BIOS in your system. Some keys, like the **Alt**, **Ctrl**, and **Shift** by themselves or in combination with each other, plus the **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 BIOS 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.

The following table lists all of the keys on the 101-key "enhanced" keyboard. The keys are arranged roughly in scan code order, which is generally left to right, moving from the top of the keyboard to the bottom.

Column 1 shows the key's keycap symbol or name. Columns 2 and 3 show the scan and ASCII code if the key is unshifted. Columns 4 & 5 contain the codes for the shifted key. Columns 6 & 7 show the code for Ctrl plus the key. The last column contains the scan code for Alt plus the key (Alt keystrokes have no ASCII code and always generate an ASCII code of 0, which is not shown).

Keys with ASCII values return the same scan code regardless of whether **Shift**, **Ctrl**, or **Alt** is pressed, so the scan code columns below may seem repetitive for these keys. We've included them in the interest of completeness, since you may need to verify the exact code for a particular key combination.

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 are not reported at all by the BIOS, like Ctrl-1 and Alt-PgUp.

Scan Codes and Key Codes for Top Two Keyboard Rows

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
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

Scan Codes and Key Codes for Bottom Two Keyboard Rows

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
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

Scan Codes and Key Codes for Key Pads and 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
np *	55	42	55	42	150	0	55
np Home	71	0	71	55	119	0	
cp Home	71	224	71	224	119	224	151
np Up	72	0	72	56	141	0	
cp Up	72	224	72	224	141	224	152
np PgUp	73	0	73	57	132	0	
cp PgUp	73	224	73	224	132	224	153
np Minus	74	45	74	45	142	0	74
np Left	75	0	75	52	115	0	
cp Left	75	224	75	224	115	224	155
np 5	76	0	76	53	143	0	
np Right	77	0	77	54	116	0	
cp Right	77	224	77	224	116	224	157
np Plus	78	43	78	43	144	0	78
np End	79	0	79	49	117	0	
cp End	79	224	79	224	117	224	159
np Down	80	0	80	50	145	0	
cp Down	80	224	80	224	145	224	160
np PgDn	81	0	81	51	118	0	
cp PgDn	81	224	81	224	118	224	161
np Ins	82	0	82	48	146	0	
cp Ins	82	224	82	224	146	224	162
np Del	83	0	83	46	147	0	
cp Del	83	224	83	224	147	224	163
np /	224	47	224	47	149	0	164
np Enter	224	13	224	13	224	10	166

APPENDIX C / TECHNICAL INFORMATION

This appendix provides technical information for programmers who wish to build interfaces to 4DOS. It covers detecting 4DOS, placing keystrokes in the Keystack, writing installable commands, and using the *DESCRIPT.ION* file.

Detecting 4DOS

Detecting 4DOS From a Batch File

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

```
if not "%@eval[2+2]" == "4" echo 4DOS is loaded!
```

This test can never succeed in *COMMAND.COM* and is therefore a reliable way to detect 4DOS. Other variable functions could be used for the same purpose.

Detecting 4DOS From a Program

Any program can test for the presence of 4DOS by making a simple INT 2Fh call. Be sure to check the INT 2Fh vector first as it may be 0 under some versions of DOS 2 if no program has hooked the interrupt. To detect 4DOS, call INT 2Fh with:

AX = D44Dh
BX = 0

If 4DOS is not loaded, AX should be returned unchanged. If 4DOS is loaded, it will return the following (no other registers are modified):

AX = 44DDh
BX = Version number (BL = major version, BH = minor version)
CX = 4DOS PSP segment address
DL = 4DOS shell number

The shell number is incremented each time a new copy of 4DOS is loaded, either in a different multitasker window (for example, under DESQView), or via nested shells. The primary shell is shell number 0.

This function tells you if 4DOS is loaded in memory, but not whether it is the parent process of your program. You can determine if 4DOS is the parent

process by comparing the PSP value returned in CX to the PSP chain pointer at offset 16h in your program's PSP.

Detecting the 4DOS Prompt

4DOS generates INT 2Fh calls before and after the prompt is displayed to allow TSRs to detect that 4DOS is at the prompt. The calls are:

AX = D44Eh
BX = 0: 4DOS is about to display the prompt
1: 4DOS is about to accept keyboard input at the prompt

The BX = 0 call occurs immediately before displaying the prompt; the BX = 1 call occurs after displaying the prompt and immediately before accepting keyboard input. Any routine intercepting these calls should preserve the SI, DI, BP, SP, DS, ES, and SS registers.

Placing Keystrokes Into the Keystack

You can put keystrokes into the 4DOS Keystack with an INT 2Fh call. First, you must make a call to check whether KSTACK.COM is loaded:

AX = D44Fh
BX = 0

If *KSTACK.COM* is not loaded, this call will return AX unchanged. If it is loaded, AX will be returned as 44DDh; other registers will be unchanged. Once you have determined that *KSTACK.COM* is loaded, you can send keystrokes with this call:

AX = D44Fh
BX = 1
CX = number of keystrokes being passed
DS:DX = address of the keystroke array

On return, if the call succeeded then AX will be 0; if it failed, AX will be non-zero. BX, CX, and DX are destroyed; other registers are preserved. If the call succeeds, subsequent calls to INT 16h functions 0, 1, 10h, or 11h will receive the stacked keystrokes.

The keystroke array passed to KSTACK must be an array of words containing the values to return from INT 16h. The high byte of each word is a scan code and the low byte is an ASCII code. Many programs accept keystrokes properly with only the ASCII code, but some require the scan code as well. See page 314 for a list of ASCII and scan codes for most keyboards. To insert a delay in the

keystroke sequence, include a word set to FFFFh followed by a word containing the desired delay in clock ticks.

Writing Installable Commands

An "installable command" is created with a memory-resident program (TSR) which can receive signals from 4DOS and process commands. 4DOS makes every command available to such TSRs before it is executed; if any TSR chooses to execute the command, 4DOS will do no further processing. Otherwise, 4DOS processes the command normally.

The 4DOS "Installable command" interface is compatible with an undocumented interface present in *COMMAND.COM* for MS-DOS and PC-DOS 3.3 and above. This interface is documented more thoroughly on pages 373 - 379 of the excellent reference text *Undocumented DOS* by Schulman et. al., published by Addison Wesley (ISBN 0-201-57064-5).

4DOS looks for an installable command after alias expansion and after checking to see if the command is a drive change, but before checking for an internal or external command.

4DOS first makes an INT 2Fh call to determine whether any TSR loaded will respond to the command, with:

AX = AE00h
BX = offset of command line buffer:
first byte = maximum length of command line (128)
second byte = actual length of command line, not including trailing CR
remainder = command line, with a trailing CR
CH = FFh
CL = length of command line, not including the command name
DX = FFFFh
SI = offset of command name buffer:
first byte = length of command name
remainder = command name, shifted to upper case and padded with blanks to 11 characters
DI = 0

If the TSR does not recognize the command as its own, it must pass the INT 2Fh along with registers unchanged. If it does recognize the command, it must return 0FFh in AL. The command should not be executed at this point. 4DOS will then make another call (buffer formats are the same as above):

AX = AE01h

BX = offset of command line buffer
CH = 0
CL = length of command name
DX = FFFFh
SI = offset of command name buffer

If the TSR executed the command line, it must set the command name length (DS:[SI]) to 0. If the command line length is not 0, 4DOS will attempt to execute the command as an internal or external command. The installable command should return the result of the command (zero for normal return or non-zero for an error) in AL.

Using *DESCRIPT.ION*

4DOS uses the file *DESCRIPT.ION* to store file descriptions. This file is created as a hidden file in each subdirectory which has descriptions, and deleted when all descriptions are removed or when all files with descriptions are deleted.

Your programs can access *DESCRIPT.ION* to create, retrieve, or modify file descriptions, and to store other information. *DESCRIPT.ION* has one line per file, and is unsorted. Each line is in the following format:

```
filename.ext Description[◆<ID>Other program info]...<CR>
```

There is normally one space between the description and filename but additional spaces may be used in future versions of 4DOS. The characters after the description allow extension of the description format for use by other programs. They are as follows:

◆ is an ASCII Ctrl-D (04), and marks the end of the description text and the beginning of information for a program other than 4DOS. This symbol can appear multiple times on each line; each occurrence marks the beginning of information for another program.

<ID> is an identification byte for the program which is using this area of the particular line. If you are writing a program which will store information in *DESCRIPT.ION*, test it using an ID byte of your own choosing. When you are ready to release the program, contact JP Software and we will provide you with an ID byte value that is not in use by others to the best of our knowledge.

Other program info is any text the program wishes to store in its area of the line. The text should relate specifically to the file named on the

line. It may not contain the ◆ character, carriage returns, line feeds, or nulls (ASCII 0s).

4DOS will copy, delete, or move all the information on a line in *DESCRIPT.ION*, including information owned by other programs, when performing the same action on the corresponding file. 4DOS will also change the name if a file is renamed. To support *DESCRIPT.ION* properly, your program must do the same if it copies, deletes, moves, or renames files. Take care not to remove information which does not belong to your program, or delete lines which contain information for other programs. Your program should be able to handle a line terminated by a CR or LF alone, a CR/LF pair, an EOF (ASCII 26), or the physical end of the file. The lines it creates should be terminated with CR / LF. The line length limit is 4096 bytes; exceeding this limit will cause unpredictable results.

GLOSSARY

Some items in this glossary refer to 4DOS features. For more information on how a particular feature works, look the feature up in the index under the name used here.

4DOS.INI: The 4DOS initialization file containing directives which set 4DOS's startup configuration parameters.

4EXIT: A batch file which is executed whenever a secondary 4DOS shell ends.

4START: A batch file which is executed whenever 4DOS is started, either as a primary shell or a secondary shell.

Alias: A shorthand name for a command or series of commands.

Alias Argument: A numeric variable included in an alias definition, allowing a different value to be used in the alias each time it is executed.

AND: A logical combination of two true or false conditions such that if both conditions are true the result is true; if either condition is false the result is false.

ANSI.SYS: A device driver supplied with DOS which provides enhanced screen display and keyboard macros, or one of the many similar programs.

Append: Concatenation of one file or string onto the end of another. (There is also a DOS external command, APPEND, which has an entirely different meaning and which should be used with caution!)

Application: A program run from the command prompt or a batch file. The term is 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 (presuming that the last backup cleared the archive attribute, as most backup programs do). Also sometimes used to refer to a single file which contains a number of other files in compressed form.

Argument: A piece of additional information placed after a command or function name. For example in the command DIR XYZ, XYZ is an argument. Also used to refer to an alias argument or batch file argument.

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.

ASCII File: A file containing ASCII text, as opposed to a binary file which may contain codes, numbers, or other information that cannot be sensibly interpreted as text.

Attribute: A characteristic of a file which can be set or cleared. The standard attributes are Read-Only, Hidden, System, and Archive. The special attributes Volume Label and Directory are also used for files of those types.

AUTOEXEC.BAT: A batch file which is AUTOmatically EXECuted when the computer is started.

Automatic Directory Change: A 4DOS feature which allows changing directories by typing the directory name and a backslash [\] at the prompt.

Base Memory: The portion of your computer's memory available for use by DOS, the command processor, and application programs. On most PCs this area consists of the first 640K bytes of the computer's memory (one K is 1024 bytes).

BAT File: Same as a Batch File.

Batch File: A text file containing a sequence of DOS or 4DOS commands. Batch files are used to save command sequences so that they can be re-executed at any time, transferred to another system, etc.

Batch File Argument: A numeric variable used within a batch file, allowing a different value to be used at that spot in the file each time it is executed.

Binary File: A file containing information which does not represent or cannot sensibly be interpreted as text. See also **ASCII File**.

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, and many networks.

Boot: The process of starting the computer and loading DOS and the command processor into memory.

Boot Directory: The current directory at the time the system is booted, almost always 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).

Break: A signal sent to a program (including 4DOS) 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 batch file which is loaded into memory, dramatically speeding up execution.

Buffer: An area of memory set aside for storage. Usually refers to disk buffers, used to save information as it is transferred between your program and the disk; or to the keyboard buffer, which holds keystrokes until a program can use them.

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.

Code Page: A set of definitions which tells DOS how to get and display date, time, and other information in the format appropriate to a particular country.

Command Completion: A 4DOS 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 History: A 4DOS feature which retains commands you have previously executed, so that they can be modified and re-executed later.

Command Interpreter: A program, such as 4DOS or *COMMAND.COM*, which interprets commands and executes other programs.

Command Recall: See **Command History**.

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**.

COMSPEC: An environment variable which defines where to get the command processor for a secondary shell.

Conditional Commands: A 4DOS feature allowing commands to be executed or skipped depending on the results of a previous command. See also **Exit Code**.

CONFIG.SYS: A file which tells DOS what device drivers to install, what command processor to use, and other information about what to do when your system boots. See your DOS manual for more information.

Console: The PC keyboard and display.

Control Character: A character which is part of the ASCII code, but does not have a normal text representation, and which can be generated by pressing the Ctrl key along with another key. For example the control character called "BEL" is generated by pressing Ctrl-G, and will cause the computer to beep if pressed at the DOS prompt.

Country Code: See **Code Page**.

CR: The ASCII character "carriage return", 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 input, output, or network access, which prevents a program from continuing. When a critical error occurs, a prompt such as "R(etry), I(gnore), F(ail), or A(bort)" appears on the screen and you must decide what action to take.

Current Directory: The directory in which all file operations will take place unless otherwise specified. For example if the current directory is C:\MYFILES then the command "DEL XYZ" will delete the file XYZ in the directory C:\MYFILES, and will not affect other files with the same name which may be in other directories.

Description: A string of characters assigned to describe a file with the 4DOS DESCRIBE command.

Destination: In the 4DOS file commands (COPY, MOVE, and RENAME), the name or directory files should have after any copying or modification has taken place; generally referred to by the last specification on the command line. See also **Source**.

Device: A physical device for input or output such as the console, a communications port, or a printer. Sometimes the term "device" alone is used to refer to character devices such as those listed above, and excludes block devices.

Device Driver: A program which allows DOS to communicate with a device, and which is loaded into memory when the system is booted, via a statement in CONFIG.SYS. Some device drivers are used to manage memory or for other similar internal functions, rather than to communicate with an external device.

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.

Disk Swapping: A type of swapping in which the transient portion of 4DOS is stored on disk while an application is running.

DOS Memory: See Base Memory.

Echo: See Command Echoing.

EMS Memory: Memory which conforms to the Lotus - Intel - Microsoft Expanded Memory Specification (LIM EMS). This hardware/software standard allows programs to access large amounts of memory outside of base memory or extended memory. Most systems which have EMS memory have either a specific EMS board (on any PC or compatible system), EMS emulation software (on PC-AT compatible systems), or a 386 control program such as QEMM or 386MAX (on 386 systems). If you do not have one of these items, you probably do not have EMS memory.

EMS Swapping: A type of swapping in which the transient portion of 4DOS is stored in EMS memory while an application is running.

Environment: An area of memory which contains multiple entries in the form "NAME=value". Each entry is called an environment variable. See also **Master Environment** and **Passed Environment**.

Environment Variable: The name of a single entry in the environment. Environment variables are entered with the SET command, and used to associate the specified value with the specified name for any purpose desired by the user. Typically, their function is to set default switches for a program, specify a directory path where files can be found, and so on. When 4DOS encounters an environment variable name in a command in the form "%NAME" (or "%NAME%"), it substitutes the corresponding value in its place.

Errorlevel: A numeric value between 0 and 255 returned from an external command to indicate its result (*e.g.*, success, failure, response to a question), and accessible via the IF ERRORLEVEL command and the %? environment variable. See also **Exit Code**.

Escape Character: This term has two meanings. In some contexts it means the 4DOS escape character, normally Ctrl-X, which is used to suppress the normal meaning of or give special meaning to the following character. In other

cases it refers to the specific ASCII character ESC. The meaning must be determined from the context.

Executable Extensions: A 4DOS 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 with the extension .COM or .EXE, which can be loaded into memory and run as a program.

Exit Code: The error level returned by an external command, or a similar exit code returned by a 4DOS internal command. 4DOS internal commands return an exit code of 0 if successful, or non-zero if unsuccessful. See also **Errorlevel**.

Expansion: The process 4DOS goes through when it scans a command line and substitutes the appropriate actual values for aliases, alias arguments, batch file arguments, 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 Memory: Any memory on a 286-based (PC-AT compatible) or 386-based computer system which is above the 1 MB (one megabyte, or 1024*1024 bytes) of memory that DOS can address directly. Any PC-AT compatible or 386 system with more than 640K of memory has extended memory. This memory can be used for a variety of purposes depending on the software installed to utilize it. See also **XMS**.

External Command: A program which resides in an executable file, as opposed to an internal command which is part of the command processor. For example, FORMAT and DISKCOPY are external commands; you can find them on your DOS disk under the names *FORMAT.COM* and *DISKCOPY.COM*.

File Attribute: See **Attribute**.

File Description: See **Description**.

Filename Completion: A 4DOS feature which allows you to type part of a filename on the command line, and have 4DOS fill in the rest for you.

Free Memory: Usually, the amount of base memory which is currently unoccupied and available for use by applications.

Hidden: A file attribute indicating that the file should not be displayed with a normal DIR command, and should not be available for access by programs unless they specifically request use of hidden files.

History: See **Command History**.

History Window: A pop-up window used by 4DOS to display the command history, allowing you to choose a previous command to modify and / or execute.

Include List: A concise method of specifying several files or groups of files in the same directory, for use with all 4DOS commands which take file names as arguments.

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. For example, DIR and COPY are internal commands.

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.

Label: A marker in a batch file. Labels allow GOTO and GOSUB commands to "jump" to the command on the line following the label. See also **Volume Label**.

LF: The ASCII character "line feed", not usually generated from the keyboard, but stored in most ASCII files at the end of each line after the CR character.

Master Environment: The master copy of the environment maintained by the command processor. The master environment is manipulated with the SET, ESET, and UNSET commands.

Memory Resident Mode: A method of installing 4DOS in which swapping is disabled, and all of 4DOS remains permanently resident in memory. Memory-resident mode requires much more memory than the normal swapping mode.

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 4DOS feature which allows multiple commands to be placed on a line, separated by a caret (^) or other 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 programs like DESQview.

Non-Swapping Mode: See **Memory Resident Mode**.

Option: See **Switch**.

OR: A logical combination of two true or false conditions such that 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**.

Parameter: See **Argument**.

Parsing: The process 4DOS performs when it analyzes the command line, performs alias and environment variable expansion, and finds the appropriate internal command or external command to execute.

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. The size of the passed environment is determined by the amount of space actually used in the master environment when the application is started.

Path: A specification of the directories a file resides in, including all parent directories. For example, the path for C:\WPFILES\MYDIR\MEMO.TXT is C:\WPFILES\MYDIR\. **Path** is 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**.

Primary Shell: The copy of the command processor which is loaded when the system boots.

RAM Disk: A pseudo "disk drive", created by software, which appears like a normal physical disk drive to programs. RAM disks can be stored in extended

memory, XMS memory, EMS memory, or base memory, and provide a convenient way for programs to use this memory for temporary files.

Read-Only: A file attribute indicating that the file can be read, but not written by DOS and 4DOS.

Reboot: The process of restarting the computer, usually by pressing the Ctrl, Alt, and Del keys simultaneously.

Redirection: A method for collecting the standard output of a program in a file, and/or of providing the standard input for a program from a file, signified respectively by a greater than symbol ">" or less than symbol "<" on the command line. See also **Pipe**.

Replaceable Parameter: See **Alias Argument** and **Batch File Argument**.

Resident Portion: The small portion of 4DOS stored permanently in memory when swapping mode is in use, as opposed to the larger transient portion. The resident portion is stored as close as possible to the beginning of base memory.

Root Directory: The first directory, from which all other directories are "descended." The root directory is referenced with a single backslash (\). When a disk is first formatted, it is the only directory.

Secondary Shell: Any copy of any command processor which is loaded after the system boots. Secondary shells may be started from the command line; by executing a "shell to DOS" operation from within an application; implicitly by executing some DOS-related functions within certain applications; or by starting a new partition in a multi-tasking or task switching system.

Shell: A command processor. Also used to refer to a program which gives access to DOS functions and commands through a menu- or mouse-driven system.

Source: In the 4DOS file commands (COPY, MOVE, and RENAME), the original files before any copying or modification has taken place, i.e. those specified earlier on the command line. See also **Target**.

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: A file or character device where error messages from a program are displayed. Standard error output always goes to the console, unless redirection is used.

Standard Input: A file or character device where a program obtains its normal input. Standard input always comes from the console, unless redirection is used.

Standard Output: A file or character device where normal output from a program is displayed. Standard output always goes to the console, unless redirection is used.

Subdirectory: Any directory other than the root directory. See also **Root Directory**.

Swap File: A disk file created by 4DOS to store its transient portion when disk swapping is in use.

Swapping: A 4DOS feature which removes the larger transient portion of 4DOS from base memory while an application is running, leaving the maximum possible amount of memory for the application, and allows 4DOS to restore information such as the command history after the application is finished. See also **XMS Swapping**, **EMS Swapping**, and **Disk Swapping**.

Switch: An argument to 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". Also, a configuration value for 4DOS specified on the SHELL= line in CONFIG.SYS.

System: A file attribute indicating that the file belongs to DOS and should not be accessed by normal programs.

Target: See **Destination**.

Transient Portion: The larger portion of 4DOS stored in memory temporarily when swapping mode is in use, as opposed to the smaller resident portion. The transient portion is stored as close as possible to the end of base memory, and releases the space it is using whenever an application is executed.

TSR: A **T**erminate and **S**tay **R**esident program, i.e. a program which "terminates" but remains resident in base memory, to provide capabilities such as network support, a pop-up notepad or telephone dialer, a disk cache, or mouse support.

UMB: An XMS Upper Memory Block, whose address is above the end of normal base memory, but is within the 1 megabyte of memory that DOS can address directly.

Variable: See **Alias Argument, Batch File Argument,** and **Environment Variable**.

Volume Label: A special, hidden file placed on any disk whose name constitutes a "label" for the entire disk.

White Space Character: Generally, a character used to separate arguments on the command line. The white space characters recognized by 4DOS are the space, tab, comma, semicolon, and equal sign.

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.

XMS Memory: A software method for accessing extended memory on 286 and 386 systems (and some systems which support version 4.0 of the EMS memory specification). XMS memory is not additional memory, but is a software specification only. Its use generally requires Microsoft's HIMEM.SYS be installed as a device driver, or that another program such as 386MAX or MOVE-EM be used.

XMS Swapping: A type of swapping where the transient portion of 4DOS is stored in XMS memory while an application is running.

XOR (exclusive OR): A logical combination of two true or false conditions such that 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.

Index

Conventions: Most fully capitalized terms (*e.g.* ECHO, SELECT) are 4DOS command names unless otherwise noted. When appropriate, a **bold** page number is used to indicate the primary page for information on terms with multiple references.

Special Characters

4

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

How *4DOS* Works

4DOS replaces the standard DOS command processor (COMMAND.COM), and gives you a complete, compatible new interface, right at the familiar "C>" prompt. All the commands you're used to work just as you expect—and most have significant enhancements. Then *4DOS* adds over 40 new commands that make the command line a friendly and powerful place to work. Because *4DOS* replaces only COMMAND.COM, the rest of DOS remains unchanged, and your other software works just as it always has. All of *4DOS*'s capabilties are built in, replacing dozens of external utilities and memory-hungry TSRs—in fact, *4DOS* uses less of your sytem's base memory than COMMAND.COM!

A Friendly Command Line

With *4DOS* you can edit DOS commands and correct typing errors with standard editing keys. The command history lets you recall previous commands one at a time or in a pop-up window, edit them, and re-execute them—all at the touch of a few keys. You can even assign common commands to a single keystroke. And if you can't remember how a command works, just press F1 at the prompt to call up complete, cross-referenced help, with examples.

Batch File Power

If you write batch files, *4DOS* will make your work easier in dozens of ways. Accept user input, construct screen displays with ease, perform logic with block-structured IF statements—in batch files that run 2 to 10 times faster! Over 80 built-in variables and functions give you easy access to everything from date and time to the individual parts of a file name, and let you evaluate complex arithmetic expressions with ease. Convert small batch files to aliases, and you'll improve performance—and save disk space too.

JP software
P.O. Box 1470, East Arlington, MA 02174, USA
Tel. 617.646.3975 Fax 617.646.0904

What the Reviewers Say

"The best thing that's happened to DOS since the subdirectory"
John Wolfskill, *PC Resource*

"Lets you do everything DOS should let you do but doesn't, and gives you up to 4K extra RAM as a bonus"
Edward Mendelson, *PC Magazine*

"Packs the brawny commands that DOS forgot...makes entering DOS commands a breeeze"
Chris DeVoney, *PC/Computing*

"Superior replacement for COMMAND.COM...a sophisticated command processor and batch language"
John Udell, *Byte*

1989
- FINALIST -
PC Magazine Award
for Technical Excellence

UTILITY SOFTWARE
***4DOS*, Version 2.21**
JP Software

System Requirements & Compatibility

IBM PC, XT, AT, 386, 486, PS/2, or compatible, any PC video board and monitor. Two floppy disks minimum; hard disk recommended.

Minimum 256K Installed RAM; uses 3K or less while applications are running. Supports expanded (EMS) and extended (XMS) memory.

MS-DOS and PC-DOS: version 2.0 or above, 3.0 or above recommended, fully supports DOS 5.0. **DR-DOS:** version 3.4, 5.0, and 6.0. **OS/2:** DOS box in all versions.

Compatible with all standard applications, including memory-resident programs (TSRs); multitaskers like Windows and DESQview; all DOS-compatible networks including Novell, 3COM, Banyan, and LANtastic.