Personal Computer
Computer Language
Series

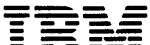
Disk Operating System

by Microsoft, Inc.

6936752

IMPORTANT INFORMATION

If you have IBM applications which ran under DOS Versions 1.00 or 1.10, please refer to Appendix K for information about using those applications under DOS Version 2.00.



Personal Computer
Computer Language
Series

Disk Operating System

by Microsoft, Inc.

First Edition (January 1983)
Version 2.00

Products are not stocked at the address below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM Personal Computer Dealer.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication.

A Reader's Comment Form is provided at the back of this publication. If this form has been removed, address comments to: IBM Corp., Personal Computer, P.O. Box 1328-C, Boca Raton, Florida 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligations whatever.

Preface

Read This First

This reference manual comes in two sections. Sections 1 and 2 explain how to use the IBM Personal Computer Disk Operating System Version 2.00 (DOS). Section 1 *DOS Guide* covers those topics that apply to all DOS users that are using developed application programs. Section 2 *Advanced DOS Features and Technical Reference* covers those topics that apply to more experienced DOS users, system programmers, or those who will be developing their own applications. These two sections provide information on how to issue commands to DOS to create, edit, link, debug, and execute programs. They also explain how to prepare DOS and your fixed disk for use, and how to use the new features of DOS Version 2.00.

First Time Users

Before using your DOS diskette for the first time read the sections "About Diskettes" and "Backing Up Your DOS Diskettes" in Chapter 1.

Experienced DOS Users

Before you use DOS Version 2.00 for the first time, please read "DOS Version 2.00 Enhancements" in Chapter 1 and "Appendix A" in Section 2.

About Your DOS Diskettes

DOS Version 2.00 is shipped on two diskettes. The first labeled "DOS" contains the DOS programs and commands. It is referred to as the DOS diskette in this manual. The second diskette labeled "DOS Supplemental Programs" contains the LINK Utility, EXE2BIN, DEBUG, and several BASIC sample programs (see the BASIC manual for information about the sample programs).

Terms Used

The terms "disk," "diskette," and "fixed disk" are used throughout this manual. Where "diskette" is used, it applies only to diskette drives and diskettes. Where "fixed disk" is used, it applies only to the IBM nonremovable fixed disk drive. Where "disk" is used, it applies to both fixed disks and diskettes.

Considerations for Using Applications

If you have any of the following applications, please refer to Appendix K for additional information about using these applications with DOS Version 2.00:

- Accounting Packages by BPI Systems, Inc.
- Accounting Packages Version 1.00 by Peachtree Software, Inc.
- Accounting Packages Version 1.10 by Peachtree Software, Inc.
- Arithmetic Games 1 and 2
- Asynchronous Communications Support Version 1.00
- Asynchronous Communications Support Version 2.00
- EasyWriter Version 1.10
- Fact Track
- PFS:File
- PFS:Report
- SNA 3270 Emulation and RJE Support Version 1.00
- The Dow Jones Reporter Version 1.00

- Typing Tutor
- VisiCalc Version 1.10
- 3101 Emulator Version 1.00

Organization of This Manual

This manual has 14 chapters and 11 appendixes.

Section 1. DOS Guide contains:

Chapter 1 contains introductory information about DOS, DOS diskettes, diskette drives, files, and automatic program execution.

Chapter 2 contains information about starting DOS, as well as directions on how to use the control keys and DOS editing keys.

Chapter 3 is an introduction to some of the more commonly used DOS commands.

Chapter 4 contains detailed descriptions of using fixed disk on your system.

Chapter 5 describes how to use tree-structured directories.

Chapter 6 contains detailed descriptions of the commands you can issue to DOS. These commands enable you to manage files, maintain disks, and create and execute programs.

Chapter 7 describes how to use the Line Editor (EDLIN) program to create, alter, and display source language files and text files.

Chapter 8 lists messages generated by the programs described in this manual.

Section 2. Advanced DOS Features and Technical Reference contains:

Chapter 9 describes how to configure your system.

Chapter 10 contains detailed descriptions of the advanced commands you can issue to DOS.

Chapter 11 describes how to use the linker (LINK) program to link programs together before execution.

Chapter 12 describes how the DEBUG program provides a controlled test environment so you can monitor and control the execution of a program to be debugged; by altering, displaying, and executing object files.

Chapter 13 contains detailed information about using extended screen and keyboard functions.

Chapter 14 contains detailed information about Device Drivers.

Appendix A describes DOS enhancements.

Appendix B contains general technical information, and introduces the technical information in Appendixes C-H.

Appendix C describes allocation of space on disk.

Appendix D describes the system interrupts and function calls.

Appendix E describes control blocks and work areas, including a Memory Map, Program Segment, and File Control Block.

Appendix F describes how to execute commands from within an application.

Appendix G contains technical information about DOS support of fixed disks.

Appendix H contains detailed information about .EXE file structure.

Appendix I describes how to run compilers and the macro assembler with a fixed disk.

Appendix J describes how to run the Pascal compiler with a fixed disk.

Appendix K describes how to use applications with DOS Version 2.00.

Contents

Figures

Notes:

Section 1. DOS Guide

Contents

Notes:

Chapter 1. Introduction

Contents

Notes:

What Is DOS

The IBM Personal Computer Disk Operating System (DOS) is a group of programs that you can use to do work with your computer. DOS is a *disk* operating system, meaning that the DOS programs are to be used with diskettes or with fixed disks. The programs that DOS contains are important because they provide a way to organize and use the information you place on disks.

The DOS programs control the way your computer uses other programs, such as application programs. DOS tells your computer how to use or *read* information that you supply to the programs. DOS also tells your computer how to return or *write* information that the programs supply to you.

Why you should use DOS

DOS gives you an easy way to use applications and create and manage files for your applications. DOS also lets you use devices such as printers and disk drives with your computer.

You should use DOS if you are doing any of the following:

- Using applications that need DOS
- Using new disks with applications that use DOS
- Copying disks that have been used with DOS or with DOS applications
- Performing other tasks on disks that have been used with DOS or with DOS applications

Your applications will tell you if you need to use DOS with them. If you are using DOS, this book will help you learn more about the tasks that you can perform with DOS to organize and maintain the information you place on your disks.

DOS Version 2.00 Enhancements

DOS Version 2.00 contains significant functional enhancements and some minor operational and technical differences from previous versions of DOS. We strongly recommend that you take the time to review the information in Appendix A (Section 2), whether you are an experienced DOS user or are about to use DOS for the first time.

Because of the significant amount of function added, DOS Version 2.00 is considerably larger than previous versions. We recommend a minimum memory size of 64K bytes for DOS Version 2.00 (128K bytes if you are using a fixed disk).

Backing Up Your DOS Diskettes

Making a copy of your DOS diskette and your DOS Supplemental Program diskette should be one of the first things you do with your IBM Personal Computer after you get DOS. That way you won't be "shut down" if your DOS diskette becomes misplaced or accidentally damaged. This copy is called a *backup* and making the copy is usually called *backing up*. Refer to "Backing Up a Diskette" in Chapter 3 and follow the procedure to backup your DOS diskette and DOS Supplemental Program diskette. Label and date the backup diskette. Use a felt-tip pen. Store the original DOS diskette properly and use the backup diskette in your daily operations.

About Diskettes

Formatting Your Diskettes

You must format *every* diskette before it can be used by DOS. You do not need to use FORMAT every time you want to put information on a diskette. (Only the first time you use a diskette.)

The DOS FORMAT command writes on every sector of your diskette, sets up the directory and File Allocation Table, and puts the boot record program at the beginning of your diskette.

FORMAT also creates a copy of DOS on a new diskette if you specify it in your command. This way, you can create a diskette containing DOS and have plenty of space for your own data on the same diskette. Keep in mind that only DOS system files are copied when you run FORMAT - none of the other files you may have on your DOS diskette are copied.

For more information about formatting diskettes, refer to "Getting a Diskette Ready to be Used" in Chapter 3. For more information about FORMAT, refer to Chapter 6.

Single- and Dual-Sided Diskettes

Your IBM Personal Computer uses 5¼ inch (133 mm) *diskettes* for storing information. (You may also have heard the terms "floppy disk," "mini-floppy," or "disk" - we will use "diskette.")

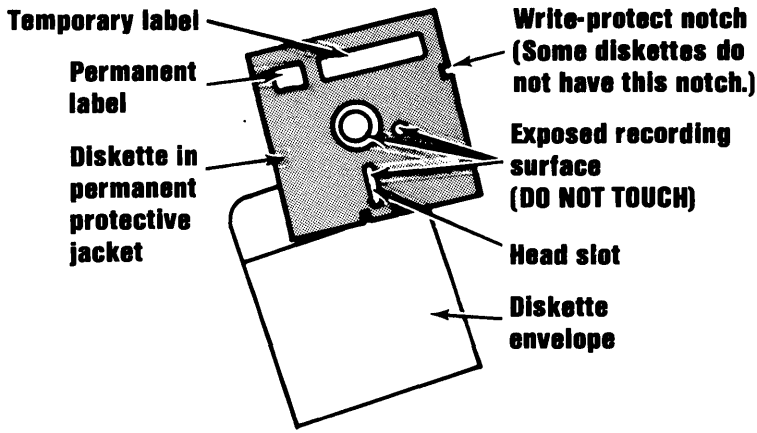
If you have single-sided drives, a diskette can hold either 163,840 or 184,320 characters of information. If you have dual-sided drives, you can format your diskette to hold either 327,680 or 368,640 characters of information.

The reason for showing two sets of numbers for each type of diskette is that diskettes formatted by DOS Version 1.00 or 1.10 contain only the lower number of characters. DOS Version 2.00, though, allows more data to be put on a single diskette by using more of the diskette's recording surface.

You can use diskettes formatted by DOS Version 1.00 and 1.10 with DOS Version 2.00, but you can not use diskettes formatted by DOS Version 2.00 with earlier versions of DOS, unless you use the FORMAT /8 or /B parameter when you format them.

Protective Jacket

The permanent protective jacket contains a flexible diskette that is coated with a magnetic substance. When in use, the diskette spins inside the jacket. The read/write head comes into contact with the recording surface through the long hole in the protective jacket, called the *head slot.*



How It Works

Information is written on or read from the magnetic surface of the diskette, similar to the way an ordinary tape recorder operates.

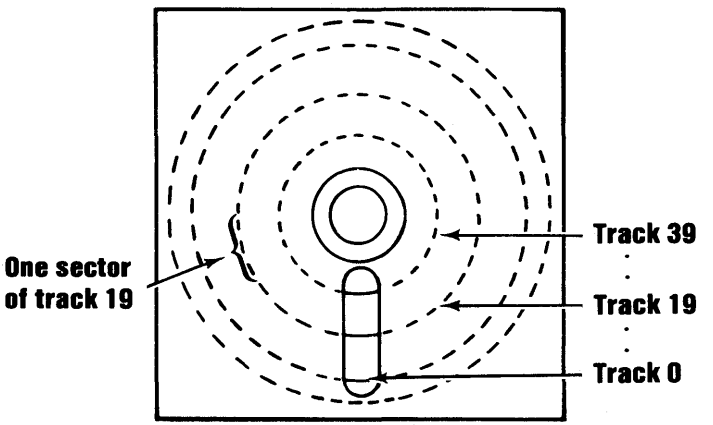
The information on a diskette can be read by the computer as often as it needs, or the computer can write new information on the diskette in an unused space.

The computer can also *replace* old information with new information by writing over it. In this case, the old information is erased and can no longer be read. Similarly, if you record a Chicago Symphony program on an Elvis Presley tape, you can no longer listen to Elvis.

Tracks, Bytes, and Sectors

Information is written on the diskette along concentric circles called *tracks*. The read/write head of the diskette drive moves back and forth from one track to another as the diskette spins over it. This lets the head find certain data to read or find a place to write some new information.

There are 40 tracks on a diskette, numbered from 0 to 39. DOS reserves portions of track 0 of each diskette. The rest of the diskette is available for your information and for a copy of DOS, if you want.



You'll also hear the words *byte* (pronounced like "bite") and *sector* used in talking about diskettes.

Space on a diskette (and the computer's memory, too) is measured in bytes. One byte can hold one character; thus, for diskettes formatted by DOS 2.00 the 5¼ inch diskettes can hold up to 184,320 characters for a single-sided diskette and up to 368,640 characters for a dual-sided diskette.

Each track is divided into eight or nine sectors that are 512 bytes long. One or more sector's worth of information can be sent back and forth between the computer and a diskette at one time.

Information on a diskette can be quickly located by its side, track, and sector numbers - just as the post office can locate your home by using the town (side number 0 or 1), street name (track number), and the address (sector number).

As long as you use DOS provided functions, you will never have to know (or use) this side, track, or sector, information—DOS will take care of it for you.

Careful Does It

Be careful with your diskettes. We'd like to emphasize these things here:

- Do not touch the exposed recording surfaces.
- Protect the diskettes from dust by putting them back in their envelopes *as soon as* you remove them from the diskette drive.
- Store often-used diskettes in their envelopes. Don't lay heavy objects on top of them. If you stand them on edge, make sure they aren't bending or sagging.
- Store seldom-used diskettes in storage boxes, away from heat and magnetic field sources such as telephones, dictation equipment, and electronic calculators.
- Because each piece of information occupies such a tiny spot on the diskette, small scratches, dust, food or tobacco particles may make the information unusable.

Take care of your diskettes because running your computer without programs and data is like running your car without gasoline.

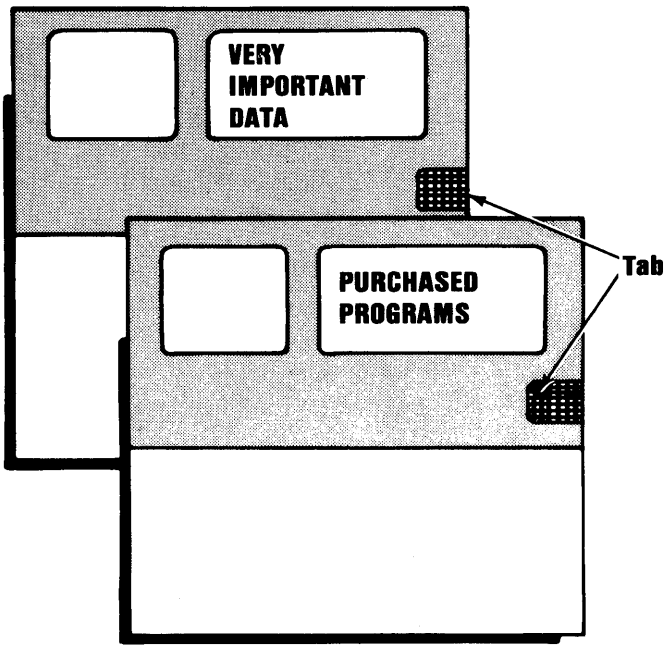
Write-Protect Notch

Besides making sure your diskettes aren't scratched or dirtied, you can make sure the computer cannot write over information already on a diskette.

If your diskette has no write-protect notch, it is already *write protected*. This means that the computer cannot store (write) any information on it. Your DOS diskette does not have a write-protect notch.

If your diskette has a *write-protect notch*, you can cover this notch with a tab supplied with the diskette, or use a piece of tape. Then the computer *cannot* write on the diskette. In this case, information can only be read from the diskette; information already on the diskette cannot accidentally be erased by being overwritten with new information.

Important diskettes (for example, purchased program diskettes) are often protected this way.



Just for reading

About Diskette and Drive Compatibility

DOS supports both single- and dual-sided diskette drives in any combination. Your drives do not need to be of the same type.

Diskettes formatted for single-sided use (see FORMAT Command in Chapter 6) can be used in either single- or dual-sided drives. However, diskettes formatted for dual-sided use are usable only in dual-sided drives, because data is recorded on both surfaces, and the method of allocating space is different. Therefore, you should never attempt to use a dual-sided diskette in a single-sided drive. Also, diskettes formatted by DOS Version 2.00 can not be properly used by prior DOS versions, if they are formatted at 9 sectors per track.

Single Diskette-Drive Systems

On a single diskette-drive system, you enter the commands the same way you would on a multiple diskette-drive system.

You should think of the single diskette-drive system as having *two* diskette drives (drive A and drive B). But, instead of A and B representing two physical drives as on a multiple diskette-drive system, the A and B represent diskettes.

If you specify drive B when the drive A diskette was last used, you are prompted to insert the diskette for drive B. For example:

```
A>COPY COMMAND.COM B:
Insert diskette for drive B:
and strike any key when ready
  1 File(s) copied
A>_
```

If you specify drive A when the drive B diskette was last used, you are again prompted to change diskettes. This time, the system prompts you to insert the drive A diskette.

The same procedure is used if a command is executed from a batch file. The system waits for you to insert the appropriate diskette and press any key before it continues. Remember that the letter displayed in the system prompt represents the default drive where DOS looks to find a file whose name is entered without a drive specifier. The letter in the system prompt does not represent the last diskette used.

For example, assume that A is the default drive. If the last operation performed was **DIR B:**, DOS believes the drive B diskette is still in the drive; however, the system prompt is still **A>**, because drive A is the default drive, and you did not specify another drive in the DIR command.

Automatic Program Execution

You may want to start a specific program every time you start DOS. You can do this with the DOS command processor by using *automatic program execution.*

Every time you start up DOS, the command processor searches for a file named AUTOEXEC.BAT in the root directory on the disk that DOS was started from. This filename is special because it refers to a *batch file* that is automatically executed whenever you start the system. With this facility, you can execute programs or commands immediately every time you start DOS.

If the system finds the AUTOEXEC.BAT file, the file is immediately executed by the command processor. The date and time prompts are bypassed.

If DOS does not find the AUTOEXEC.BAT file, DOS issues the date and time prompts. Refer to "Batch Processing" in Chapter 6 for details on how to create an AUTOEXEC.BAT file.

Chapter 2. Getting DOS Started

Contents

Loading (Starting) DOS

You will usually want to start DOS whenever you start your computer. For example, you must have DOS loaded before you can start the Disk BASIC or Advanced BASIC programs.

When you start or restart your computer, it will first check to see if it can load an operating system from diskette drive A. If a diskette is present, it will be read. If it is not present or the diskette drive door on drive A is open, one of the following two things will happen:

1. If your system does not have a fixed disk, the system will enter Cassette BASIC.

2. If your system does have a fixed disk, then an attempt will be made to load an operating system from it. If the fixed disk has not been initialized with an active partition (see Chapter 4), then Cassette BASIC will be entered.

The examples shown in this chapter describe using DOS in a *diskette-only* environment, and are intended to serve as an introduction to DOS. If you have a *fixed disk*, please refer to "Preparing Your Fixed Disk" in Chapter 4 after reviewing this chapter.

Starting DOS or *loading* DOS means that a copy of the DOS programs is read from the DOS diskette and placed in the computer's memory. Once the computer finishes its self-checks, you will hear the diskette drive whirring and clicking while the DOS programs are being read and transferred to memory.

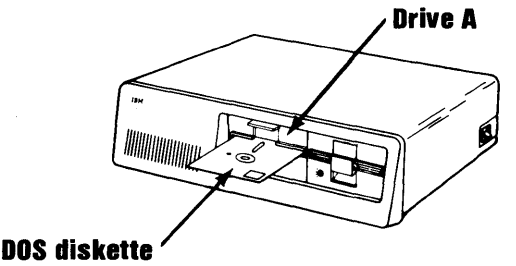
DOS tells you it is ready when it asks you for the current date and time. After that, DOS is ready for you to type a *command* - that is, to tell DOS what you want it to do.

Let's look at starting DOS, step-by-step.

There are two ways to start DOS, depending on whether your computer is off or on.

If Your Computer Is Off

1. Insert the DOS diskette into drive A and close the diskette drive door.

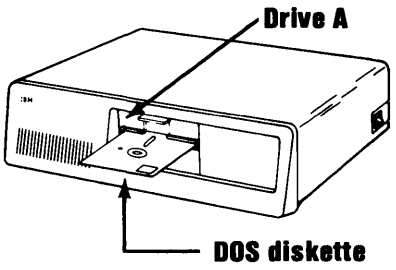


2. Turn on the printer, if you have one, the video monitor or TV, and then the computer.

3. Wait a moment while the system checks itself out. The length of the pause depends on the amount of memory in your computer. The more memory, the longer the pause. Then you will hear the diskette drive clicking and see the drive light come on while DOS is being read into the computer's memory.

If Your Computer Is On

1. Insert the DOS diskette in drive A and close the drive door.



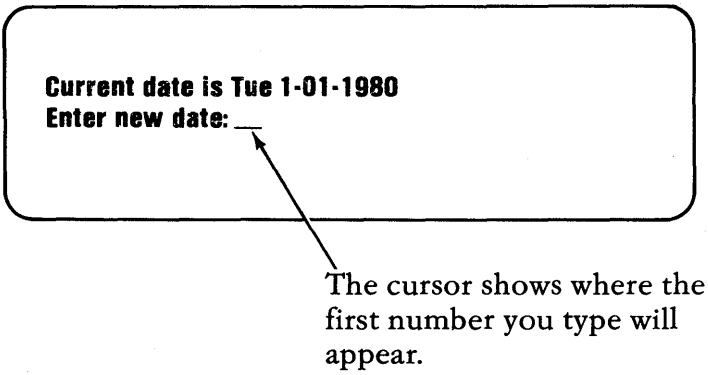
2. Press and hold Ctrl and Alt, and then press Del. Then release them all:

3. You will see the diskette drive light on while DOS is being read, and you may hear some clicks and whirs.

   Note that you will see these three keys used again in the section "To Start DOS Again" later in this chapter. This is the same-that is, use Ctrl + Alt + Del to start DOS or to restart DOS if your computer was already on and you were running something else. This is called *System Reset.*

Telling DOS the Date

When DOS is loaded and ready, you will see something similar to this on the screen:



The cursor shows where the first number you type will appear.

It's an excellent idea to fill in the current date whenever you start (or restart) DOS because then any files that you create or change will have the correct date stored in the file directory-helpful information months later, if you can't remember exactly which file has the most current information. (You'll learn about files and directories in a little bit.)

To set the date, use the number keys across the top of the keyboard:

1. Type one or two numbers between 1 and 12 for the month.

2. Type a dash  or a slash  .

3. Type one or two numbers between 1 and 31 for the day.

4. Type another dash or slash.

5. Type the last two numbers of the year between 80 and 99 or a four-digit number between 1980 and 2099.

6. Press the Enter key 

For example, suppose the current date is June 8, 1982.

Type this:	**6-8-82**
or this:	**6/8/82**
or this:	**6-08-82**
or this:	**6/08/82**
or this:	**06-08-82**
or this:	**06/08/82**
or this:	**06/8/82**

DOS checks the date that is typed. If the date that is typed does not check out, DOS displays this message:

```
Invalid date

Enter new date: _
```

These are some dates that would not check out:

Date	Reason
060882	No dashes or slashes
APR-02-80	Letters instead of numbers
13-08-83	Month too big
9-32-82	Day too big
10 15 82	Spaces, not slashes or dashes

If DOS tells you that the date is invalid, try again. Or if you do not wish to enter a new date, then press the Enter key when this is displayed:

```
Current date is 1-01-1980
Enter new date: _
```

Telling DOS the Time

After you have entered the date, DOS displays something similar to this:

```
Current time is 0:01:43.53
Enter new time: _
```

The time displayed is:

HOURS:MINUTES:SECONDS.HUNDREDTHS OF SECONDS

To set the time, use the number keys across the top of the keyboard:

1. Type one or two numbers between 0 and 23 for the hours.

2. Type a colon 

   Note: A dash  or a slash  will not work.

3. Type one or two numbers between 0 and 59 for the minutes. If you wish to enter the seconds and hundredths of a second, you can proceed with step 4, but if the hours and minutes are sufficient, proceed to step 8. DOS will set the remaining values to zero for you.

4. Type another colon.

5. Type one or two numbers between 0 and 59 for the seconds.

6. Type a period  to separate seconds and hundredths of a second.

   Note: Only a period will work.

7. Type two numbers between 00 and 99 for the hundredths of a second.

8. Press the Enter key 

For example, suppose the time is 8:30 a.m. exactly.

Type this:	**8:30:0.00**
or this:	**08:30:0.00**
or this:	**8:30:0**
or this:	**8:30**

DOS checks the time that you type. If the time that you type does not check out, DOS displays this message:

```
Invalid time
Enter new time: _
```

This is a time that would not check out:

Time	Reason
8/30/0/0	Only colons between hours, minutes, and seconds; and a period between seconds and hundredths of a second will work.

If DOS tells you the time is invalid, try again. However, if you do not wish to enter a new time, then press the Enter key when this is displayed:

```
Current time is 0:00:13.89
Enter new time: _
```


After you have entered the time, DOS displays this:

```
The IBM Personal Computer DOS
Version 2.00 (C) Copyright IBM Corp 1981,1982,1983
A>_
```

The DOS Prompt

The A> is the DOS *prompt*. A prompt tells you that it is your turn to type information; that is, to tell DOS what to do by entering a command.

This prompt tells you some other things besides saying that DOS is waiting for you to enter a command.

It tells you that DOS has completed the previous command, and that it is *DOS* that is waiting. Other programs have different prompts; for example, BASIC's prompt is: **Ok**

Ok lets you know that you should respond with a *BASIC* command.

Specifying the Default Drive

The A in the prompt designates the *default drive*. DOS searches the diskette located in the default drive to find any filenames that you enter unless you specify another drive.

You can change the default drive in the prompt by entering the new designation letter followed by a colon. For example:

```
A>      (original prompt)
A>B:    (new drive designation)
B>      (new prompt)
```

Now, B is the default drive. DOS searches the diskette located in drive B to find any filenames that you enter, unless you specify a drive.

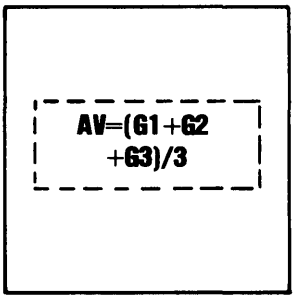
Remember, if you do not specify a drive when you enter a filename, the system automatically searches the diskette located in the default drive. Another name for the default drive is the *current* drive.

Files and Filenames

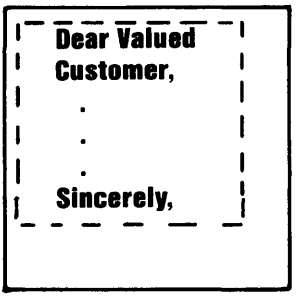
Related information on a disk is grouped into *files*, just as information in a book about particular topics may be grouped into chapters.

Each file has a name–when you want DOS to find a file, you give DOS its name (not the name that's on the diskette's label).

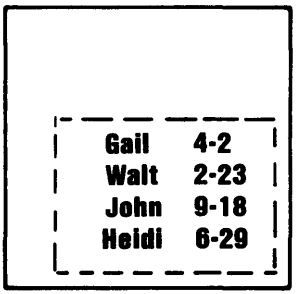
Some examples of files:



A file containing a program to calculate bowling team averages.



A file containing the text of a form letter your company sends out.



A file containing names and birthdays of your friends.

A file containing a program to print a birthday message to all your friends born this month.

Files are used so that DOS can find specific information easily, and so that information that isn't needed isn't taking up room in the computer. (For example, you don't need your bowling team averages at the same time as you need the form letter.)

You usually have a number of files on a diskette. You can have up to 64 files in the system directory of a single-sided diskette and 112 files on a dual-sided diskette. The number of files on a fixed disk is determined by the amount of space allocated to DOS. Sometimes the files on one disk are related to each other (like the programs and data files to keep track of a company's inventory), and sometimes the files have been put on whatever diskette was handy.

It doesn't matter what combination of files is on a disk. What matters is that each file has a unique name.

That means that every name on a disk has to be different—but you can have the same name on two different disks.

For our examples a little earlier, the names of these files might be:

BOWL
LETTER
BIRTHDAY
BIRTHDAY.BAS

A file's name is made up of a *filename* and an *extension.*

In DOS, *filenames* are from one to eight characters long. The characters in a filename can be:

- the letters of the alphabet
- the numbers 0 through 9
- and these special characters -

  $ # & @ ! % ( ) - { } ' _ '

  Note: Prior versions of DOS allowed the characters ¦ , <, >, and to be used within a filename. However, these characters have special meaning to DOS Version 2.00, and can no longer be used in filenames.

A filename can be followed by an optional short name called an *extension*. An extension starts with a period, has one, two, or three characters, and follows immediately after the filename.

Here are some filenames with extensions:

81PRICES.JUL
81PRICES.AUG
AVRG.$&%
WEATHER.80

Important: If a filename is followed by an extension, you *must* use both parts when telling DOS about that file.

Bad, Okay, and Good Names for Files

With all the possibilities, filenames can be unusual, to say the least.

DOS likes names that follow the rules. These files have names that DOS will not accept:

Name	Why DOS Won't Accept It
A AND B	Spaces in it
A,B, & C	Commas and spaces in it
.PGM	Filename missing
ANDTHISONEISTOO.LONG	

These names are okay for DOS:

```
()().XXX
#1#2#A3B
@@.---
Z
```

but can you guess what is in any of these files?

A *good* name for a file will help you remember what kind of information is in the file, and perhaps whether it's a file that contains a program or only data. For example, ADDRLIST.BAS is a good name for a BASIC program file that prints an address list.

File Specifications

The other thing that DOS needs to know to find information is "Where"–that is, which drive to search for a particular file.

The *drive specifier* is a letter and a colon, like **A:**, and it tells where the file is. You always need to type the colon after the drive specifier letter.

The filename and extension immediately follow the drive specifier, like this:

A:81PRICES.JUL

Don't put any spaces between the three parts. These three parts together–the drive specifier, the filename, and the extension–are called a *file specification.*

Sometimes you don't have to type the drive specifier; whenever the drive specifier is the same as the default drive, you don't have to type it.

For example, assume A is the default drive (remember you can tell by looking at the prompt–A> in this case). Then you could type:

A:address OR address ← **Look–no drive specifier!**

They are exactly the same to DOS when A is the default drive.

Here are some more file specifications:

A:81PRICES.AUG
A:KIKI.J
B:BOWL.BAS
B:MINE
YOURS ← **With this kind of file specification, DOS assumes that the file (YOURS) can be found on the default drive.**

Using Global Filename Characters

Sometimes you will want to do the same thing with several files–for example, copying a group of files at one time, or listing the names of a group of files that are somehow related.

Two special *global filename characters* let you indicate a number of files with one specification. These characters are the question mark (?) and the asterisk (*). They are used in a filename and/or an extension to mean "any character."

These characters can save you a lot of typing if your files are named appropriately. Let's look at some examples to tell you about global filename characters.

In the examples that follow, you need to know that the DOS command DIR displays information about files that match the file specification you type.

The ? Character

The ? in a filename or extension means that any character can be in that position. So, all files that have a name that matches in all except the ? positions are selected. For example, suppose the diskette in drive A has these files on it:

```
79PRICES.AVG
80PRICES.AVG
81PRICES.JAN
81PRICES.JUL
81PRICES.AUG
79INVTRY
```

If you type:

```
dir ??prices.???
```

all the files except the last one (79INVTRY) are listed on the screen. These are the files that match and are listed:

```
79PRICES.AVG
80PRICES.AVG
81PRICES.JAN
81PRICES.JUL
81PRICES.AUG
```

For another example, suppose you give DOS this command:

```
dir ??prices.a?g
```

Then these files match and are listed:

```
79PRICES.AVG
80PRICES.AVG
81PRICES.AUG
```

The * Character

The * in a filename means that any character can be in that position and in the rest of the filename. Likewise, a * in an extension means that any character can be in that position and in the rest of the extension. Using an asterisk is like typing several ?'s, two or more * are never used together. One is enough!

Assume that the same diskette we saw with "the ? character" is in drive A. If you give DOS this command:

```
dir 81*.j*
```

Then these files are selected and displayed:

```
81PRICES.JAN
81PRICES.JUL
```

For another example, perhaps we try this:

```
dir 8*.*
```

These files would be listed:

```
80PRICES.AVG
81PRICES.JAN
81PRICES.JUL
81PRICES.AUG
```

And if you type:

```
dir *.*
```

all the files would be listed.

You can use both global filename characters together. For example, if you type:

```
dir ??p*.a*
```

you would see these files listed:

```
79PRICES.AVG
80PRICES.AVG
81PRICES.AUG
```

Global characters are fun, but you need to be careful both in naming your files and in using them–you may get results you didn't expect!

So far we have discussed:

- how to get DOS started
- the DOS prompt and default drive
- formatting your diskettes
- files and filenames
- global filename characters

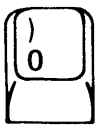
Let's take a closer look at the keyboard keys before we go any further.

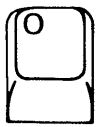
Some Keys You Use with DOS

In addition to the keys you'd find on a typewriter, your keyboard has some special keys you'll use with DOS.

Before we get to the special keys, here are a few differences between your keyboard and a typewriter that you need to know.

Characters That Look Alike – Oh's and Zeros, Ones and Els

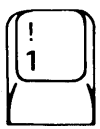


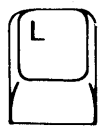


Computers are fussy about the number zero and the letter **O**—they want what they want and you can't fool them into taking the wrong one. Make sure you type the right key in commands and filenames.

On our printer, the letter **O** looks a little squarer than the number **0**.

On our screen, the number zero has a diagonal line through it, something like this: Ø.

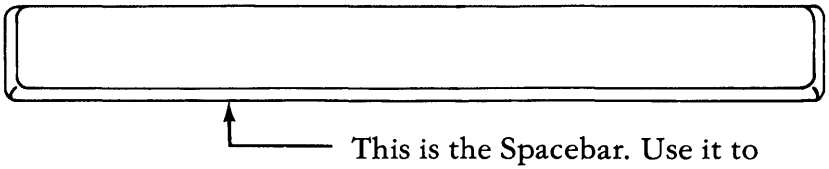




If you are used to typing a lowercase **L** for the number **1** (if you have used a portable typewriter a lot, for example), you'll have to break that habit.

Again, the computer knows the difference. Use the number key when the number **1** is required, as in a filename such as DATA123. (Now if you were to use lowercase **L** when creating a filename, that's okay. Just be *sure* to use it again when you want to use that file!)

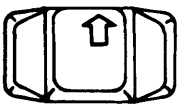
The Spacebar



This is the Spacebar. Use it to put a blank (a space) in a line you are typing. Sometimes people want to use it for moving the cursor. It will move the cursor, but it will also *replace* any characters with blanks as it moves across the screen.

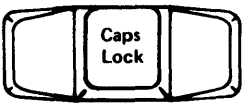
To the computer, blanks are important; a blank is as much a character as A or B. Many times blanks are used to separate what you type for the computer just as we use them to separate words in everyday writing. You have to make sure when you're typing that the computer allows a blank. Otherwise, it may not understand what you have typed.

The Shift and Caps Lock Keys



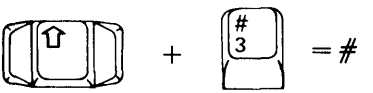
There are two Shift keys on the keyboard, located about where you find them on a typewriter keyboard.

Use them to type uppercase (capital) letters or to type the symbol shown in the upper position on the key tops for all keys except the numeric key pad on the right. That's just as you would expect.



The Caps Lock key lets you type capital letters (only) until you press it again. Only the letter keys are affected.

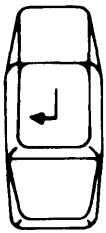
You still have to press one of the Shift keys to type the symbols in the upper position of the number keys at the top of the keyboard. For example, you must press and hold the Shift key to type a #.



Unlike a regular typewriter, pressing one of the Shift keys  does not get you out of Caps Lock mode. You must press the Caps Lock key again to type lowercase letters.

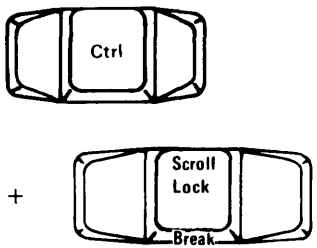
Remember that DOS will accept lowercase letters in all commands and filenames; so you'll be using these Shift keys mostly for special symbols.

To Enter a Command



Use the Enter key when you have finished typing a whole command.

To Stop a Command

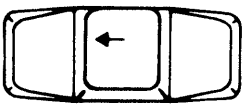


Press and hold the Ctrl (Control) key, and then press the Break key. Then release both keys to stop a command from finishing its job normally. (This is sometimes called *terminating* a program).

DOS shows you a prompt; then you can type your next command.

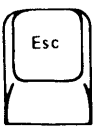
To Correct a Typing Mistake

Here are several ways to correct a mistake that you notice *before* you press the Enter key.



One of the easiest is to move the cursor backward, under the leftmost wrong character. The Backspace key deletes characters as it moves the cursor to the left. (The Backspace key is located on the top row next to the Num Lock key.)

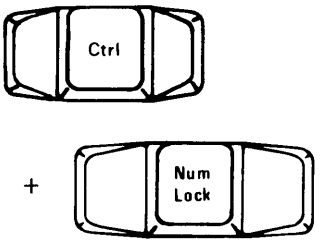
Now type the correct characters, and press the Enter key when everything is the way you want it.



Another way, if the line is just too messed up to worry about, is to press the Esc (Escape) key. A backslash (\) is displayed, and the cursor moves down one line on the screen. This cancels the messed-up line and you can then type the command correctly.

More ways to correct typing mistakes, which we call *editing*, are discussed in Chapter 3; but these two keys should be enough to get you started.

To Stop the Screen Long Enough to Read It

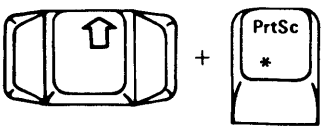


If information is appearing on the screen too fast for you to read, press and hold the Ctrl (Control) key and then press the Num Lock (Number Lock) key. Then release both of them.

When you are ready to see some more information, press *any* key.

(*Scrolling* is the term used to describe how a line of information is displayed on the screen and then is pushed upward until it is pushed off the top of the screen by new lines of information being displayed at the bottom.)

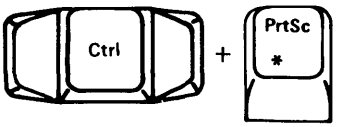
To Print What Is on the Screen



If you have a printer and want to print what is currently on the screen, first make sure the printer is on. Then press and hold either Shift key, and then press the PrtSc (Print screen) key; then release both of them.

What is printed is often called a *hard copy*.

To Print Whatever You Type

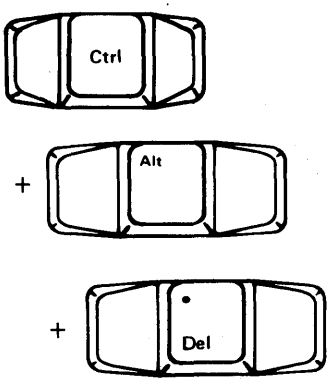


If you have a printer and want to print whatever you type and what the computer displays, press and hold the Ctrl (Control) key and then press the PrtSc (Print Screen) key. Then release them both. Now, each time you press the Enter key, or the computer displays a line, the line will be printed or *echoed* on the printer.

To stop echoing to the printer, press the Ctrl and PrtSc keys again.

This is *different* from Shift and PrtSc. Shift and PrtSc prints a whole screen's worth and then is done. But Ctrl and PrtSc prints one line at a time, line after line, until you press Ctrl and PrtSc again to stop the printing.

To Start DOS Again



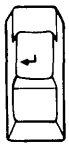
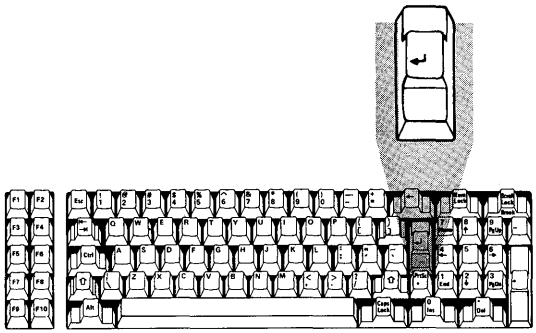
If you want to start DOS over from the beginning, put your DOS diskette into drive A. Then press and hold down the Ctrl (Control) and Alt (Alternate) keys and then press the Del (Delete) key. Then release all three. Remember, you may see this called *System Reset.*

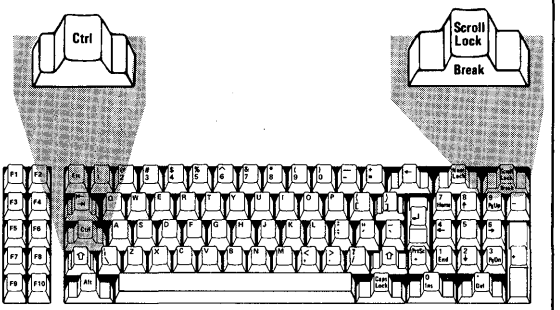
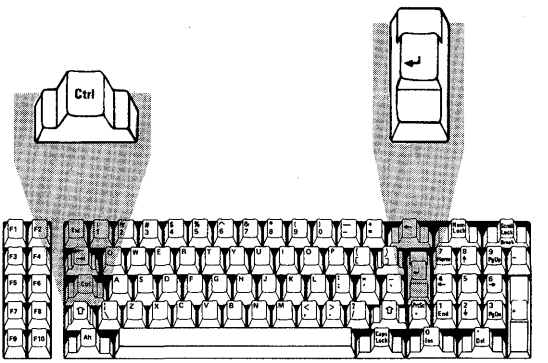
After a bit, you'll see the DOS startup messages.

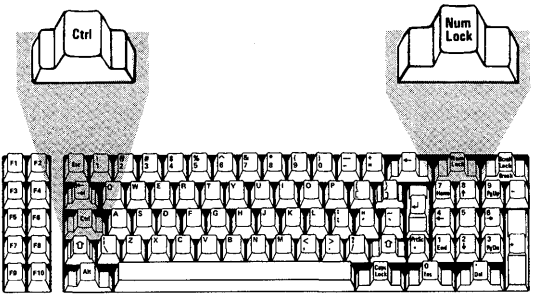
Control Keys

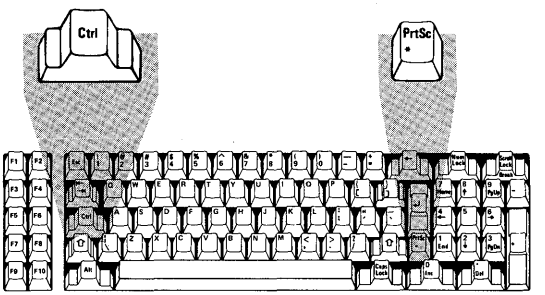
Use the control keys when you are entering commands or input lines to any program. Where two keys are specified, for example Ctrl-Break, you must press and hold down the first key and then press the second key.

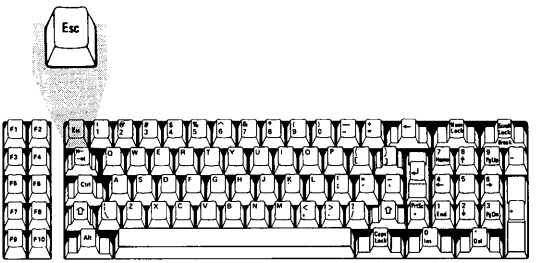
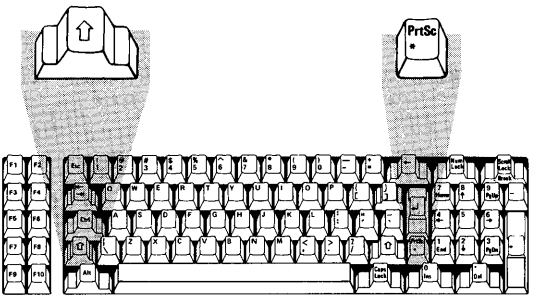
Here is a summary of the control keys, their functions, and their locations on the keyboard:

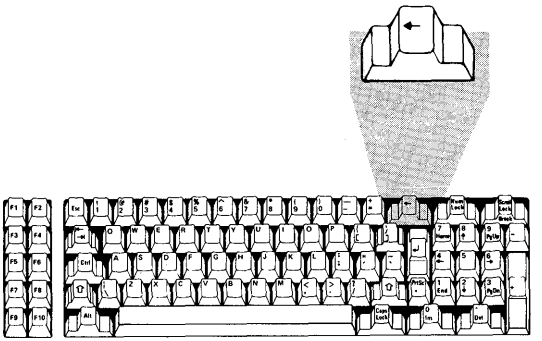
Control Key	Function
	This is the Enter key. You press the Enter key to send the displayed line to the requesting program. 

Control Key	Function
Ctrl-Break	Ends (cancels) the current operation. 
Ctrl-Enter	Allows you to go to the next displayed line on the screen to continue entering the line you are typing. 

Control Key	Function
Ctrl-Num Lock	Suspends system operation. You must press any character key to resume operation. This is useful when a large amount of screen output is being generated. You can press Ctrl-Num Lock to temporarily suspend the display of your output so you can review it. You can then press any other character key to restart the display. 

Control Key	Function
Ctrl-PrtSc	These keys serve as an on/off switch for sending displayed output to the printer as well as to the screen. You can press these keys to print displayed output on the printer and press them again to *stop* printing displayed output on the printer. Although this allows the printer to function as a system log, it slows down some operations because the computer waits during the printing. 

Control Key	Function
Esc	Cancels the current line and moves to the next displayed line. A backslash (\) is displayed to indicate the canceled line. 
Shift-PrtSc	Sends a copy of what is currently displayed on the screen to the printer. This, in effect, prints a "snapshot" of the screen. 

Control Key	Function
←	Backspaces and removes a character from the screen. This is the key to the left of Num Lock, *not* key 4 on the numeric key pad. 

DOS Editing Keys

Use the DOS editing keys to make corrections to commands and input lines as they are being entered. Note that the meaning of these keys can change if you alter their assignments through extended keyboard control, refer to "Using Extended Cursor and Keyboard Control" in Chapter 13 for additional information.

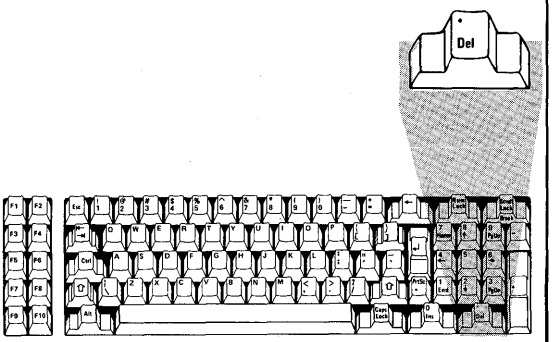
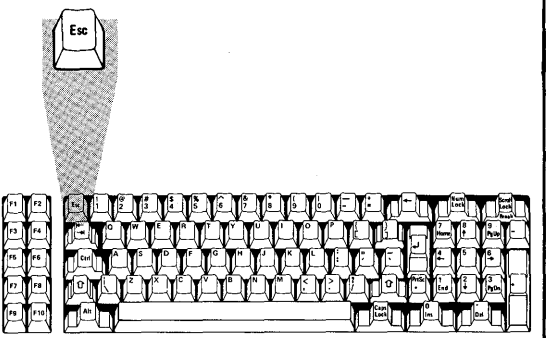
The DOS editing keys are used to edit *within* a line. The Line Editor (EDLIN) program operates on *complete lines* within a file or document. When you are working with EDLIN and want to edit within a line, however, use the DOS editing keys. For more information about EDLIN, refer to Chapter 7.

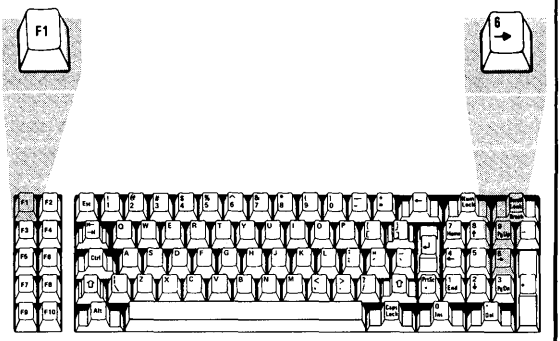
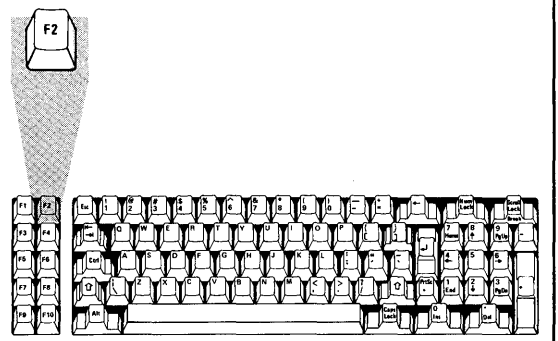
> **Note:** Some word processing programs define special editing rules; therefore, the DOS editing keys may not work as described in this chapter. You can also define special editing rules when using the BASIC Program Editor while programming in BASIC.

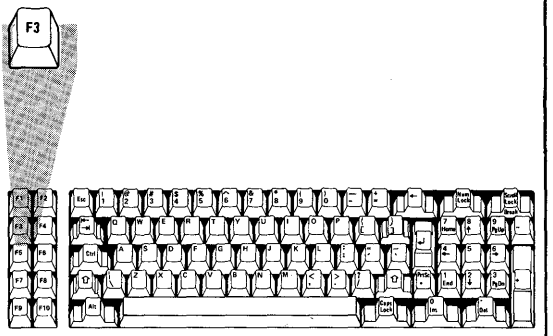
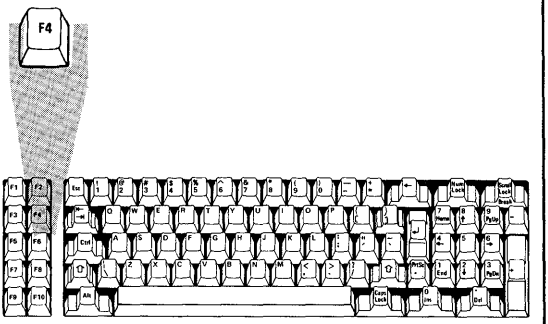
Any line you enter from the keyboard is retained in an input buffer when you press Enter. The line is then made available to your program for processing.

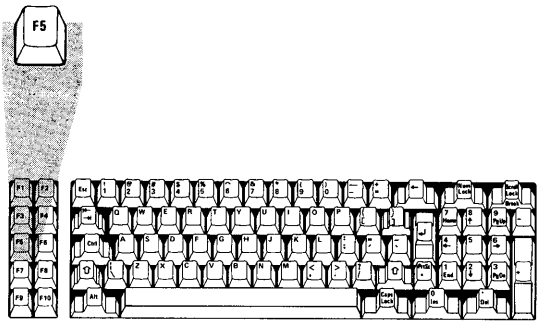
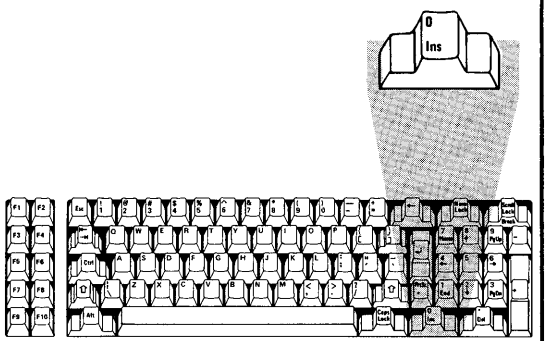
Since the line remains in the input buffer, you can use that line as a *template* for editing purposes. The DOS editing keys operate on that copy of the line. You can repeat or change the line by using the DOS editing keys, or you can enter an entirely new line.

Here is a summary of the DOS editing keys, their functions, and their locations on the keyboard:

DOS Editing Key	Function
Del	Skips over one character in the template. The cursor does not move. 
Esc	Cancels the line currently being displayed. The template remains unchanged. 

DOS Editing Key	Function
F1 or →	Copies one character from the template and displays it. 
F2	Copies all characters up to a specified character. 

DOS Editing Key	Function
F3	Copies all remaining characters from the template to the screen. 
F4	Skips over all characters up to a specified character. (F4 is the opposite of F2.) 

DOS Editing Key	Function
F5	Accepts an edited line for continued editing - the currently displayed line becomes the template, but it is not sent to the requesting program. 
Ins	Allows you to insert characters within a line. 

Examples of Ways to Use DOS Editing Keys

The following examples show how you use the DOS editing keys with the Line Editor (EDLIN) program.

If you want to try these examples, you must use the EDLIN program. The EDLIN program is on your DOS diskette and is discussed in Chapter 7. You do not need to review the EDLIN chapter to complete these examples - just follow the steps provided.

Notes:

1. Because the DOS diskette shipped with your IBM Personal Computer is *write protected*, you cannot create the file used in the following examples on that diskette. You must use a *copy* of your DOS diskette to complete these examples. Refer to the section called "Write-Protect Notch" in Chapter 1, for more information about write protected diskettes.

2. In the following examples, to *enter* something means that you should *type* the information and then press the Enter key.

3. If you finish one or more of the following examples and you do not want to try the rest of the examples, go to "To Stop the Editing Session" at the end of this chapter.

To Start EDLIN

1. Insert your DOS diskette into drive A.

2. Create a file named EXAMPLES.

   If you want the EXAMPLES file to reside on the diskette in your default drive, enter:

   ```
   EDLIN EXAMPLES
   ```

   or

   If you want the EXAMPLES file to reside on the diskette in another drive, you must specify the drive, as in:

   ```
   EDLIN B:EXAMPLES
   ```

   This command tells DOS to load the EDLIN program and create a file called EXAMPLES.

   The following message and prompt will be displayed:

   ```
   New file
   *_
   ```

   Notice that the prompt for EDLIN is an asterisk (*).

3. Now, enter the letter I.

   This tells EDLIN that you want to begin *inserting* lines in the file named EXAMPLES.

   The screen looks like this:

   ```
   New file
   *I
     1:*_
   ```

4. Type **This is a mailorder file.** on line 1 and press Enter.

5. Type **Editing is easy.** on line 2 and press Enter.

   You now have two lines of text in your EXAMPLES file.

6. Press the Ctrl-Break keys.

   Pressing Ctrl-Break will end the insert mode of operation and return you to the EDLIN prompt.

7. Enter the number 1.

   This tells EDLIN that you want to display line 1 on the screen.

   The screen should look like this:

```
1:*This is a mailorder file.
2:*Editing is easy.
3:*_
```

You are now ready to begin the examples.

Note: If you encounter any problems while trying these examples, press the Ctrl-Break keys. The EDLIN prompt will be displayed and you can start over.

Example 1

Let's delete the first two characters in the word **This** and then copy the remainder of the line.

1. Press the Del key twice to delete the first two characters.

2. Press F3 to copy the remainder of the line to the screen. The screen looks like this:

```
1:*This is a mailorder file.
1:*is is a mailorder file.__
```

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt. (The changes you made to line 1 will not be saved.)

2. Enter the number 1.

Example 2

Now we'll change line 1; then, using Esc, we will cancel the change. A backslash (\) will be displayed to indicate that the displayed line has been cancelled.

> **Note:** If the insert mode is on, the system automatically turns it off when you use Esc.

The screen looks like this:

```
1:*This is a mailorder file.
1:*__
```

To change line 1 to **Sample file**:

1. Type **Sample file**, but do *not* press Enter.

```
1:*This is a mailorder file.
1:*Sample file_
```

2. To cancel the line we just entered, press the Esc key.

```
1:*This is a mailorder file.
1:*Sample file\
  _
```

   Now we can continue to edit the original line **This is a mailorder file.**

3. Press F3 to copy the original line to the screen.

   The screen looks like this:

```
1:*This is a mailorder file.
1:*Sample file\
   This is a mailorder file._
```

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number 2.

Example 3

Now let's copy one character by using F1 or → . (F1 or → is the opposite of Del. Del skips over one character in the template.)

The screen looks like this:

```
2:*Editing is easy.
2:*_
```

1. Press the F1 or → key three times.

   The screen looks like this:

   ```
   2:*Editing is easy.
   2:*Edi_
   ```

   Each time you press the F1 or → key, one more character appears.

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number **2**.

Example 4

Now let's use the F2 key. Remember, the F2 key copies all characters from the template to the screen up to, but not including, the first occurrence of a specified character.

You must always specify a character when using this key. If the specified character is not present in the template, nothing is copied.

The screen looks like this:

```
2:*Editing is easy.
2:*_
```

1. Press the F2 key and enter the letter **g**.

   The screen looks like this:

   ```
   2:*Editing is easy.
   2:*Editin_
   ```

   Now we'll copy all the remaining characters in the template to the screen by using the F3 key.

   (If you pressed Enter now, only **Editin** would be saved in the EXAMPLES file as line 2.)

2. Press the F3 key.

   The screen looks like this:

   ```
   2:*Editing is easy.
   2:*Editing is easy._
   ```

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number 1.

Example 5

Now let's scan and locate specific characters within the template by using the F4 key. This is a way to skip over characters. The cursor does not move when you use this key, and no characters are displayed.

You must always specify a character after you press the F4 key. If the specified character is not present in the template, no characters in the template will be skipped.

We will also use the F3 key to copy the remaining characters in the template to the screen.

The screen looks like this:

```
1:*This is a mailorder file.
1:*_
```

1. Press the F4 key and enter the letter **o**. (No characters are displayed.)

2. Press the F3 key to copy the remainder of the line.

   The screen looks like this:

```
1:*This is a mailorder file.
1:*order file._
```

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number **1**.

Example 6

Now we'll move the currently displayed line into the template by using the F5 key. Pressing F5 is the same as pressing Enter, except that the line is *not* sent to your program. An @ character is displayed to indicate that the new line is now the template.

> **Note:** If the insert mode is on, the system automatically turns it off when you use F5.

Once you press F5, you can continue to make changes to a line. When you are finished, press Enter to send the line to your program.

The screen looks like this:

```
1:*This is a mailorder file.
1:*_
```

1. Type **This is not a sample file.**

   The screen looks like this:

   ```
   1:*This is a mailorder file.
   1:*This is not a sample file._
   ```

2. Press F5.

   The result is:

   ```
   1:*This is a mailorder file.
   1:*This is not a sample file.@
     —
   ```

   The replacement line **This is not a sample file.** is now in the template. The replacement line is acceptable, but let's continue to edit it.

3. To remove the word **not** from the replacement line, press F1 eight times:

```
1:*This is a mailorder file.
1:*This is not a sample file.@
```

4. Press Del four times to remove one blank space and the word **not**.

5. Press F3 to copy the remaining characters to the screen.

   The screen looks like this:

```
1:*This is a mailorder file.
1:*This is not a sample file.@
   This is a sample file.__
```

6. Press Enter to make the replacement line **This is a sample file.** the template in place of the original line *and* to send the line to your program.

   (If you want to do more editing without sending the line to your program, press F5 again to put the displayed line into the template.)

   > **Note:** Pressing Enter *immediately* after pressing F5 empties the template.

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number 1.

Example 7

Let's look at an example using the Ins key. The Ins key serves as an on/off switch for entering and leaving insert mode. You can press the Ins key to enter insert mode, and press the Ins key again to leave the insert mode.

While in the insert mode of operation, any characters that you enter are *inserted* in the line being displayed. The characters do not *replace* characters in the template.

When you are not in the insert mode of operation, any characters that you enter *replace* characters in the template. If you are entering characters at the end of a line, the characters will be added to the line.

The screen looks like this:

```
1:*This is a sample file.
1:*_
```

Let's change the word **sample** to **salary.**

1. Press the F2 key and enter the letter **m.**

   The screen looks like this:

   ```
   1:*This is a sample file.
   1:*This is a sa_
   ```

2. Press the Ins key and enter the characters **lary.**

   The screen looks like this:

   ```
   1:*This is a sample file.
   1:*This is a salary_
   ```

Notice that the characters **lary** were inserted, but no characters from the template were replaced.

3. Now, press Ins again to leave the insert mode.

4. Enter one blank space and the three characters **tax**.

```
1:*This is a sample file.
1:*This is a salary tax_
```

5. Press F3 to copy the remaining characters in the template to the screen.

```
1:*This is a sample file.
1:*This is a salary tax file._
```

Notice that we *inserted* **lary** and we *replaced* **mple** with a blank space and **tax**.

6. Now press Enter to make the replacement line the template in place of the original line and send the line to the requesting program.

To Stop the Editing Session

You have now completed the examples.

To return to the A> prompt:

1. Press Ctrl-Break.

2. Enter the letter Q.

   Q tells EDLIN that you don't want to save the EXAMPLES file and that you want to *quit* the editing session. EDLIN will prompt you with this message:

   Abort edit (Y/N)?

   to make sure you don't want to save the file.

3. Enter the letter Y.

Chapter 3. Using DOS

Contents

Introduction

This chapter steps you through procedures that do some everyday tasks. Read it now to get familiar with these commands and then you can use it to refer to when you really have to do that task.

The examples shown in this chapter describe using DOS in a *diskette-only* environment, and are intended to serve as an introduction to DOS. If you have a *fixed disk*, please refer to "Preparing Your Fixed Disk" in Chapter 4 after reviewing this chapter.

Just remember, you will have to *substitute the names of your own files* for those we have used here–it's that easy.

Giving DOS a Command

To give DOS a command:

1. Wait until you see the DOS prompt, A>.

2. Type the command and any other parts the command requires (for example, a drive specifier or a file specification).

   You can type in uppercase or lower case letters (or a combination). Use a blank (the Spacebar) to separate the parts of the command from each other.

3. Press the Enter key when you have finished typing.

Getting a Diskette Ready to Be Used

Using the FORMAT Command

The FORMAT command gets a diskette ready to receive information. FORMAT checks the diskette for bad spots, builds a directory to hold information about the files that will eventually be written on it, and optionally, copies the DOS system files onto the diskette.

Sometimes the terms *initializing* and *preparing* are used instead of *formatting*.

You *must* use FORMAT before you try to use a *new* diskette unless you are copying from another diskette using the DISKCOPY command.

You can use FORMAT if a diskette has developed defective areas. FORMAT makes sure these areas are not used for your files. FORMAT and CHKDSK will tell you if there are defective areas on your diskette.

You can also use FORMAT as a way to prepare a diskette that has information you *no longer need* for use again.

DO NOT use FORMAT each and every time you want to put information on a diskette, because FORMAT wipes off what was already there.

To sum up, you need to use FORMAT once per diskette, when it is new. Very occasionally you will re-FORMAT a diskette if it has defective areas or if you want to forget what was on it and use it as if it were new.

> **Note:** If you have a fixed disk, it will also need to be initialized. Refer to "Preparing Your Fixed Disk" in Chapter 4 for a description of the initialization procedure.

Before You Begin

Have on hand your DOS diskette and the diskette you want to format.

Also decide whether you want a copy of DOS on your new diskette. Sometimes you will want DOS on a diskette if you will be putting major programs on it. On the other hand, if the diskette is going to be used for data files, you will probably not want DOS on the diskette.

If You Want DOS on Your Diskette

With One Drive

1. Make sure DOS is ready and A> is displayed.

2. Insert your DOS Diskette into drive A, if it is not already there.

3. Type:

format a:/s

and press the Enter key. If the target drive is dual-sided, this will create a diskette for dual-sided use only. If you want to format a diskette that can be used in either a single-sided or a double-sided drive then type:

format a:/s/1

4. Now you will see this message:

Insert new diskette for drive A
and strike any key when ready

5. When the drive A "in use" light is off, remove the DOS diskette from drive A.

6. Put your new diskette into drive A.

7. Press any key—the Spacebar for instance.

8. You will see this message:

Formatting...

9. After some whirring and clicking, you will see this:

```
Formatting...Format complete
System transferred

    xxxxxx bytes total disk space
    xxxxxx bytes used by system
    xxxxxx bytes available on disk

Format another (Y/N)?
```

10. Type:

    n

    You don't have to press the Enter key.

11. Now you will see the DOS prompt, **A>**, and you can remove your newly formatted diskette.

    The diskette is now ready to be used. It has a copy of the DOS system on it (that's because in step 3 you typed **/s**–the *s* stands for *system*).

With Two Drives

If you have two diskette drives, insert your DOS diskette into drive A and put your new diskette into drive B.

Then, for step 3, type:

format b:/s
(format b:/s/1 for single-sided)
or
format b:
(format b:/1 for single-sided)

depending on whether you want the DOS system on your diskette.

In step 4, the message is now going to be:

Insert new diskette for drive B
and strike any key when ready

The rest of the steps are the same except you will *not* have to remove the DOS diskette.

If You Do Not Want DOS on Your Diskette

If you don't want DOS on your diskette, in step 3 you can type instead:

format a:

or

format a:/1 for single-sided format

and press the Enter key. Then the line that says **System transferred** (step 9) does not appear.

Formatting Several Diskettes

If you want to format several diskettes in a row, follow steps 1 through 9. But do this at step 10.

Type:

y

(You probably already guessed.)

You will again see the message at step 4. Follow the same steps for each diskette you want to format at this time.

Backing Up a Diskette

Using the DISKCOPY Command

Backing up a diskette means to make a copy of the diskette's data on another diskette. Similarly, *backing up a file* means to make a copy of the file (usually on a different diskette). (Backing up one file is discussed in the next part of this chapter.)

A *backup*, that is, the copy, saves you the time, trouble, and sometimes, the expense of recovering the information on a diskette that has been lost, damaged, or accidentally written over. A backup often saves your temper, too.

It is a good habit to back up your *important program* diskettes as soon as you purchase or create them. Then, store your original diskettes properly in a place where you can find them if you need to. And then use the backup diskettes for everyday operations.

> **Note:** Some purchased program diskettes cannot be copied. In these cases, the documentation that comes with the programs will explain the best methods of backing them up.

Your *diskettes* should be backed up every time you add or change information on them.

Before You Begin

You will need these diskettes:

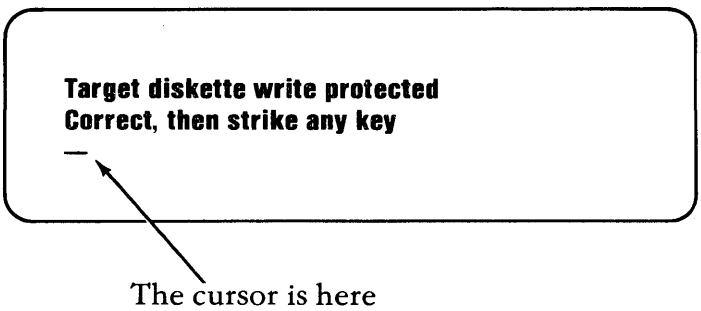
- The diskette you want to back up–we're going to call this your *original* diskette. You may also see it called the *source* diskette.

- The diskette that will become the backup diskette. Other names for this diskette are the *target* diskette, or the *destination* diskette.

Protecting Your Original Diskette

Hint: It's a good idea to *put a tab* over the write-protect notch in order to make sure your original diskette is not accidentally written on. Then you can remove it when the backup has been made, if you want to.

With the write-protect notch covered, if the diskettes get mixed up, a message similar to this is displayed:



If you get this message:

1. Remove the original diskette from the drive.
2. Insert the backup diskette into the drive.
3. Press any key.

   (You do *not* have to press the Enter key.)

Backing Up with One Drive

If you have only one diskette drive, you must remove the original diskette and insert the backup diskette. You will have to make this switch ("swap") four or more times. How many times you must exchange diskettes depends on whether you're using DISKCOPY or COPY, or how large the files being copied are, and on how much memory your computer has. DOS will tell you when you must exchange diskettes. DISKCOPY gives you these messages:

```
Insert source diskette in drive A:
```

```
Insert target diskette in drive A:
```

So you should:

Insert	When
Original diskette	"source" message is displayed
Backup diskette	"target" message is displayed

1. Make sure DOS is ready and **A>** is displayed.
2. Insert the DOS diskette into the drive, if it is not already there.
3. Type this:

   diskcopy

   and press the Enter key.
4. This message appears:

> **Insert source diskette in drive A:**
>
> **Strike any key when ready**

Before You Press a Key:

a. Remove the diskette that is in the drive.

b. Insert your *original* diskette into the drive.

c. *Now* press a key.

5. You will see the "in use" light come on while the original diskette is being read, and then this is displayed:

```
Insert target diskette in drive A:

Strike any key when ready
```

Before You Press a Key:

a. Remove your original diskette.

b. Insert the *backup* diskette.

c. *Now* press a key to tell DOS the correct diskette has been inserted.

6. You will see the "in use" light come on while the backup diskette is being written. Then the message shown in step 4 appears again.

   Hint: For this procedure, you can remember which diskette to insert, if you remember "Original=Source." Insert your original diskette when DISKCOPY asks for the source diskette.

7. Keep repeating steps 4 and 5 until this message appears:

```
Copy complete

Copy another (Y/N)?
```

8. Type this:

   n

   You don't have to press the Enter key.

9. The DOS prompt, A>, is displayed. Remove the backup diskette from the drive. With a felt-tip pen, mark the label with the contents, the date, and perhaps the word "Backup."

Backing Up with Two Drives

1. Make sure DOS is ready and A> is displayed.
2. Insert your DOS diskette into drive A.
3. Type this:

   diskcopy a: b:

   and press the Enter key.

4. You will see this message:

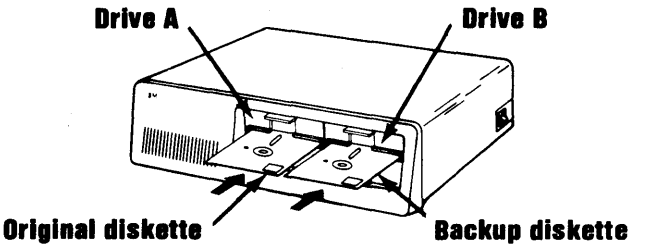
   Insert source diskette in drive A:

   Insert target diskette in drive B:

   Strike any key when ready

5. Remove your DOS diskette from drive A.
6. Insert your *original* diskette into drive A.

7. Insert your *backup* diskette into drive B.



8. Press any key.

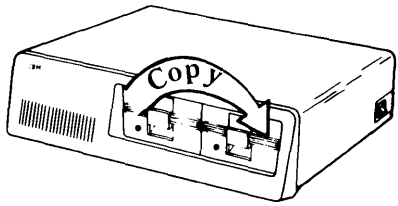
   This tells DOS you are ready and this message is displayed:

```
Copying 9 sectors per track, 1 side(s)
Formatting while copying
```

   The number 8 or 9 may appear in the first line, indicating the number of sectors per track that could be read from the original diskette.

   The **Formatting while copying** message will only appear if the diskette had not previously been formatted with the same format as the source diskette.

9. Now all information is being copied from the diskette in drive A to the diskette in drive B.



You will see one "in use" light go on and then the other.

10. When the copy has been made, you will see a message similar to this:

```
Copy complete

Copy another (Y/N)?_
```

The cursor is here.

11. Type:

    n

    and press the Enter key. The DOS prompt, A>, is displayed.

12. Remove both diskettes.

    Use a felt-tip pen to label and date the backup diskette. You may also want to write "Backup" on the label to remind yourself that this is a copy of another diskette.

Backing Up One File

Using the COPY Command

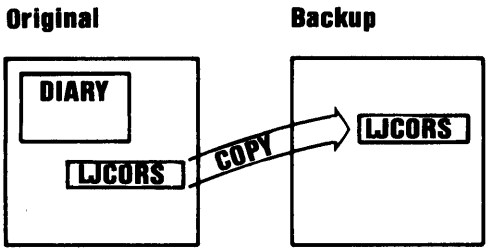
Sometimes you will find it useful to copy only *one* file. You might want to copy one file instead of a whole diskette when:

- Only one file of a whole diskette has been changed and needs to be backed up.
- You want to make extensive changes to a file, or you're not too sure of the effect of the changes. It is safer to change a copy of the file rather than the original.
- You want to build a new file based on the contents of an "old file"—similar to copying a letter and then marking up the copy to make a new letter.
- You need two copies of a file. For example, if you want to give a program you wrote to a friend, you'd probably want to give a copy of the program and not your original program.

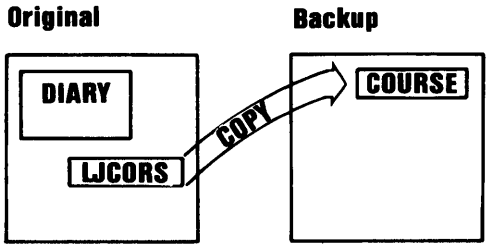
Let's use some diagrams to illustrate the various possibilities with COPY.

Assume that two of the files on the original diskette are LJCORS and DIARY, and that the file to be copied is LJCORS.

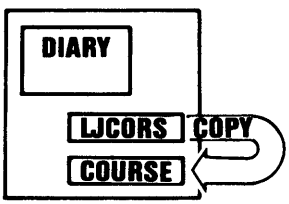
- The file (LJCORS) is copied to another diskette and the name is the same on both diskettes:



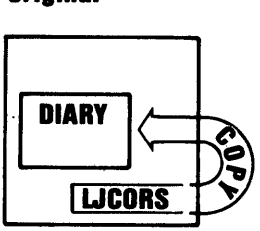
- The file (LJCORS) is copied to another diskette and the name of the copy is changed:



- The file (LJCORS) is copied onto the original diskette with a changed name:



- The file (LJCORS) is copied onto the original diskette with a name that is already being used:



Before You Begin

Decide which kind of copy you want to make. Then collect:

- The diskette that contains the file you want to copy. We'll call this the *original* diskette.

- The diskette that will contain the copy of the file when you are done.

  This may be the same diskette as your original diskette. Or it may be a different diskette, in which case we will call it your *backup* diskette.

Also decide on the name for the copied file.

Copying a File to the Same Diskette

For this example, let's assume that LETTER is the name of the file you want to copy, and that the copy's name will be MEMO.

Recall that you need a different name because every name on a diskette must be unique. You might want to check that the name you have chosen isn't already being used, unless you really mean to **replace** that file. For this example, check by typing: dir memo.

Let's do it:

1. Make sure DOS is ready and A> is displayed.

2. Insert the original diskette (the diskette with LETTER on it) into drive A.

3. Type this:

   copy letter memo

   and press Enter.

   Notice that in the command, you type the original filename first and the backup filename second. Separate the filenames with a space.

4. After a few seconds (depending on how long the file LETTER is), you will see this on the screen.

```
        1 File(s) copied
A>_
```

5. LETTER has been copied and the copy has been given the filename MEMO.

   You can use the DIR command to check, by typing this:

   dir memo

   Now press the Enter key.

6. Remove the diskette and put it away in its envelope because we are through with this example.

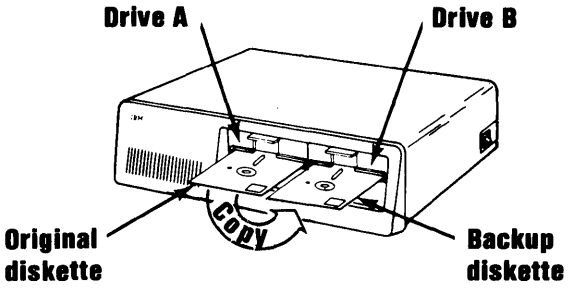
Copying a File to Another Diskette Using One Drive

Remember, with only one diskette drive, you will need to exchange diskettes while COPY is running. DOS will tell you when the change must be made by showing you a message.

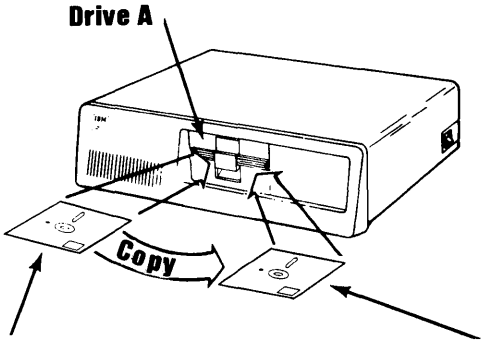
COPY's messages sound as if you had two drives instead of one. You should think of the messages as referring to *diskettes* instead of drives.

For our examples, think of your original diskette as the *drive A* diskette. Think of the backup diskette as the *drive B* diskette. (Even though you have no drive B, the command is the same for one drive or two. The computer keeps track of the real drive A and the "imaginary" drive B.)

If you had two drives



But with one drive



Let's assume that the file you want to copy is LETTER. For this example, we are going to copy LETTER to another diskette without changing its name.

1. Make sure DOS is ready and A> is displayed.

2. Insert the diskette with LETTER on it (the *original* diskette) into your drive.

3. Type this:

   copy letter b:

   and press the Enter key.

   Notice again where the spaces are.

4. This message is displayed:

   Insert diskette for drive B: and strike any key when ready

5. Remove the original diskette. (But don't put it away—you may need it soon.)

6. Insert the previously formatted backup diskette into the drive.

7. Now press a key to tell DOS that the diskettes have been switched.

8. Depending on the size of the file being copied (LETTER, in this example) and your computer's memory, you may see this message:

```
Insert diskette for drive A: and strike
any key when ready
```

If you do see the message, do this:

a. Remove the backup diskette from the drive.

b. Insert the original diskette into the drive.

c. Press a key.

d. Now go back to step 4.

Hint: It's easier to remember which diskette to insert if you think "B is for backup." Insert the backup diskette when the drive B message appears.

9. When the copy has been made, this message is displayed:

```
        1 File(s) copied
A>_
```

10. Remove the backup diskette from the drive and label it with a felt-tip pen.

11. Now you can put both diskettes away—LETTER has been copied to the second diskette.

For You to Try

Select a file on your DOS Diskette to copy to another diskette—how about COMP.COM? Keep the name the same (unless the diskette you'll put the copy on already has a file called COMP.COM).

Copying a File to Another Diskette Using Two Drives

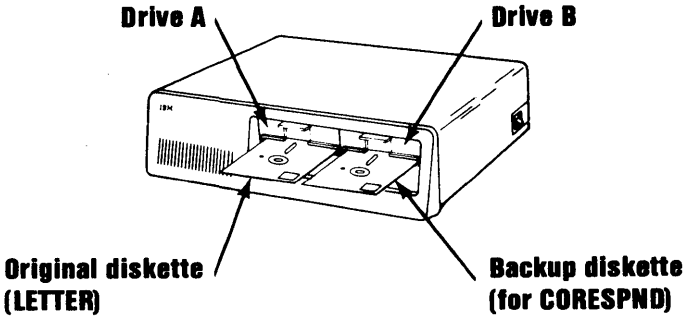
Let's assume you want to copy LETTER (again!). But this time, you want to put the copy on another diskette.

What name do you want it to have? For this example, we have picked the filename CORESPND for you. Here are the steps:

1. Make sure DOS is ready and A> is displayed.

2. Insert the diskette with LETTER on it into drive A.

3. Insert the diskette that you want the copy (CORESPND) to be written on, into drive B.



4. Type this:

copy letter B:corespnd

and press the Enter key.

Notice the spaces between **copy** and **letter** and between **letter** and **b:corespnd**. *Do not* put a space between **b:** and **corespnd**.

Also notice that you must indicate the diskette drive (**b:**) of the target diskette because it is not the same as the default drive. (Look back at "The DOS Prompt" in Chapter 2 for information about the default drive.)

5. One "in use" light will come on and then the other.

When the copy has been made, this is displayed:

```
        1 File(s) copied
A>_
```

6. You can now remove both diskettes and use a felt-tip pen to label the backup diskette.

   Now you can put both diskettes away—LETTER has been copied to the second diskette (and has been retired from use in all future examples).

   If you want the copied file to have the "same old name" on the other diskette, this is what you'd type in step 4:

   copy letter b:

   or

   copy letter b:letter

For You to Try

Practice this exercise by copying the file DISKCOPY.COM to another diskette. Change the name, if you wish. Remember, the backup diskette needs to have been formatted if it's brand new.

Backing Up More than One File

Using the COPY Command

Sometimes you will find it useful to copy more than one file at a time. For example, you want to build a *new* diskette file based on the contents of an *old* diskette file.

To backup all the files on a diskette, enter:

```
COPY *.* B:
```

The COPY command with the global filename characters *.* in the command line will copy *all* the files from the diskette in default drive A to the diskette in drive B, with no change in the filenames or in the extensions.

The filenames of the files being copied are displayed as the files are copied (refer to the "COPY Command" in Chapter 6 for additional information).

Finding Out What Is on a Diskette

Using the DIR Command

It is often handy to find out what files are on a diskette—perhaps because you need to find out how a particular filename is spelled, or because you can't recall what's on a seldom-used diskette.

The Directory command (DIR) displays a list of all the files that match a name you specify. You've seen this command before in the section "Using Global Filename Characters." Let's see how you might use it.

Before You Begin

All you need for DIR is the diskette that you want to see the directory.

To List All the Files

With One Drive

1. Make sure DOS is ready and A> is displayed.
2. Remove the DOS diskette from drive A, if it is there.
3. Insert into drive A the diskette whose directory you want to list.

4. Type:

   dir

   and press the Enter key.

5. Watch the screen.

   The first message to appear will display the volume label of the diskette (if it has one), followed by the name of the directory that is being listed. Refer to Chapter 5, "Using Tree-Structured Directories" for more information. Then the files on the diskette are listed.

   The screen displays the *filename*, the *extension*, the *size* of the file (in bytes), the date and the time that information was last written in the file. One line is displayed for each file on the diskette.

   After the files have been listed, DIR displays the amount of free space left on the diskette (in bytes).

   Remember, if the information is moving too fast for you to read it, press and hold the Ctrl key and then press the Num Lock key. Then press another key when you're done reading to start displaying again. It is also a good idea to print a copy of a diskette's directory (Ctrl + PrtSc) to keep with the diskette.

6. When all the files have been displayed, the DOS prompt appears:

   A>

With Two Drives

If you have two diskette drives, you can keep the DOS diskette in drive A, if it is there, and insert your diskette into drive B. Then at step 4, type this:

```
dir b:
```

and press the Enter key.

You have to tell DOS where to find the diskette—in this case the one in drive B. Otherwise, DOS assumes you mean the default drive diskette if you type only the command (**dir**) without a drive specifier.

To List One File

With One Drive

For this example, let's assume that you think you have a file named FACT&FIG on a diskette. Here's how you make sure:

1. With DOS ready (A> is displayed), insert the diskette into drive A.

2. Type:

   ```
   dir fact&fig
   ```

   and press the Enter key.

3. If FACT&FIG is indeed on the diskette, its filename (FACT&FIG), its extension (in this case, none), its size, and its update date and time are shown.

   If the diskette does not have a file named FACT&FIG, then after a second or two the message:

```
File not found
A>_
```

   is displayed.

With Two Drives

You can insert the diskette with the file into drive B and type this for step 2:

```
dir b:fact&fig
```

Now press the Enter key.

See, the only difference is that you have to tell DOS where (what drive) the file is.

Displaying What Is in a File

Using the TYPE Command

The TYPE command lets you "look into" a file; that is, it displays the contents of a file on the screen.

As the example will show, you can also print the contents if you have a printer.

Before You Begin

Locate the diskette that has the file you want to display. You also need to know the exact name of the file (use DIR again).

Here's How You Do It

For this example, assume that the file you want to display is ITEMS.DAT.

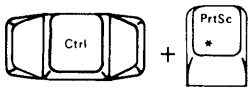
1. Make sure DOS is ready for a command; A> is displayed.

2. Insert the diskette with ITEMS.DAT into drive A.

3. If you have a printer:

   a. Make sure that the printer is turned on, is online, and has paper in it.

   b. Press and hold the Ctrl key and then press the PrtSc key.



   c. Release both keys.

   d. Check to see that "echo to the printer" has been turned on by pressing the Enter key.

   e. Both the screen and the printer should show the prompt, A>.

   f. If the printer didn't print A>, do steps 3b through 3e again.

4. Now type:

   type items.dat

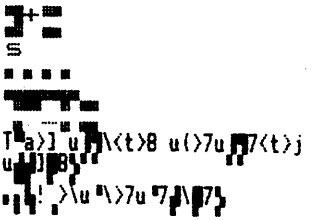
   and press the Enter key.

5. The command you just typed and contents of ITEMS.DAT will be displayed and printed.

6. When you see the DOS prompt, A>, you can remove the diskette and put it away.

7. Stop the printer from printing what is displayed on the screen by pressing and holding the Ctrl key and then pressing the PrtSc key.

   Some files contain information that you won't be able to read. These are programs or certain data files that the computer can read—but we can't. The printer information might look similar to this:



   And the information on the screen looks even stranger!

Changing a File's Name

Using the RENAME Command

The RENAME command lets you change a file's name—either its filename, its extension, or both.

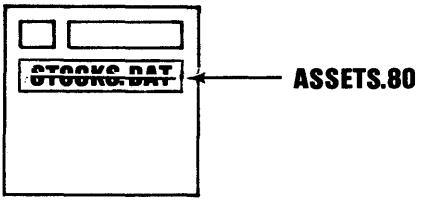
This may be useful if you've found that you simply cannot type a name the way you've spelled it. More often, the reason for changing is that you just want to refer to a file with a different name.

Before You Begin

Have on hand the diskette with the file that you want to rename. You also need to know its exact "old" filename and extension, if it has one. (Remember, you can use the DIR command to find out the filename and extension.)

With One Drive

For this example, let's assume that a diskette has a file named STOCKS.DAT on it, but that we have decided that ASSETS.80 is a more descriptive name for this information. Also assume that we have checked this diskette's directory and found that this diskette does *not* already have a file named ASSETS.80 on it.



Here's how to change the name:

1. Make sure DOS is ready and A> is displayed.

2. Insert the diskette with STOCKS.DAT on it into drive A.

3. Type this:

   rename stocks.dat assets.80

   Check your typing, and then press Enter.

   Notice that the "old" name is first, then a space, and then the "new" name.

4. In a moment, DOS displays the prompt, A>, and the file has been renamed.

5. You can check to see that the file has really been renamed by typing:

   dir stocks.dat

This should be displayed:

```
File not found
A>_
```

Then type:

```
dir assets.80
```

Now the information about that file should be displayed.

6. Remove your diskette and put it back in its envelope.

   Be careful about using global filename characters in the "old" file specification. You may get *unexpected* results. Changing back to the names you want may be more tedious than renaming the files one by one in the first place.

With Two Drives

If you have two diskette drives and already have another diskette in drive A, then insert the diskette with STOCKS.DAT on it into drive B, then type:

```
b:
```

This changes the default drive to drive B. Now press Enter.

Then repeating step 3, type:

```
rename stocks.dat assets.80
```

Now press Enter.

Repeat step 5 also, thus:

```
dir stocks.dat
```

and

```
dir assets.80
```

To switch the default drive back to drive A, type:

```
a:
```

then press Enter.

For You to Try

Try using the RENAME command on one of the files that you copied (in a previous "practice" exercise).

Removing a File from a Diskette

Using the ERASE Command

One of the housekeeping chores associated with diskettes is to remove files that you no longer need from diskettes. The ERASE command does this for you.

Removing old files makes room for new information on a diskette. It can eliminate a potential source of confusion too–you are less likely to use an old version of a program or an old data file for processing.

But *plan ahead* and *check your typing* when you use ERASE.

After a file is erased, the data is gone. Unless you have made a backup, the only way to recreate a file that has been erased is by repeating the steps that you took to create it in the first place.

Before You Begin

You need the diskette with the file that you want to erase. Again, you need to know the exact filename and extension of that file (use DIR if you need to).

And make sure you really will *not* need this file.

With One Drive

For this example, assume that the file to be removed is ASSETS.80 (the one we just used in the RENAME example).

1. Make sure DOS is ready with A> displayed.

2. Insert the diskette with the file you no longer need into drive A.

3. Type this:

   erase assets.80

   and check your typing. Now press Enter.

4. In a moment, DOS shows you the prompt, A>. The ASSETS.80 file has been removed from the diskette.

5. To assure that the file has been erased, try the DIR command again, like this:

   dir assets.80

   and press Enter.

   The message **File not found** and the DOS prompt, A>, should appear.

6. Remove the diskette and put it back into its envelope.

With Two Drives

If you can use drive B, insert the diskette into drive B, and type this for step 3:

```
erase b:assets.80
```

and this for step 5:

```
dir b:assets.80
```

Again, the only difference is that you have to remember to type the drive specifier (**b:**) because it is not the default drive.

Global Filename Characters

BE CAREFUL WITH GLOBAL FILENAME CHARACTERS.

We recommend that you DO NOT USE global filename characters in the file specification of the ERASE command until you are familiar with exactly what they will do.

For example, typing

```
erase *.*
```

Removes *all* the files from the diskette (except the DOS system files, if present).

If the files that are removed by using global characters are not the ones that you expected, you can recover this information only if you have previously made a backup diskette.

Shifting the Display on the Screen

Using the MODE Command

If you have the Color/Graphics Monitor Adapter, then you may find that the first two or three characters on a line of the display do not show up on the screen. If your display does not have a horizontal adjustment control, you can use the DOS MODE command to shift the displayed lines to the right.

Before You Begin

You need only your DOS diskette to do this.

Shift Right

1. Make sure that DOS is ready and A> is displayed (although you probably can't see it).

2. Type this:

   mode ,r,t

   Check your typing—did you put the space and then the comma before the r?

   When it's okay, press the Enter key.

3. Now a test pattern will appear on your screen.

```
01234567890123456789012345678901234567890123456789

Do you see the leftmost 0? (Y/N)
```

4. If your screen looks okay, type this:

   y

   and press the Enter key.

   If you want your screen moved over to the right a little more still, type this:

   n

   and press the Enter key.

Helps and Hints

We are ending this chapter with a few hints—maybe they will save you some trouble or help you as you use your IBM Personal Computer.

- Make backups regularly.

  (We have repeated this already, but since this is our last chance, we are saying it one more time.)

- Sometimes, commands do not work as you expected because a file specification was typed incorrectly.

  — Check your typing.
  — Do you have the correct diskette in the drive?
  — Check the directory of the diskette.
  — Has the correct diskette drive been specified or is it being assumed? Has the colon been included?
  — Has the filename been spelled correctly?
  — Have you forgotten to use the extension? (In the case of BASIC program files, for example, it is easy to forget the .BAS that BASIC uses for an extension.)

- If a command still doesn't work, refer to the page in Chapter 6 that fully describes the command.

- Be careful with global filename characters, especially in the ERASE command. If you insist on using them:

  — Use the DIR command first with the global filename specification, to test the results.
  — Check to see that these are **really** the files you want to delete.
  — Then, go ahead.

- Print a directory frequently (if you have a printer) and store the listing with the diskette. The labels on the diskettes are usually too small to hold everything you want to write on them.

- All commands (except DISKCOPY and DISKCOMP) that use files will work on both diskettes and fixed disks.

- The date and time shown with each directory entry are the date and time of the last addition or change to that file. The date and time are not changed during a COPY or a DISKCOPY.

  At first glance,

  ```
  diskcopy a: b:
  ```

  and

  ```
  copy  a:*.* b:
  ```

  may appear to have the same purpose—copying an entire diskette. They do, only when copying to a diskette with no files on it.

With COPY, if files already exist on the backup diskette, they will either be replaced (if files being copied have the same name) or left alone. This is because COPY goes through the original diskette, copying each file, one at a time. COPY does not disturb old files on the backup diskette as long as their names aren't the same as files being copied.

DISKCOPY, however, makes a "carbon copy" of the original diskette, wiping out all old files on the backup diskette during the COPY process.

Summary

We've covered a lot in this chapter. We have told you about:

- Making a backup copy of your DOS diskette
- Diskettes and terms that are used with them
- Finding out what files are on the DOS diskette
- Making a backup copy of your diskettes

We have provided step-by-step procedures for using these DOS commands:

- FORMAT
- DIR
- DISKCOPY
- COPY
- TYPE
- RENAME
- ERASE
- MODE

We hope you're well on your way to feeling comfortable with your computer and DOS. No matter how much we've covered, there's always more—in particular, we haven't given you the shortcuts that you'll start to pick up as you continue through the *Disk Operating System* book and use the commands.

Notes:

Chapter 4. Preparing Your Fixed Disk

Contents

Notes:

Introduction

If your IBM Personal Computer has a fixed disk, there are several facts you need to know, and several steps to take before DOS is able to use it. If you try to use your fixed disk before you take the following steps, you will get the error message:

Invalid drive specification

The first section in this chapter, "Fixed Disk Drive Letters" describes how drive letters are used with the fixed disk. The remaining sections describe how to prepare your fixed disk for DOS.

> **Note:** If your fixed disk is in an expansion unit (not in the system unit), the expansion unit must be turned on *before* you turn on the system unit.

A fixed disk can be divided up into separate areas called partitions. There can be from one to four partitions on a fixed disk. These partitions can be different sizes. Each partition is set up through a Fixed Disk Setup Program provided by the operating system that will use it.

You will use the DOS Fixed Disk Setup Program to set up the DOS partition. Fixed disk drives are referred to in the same way you refer to diskette drives, using the fixed disk drive specifier when you want to read from or write to the fixed disk.

If DOS is the only operating system that you intend to use with the fixed disk, you will want to assign all of the fixed disk space for use with DOS. Follow the steps in "Preparing Your Fixed Disk" to assign all of the fixed disk space to the DOS partition.

If you do intend to use part of the fixed disk with other operating systems, then you will need to divide up the available fixed disk space among them. Follow the steps in "Partitioning Your Fixed Disk" to assign a specific amount of disk space to DOS.

If you are not sure whether you will be using any other operating system with the fixed disk, you should go ahead and assign all of the fixed disk space to DOS as described in "Preparing Your Fixed Disk". If you decide later to use another operating system, you can use the BACKUP command to backup your files in your DOS partition, reassign the DOS partition, and then use the RESTORE command to restore your files from diskettes. After you have followed the instructions in "Preparing Your Fixed Disk" or "Partitioning Your Fixed Disk," you will need to follow the instructions in "Setting Up the DOS Partition" in order to make the partition you have created useable by DOS.

The menus and screens that make up the Fixed Disk Setup Program have been designed to make it easy for you to set up your fixed disk. When the Fixed Disk Setup Program asks you to enter something, it displays a default answer for you. If that is the answer you want, you need merely press the Enter key. If you want to enter something else, simply type in the entry you want and then press the Enter key.

Fixed Disk Drive Letters

You already know that if you have two diskette drives on your system, they are known to DOS as A and B. If you have only one diskette drive on your system, there are still two diskette drives (A and B) known to DOS. But in this case, they are simulated on the same physical drive. (If you are not yet familiar with this concept, please refer to "Single Diskette-Drive Systems" in Chapter 1 before you continue.)

When DOS starts, it first assigns letters to all of the diskette drives it knows about, then assigns the following letters to your fixed disks. For example, if you have one or two diskette drives, and one fixed disk, the letters A and B apply to the diskettes, and your fixed disk is known to DOS as C (if you had a second fixed disk, it would become D).

Preparing Your Fixed Disk

If DOS is the only operating system that you intend on using with your fixed disk, follow the instructions in this section. All of the fixed disk space will be assigned for use with DOS.

If you intend to use part of the fixed disk with another operating system, then you should go to "Partitioning Your Fixed Disk" in this chapter.

In order to prepare the fixed disk for use with DOS, you must first use the FDISK command as follows:

1. With your DOS diskette in drive A and the DOS prompt (A>) on the screen, type:

   FDISK

   and press Enter. The following screen appears:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

FDISK Options

Current Fixed Disk Drive: 1

Choose one of the following:

     1.  Create DOS Partition
     2.  Change Active Partition
     3.  Delete DOS Partition
     4.  Display Partition Data
     5.  Select Next Fixed Disk Drive

Enter choice: [1]
```

The current fixed disk drive and option 5 will only be shown if your system has more than one fixed disk drive. If you want to set up the DOS partition on the next drive, type:

5

and press Enter. The Fixed Disk Setup Program will allow you to create an active partition on a fixed disk other than the first fixed disk. However, DOS can only be started from the first fixed disk (refer to "Changing the Active Partition" in this chapter to change the partition status). You will see the drive number change on the screen after you press Enter.

2. Type:

1

and press Enter to set up the fixed disk for use with DOS. If the fixed disk has not already been set up for DOS or another operating system, then the following screen appears:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

Create DOS Partition

Current Fixed Disk Drive: 1


Do you wish to use the entire fixed disk
for DOS (Y/N).....? [Y]
```

If your fixed disk has already been set up, then you will see a different screen that shows how the fixed disk partitions have been assigned, and you will get a different prompt. If this happens, you should follow the steps in "Partitioning Your Fixed Disk" in this chapter.

3. You should press Enter since you want to use the entire fixed disk for DOS. The Fixed Disk Setup Program will then assign the fixed disk to DOS, and display the following message:

   Insert DOS diskette in drive A:
   Press any key when ready . . .

4. You must now restart DOS so that it will recognize your fixed disk and assign a drive letter to it. With your DOS diskette in drive A, press any key to restart DOS.

Your fixed disk has now been set up with a DOS partition. But before DOS can use it, DOS needs to create a directory and other information in the partition. To do this, follow the instructions in "Setting Up the DOS Partition."

Setting Up the DOS Partition

The DOS fixed disk partition must be formatted by the DOS FORMAT command before it can be used. You should only follow these instructions if the DOS partition has been created, but has not already been formatted and used to store data. This is because any data in the partition will be destroyed by the format operation.

1. Make sure your DOS diskette is in drive A and the DOS prompt (A>) is on the screen.

   If DOS is to be started from the fixed disk, enter:

   FORMAT d:/S/V

   If the partition is not to contain a copy of DOS (not to be automatically started), enter:

   FORMAT d:/V

   In either case, substitute the correct fixed disk drive letter for the *d* in the command (for example, if you have 1 or 2 diskette drives, you would enter C, as shown in the following screens). This prompt now appears:

   Press any key to begin formatting drive C:

2. Press any key. The red light on your fixed disk drive will light up, and the message:

   Formatting...

   appears on the screen. Do not be alarmed if several minutes go by before you see any more messages. DOS is checking the data in every location in the DOS partition and it takes several minutes. You will see the message:

   Format complete

   and, if you used /S in your FORMAT command, you also see the message:

   System transferred

This tells you that a copy of DOS has been placed on the fixed disk.

Then the following message appears:

Volume label (11 characters, ENTER for none)?

3. Enter a 1-11 character *volume label* (for example, MYFIXEDDISK) that is used to identify the fixed disk when DIR and CHKDSK displays information. If you do not want to label the fixed disk, just press Enter; however, please note that you cannot add a volume label later, so we recommend entering one now.

   FORMAT then displays the disk space statistics and the DOS prompt:

   A>

   Your fixed disk is now completely usable by DOS.

   If you have placed a copy of DOS on the fixed disk, we recommend that you do the following two steps:

4. With your DOS diskette still in drive A, enter:

   COPY *.* *d:*

   Remember to use a correct fixed disk letter for your system. This copies all the programs on the DOS diskette to your fixed disk. Once these programs are copied, all DOS commands can be run from the fixed disk and your DOS diskette can be stored away in a safe place.

Also, remember to copy the programs from your DOS Supplemental Program diskette in the same manner, if you will be using these programs.

5. Remove the DOS diskette from drive A (leave the diskette drive door open) and press the Ctrl, Alt, and Del keys simultaneously (Systems Reset). If you have correctly followed the steps above, and a copy of DOS was stored in the DOS partition, DOS will start from the fixed disk and you will be asked to enter the date and time.

   Note: Drive A must be empty (no diskette or the diskette drive door open) in order for this to work correctly. This is because the computer will first try to load an operating system from drive A. If a diskette cannot be read from drive A, then (and only then), the computer will try to load an operating system from the first fixed disk on the computer.

When you have entered the date and time, you will notice that the DOS prompt (A>) has now changed. Instead of the letter A>, you will see the drive letter of your fixed disk. DOS remembers which drive it was started from, and makes that drive the default drive.

If you have followed the instructions above, your fixed disk is now completely initialized.

Partitioning Your Fixed Disk

In order that more than one operating system can use the fixed disk, the fixed disk must be divided into separate areas called partitions. There can be from one to four partitions on the fixed disk. These partitions can be different sizes and can be set up in any order. You can specify which partition the system will get control of when you start or restart your computer. An operating system can only access one partition. You cannot transfer data directly from one partition to another.

Each operating system that supports the fixed disk provides a program to allow you to create a partition for use under that system. If you try to read from, or write to, the fixed disk using a system that has no partition assigned to it, you will get an error message.

The DOS Fixed Disk Setup Program can only be used to create or delete the DOS partition. A partition set up for another operating system can only be created or deleted from that operating system.

You can set up one partition for use under DOS at the location and size you choose. You can also delete the DOS partition if, for example, you want to create it again at a different size or location on the fixed disk. The following functions are supported by the DOS Fixed Disk Setup Program:

- Create the DOS partition.
- Change the active partition (the one that will be started when the system is restarted).

- Delete the DOS partition.
- Display fixed disk partition data.
- Select next fixed disk drive.

These functions are described below in separate sections. In order to get access to them, you need to start the Fixed Disk Setup Program as follows:

1. To start, type:

   ***d*:FDISK**

   and then press Enter. Where *d*: is the drive where the FDISK.COM program resides.

2. If it is on the default drive, just type:

   FDISK

   and press Enter. You will then see the following screen:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

FDISK Options

Current Fixed Disk Drive: 1


Choose one of the following:

      1.  Create DOS Partition
      2.  Change Active Partition
      3.  Delete DOS Partition
      4.  Display Partition Data
      5.  Select Next Fixed Disk Drive

Enter choice: [1]
```

The current fixed disk drive and option number 5 will be shown only if your system has more than one fixed disk drive. Option 5 selects the next fixed disk drive.

The Fixed Disk Setup Program will allow you to create an active partition on a fixed disk other than the first fixed disk. However, DOS can only be started from the first fixed disk. The partition's status is shown as **A** for active and **N** for not active (refer to "Changing the Active Partition" in this chapter to change the partition status).

Type the number of the option you want and press the Enter key. Note that option 1 is the default and will automatically be selected if you don't type an option number before pressing Enter. Proceed to the section below that describes the option you selected.

Creating the DOS Partition (Option 1)

You can use this option to create the DOS partition. In order to do so, you need to determine where it should be located and how large it should be. If there is already a partition assigned to DOS, you will see an error message.

A fixed disk is divided into parts called cylinders. The number of cylinders and their sizes can vary depending on the fixed disk. A 10-megabyte disk contains 305 cylinders and each cylinder contains 34,816 bytes or characters of information. If you wanted to assign the whole fixed disk for use under DOS, you would specify the size as 305 cylinders and the starting cylinder number as 000.

The screen you see depends on whether the fixed disk has any partitions. If it has been initialized, go to step 2. If not, the following screen appears:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

Create DOS Partition

Current Fixed Disk Drive: 1


Do you wish to use the entire fixed
disk for DOS (Y/N).....? [Y]
```

1. If DOS is the only operating system that you intend on using with the fixed disk, then you should follow the instructions in "Preparing Your Fixed Disk" in this chapter. Otherwise, type:

   n

   and then press the Enter key. The following message appears:

   Total fixed disk space is xxxx cylinders.
   Maximum available space is xxxx.
   cylinders at cylinder xxxx.

   These lines show the total number of cylinders on your fixed disk. Proceed to step 3.

2. You will see a screen similar to the following if your fixed disk has already been set up:

IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

Create DOS Partition

Current Fixed Disk Drive: 1

Partition	Status	Type	Start	End	Size
1	N	non-DOS	000	049	50
2	A	non-DOS	050	099	50
3	N	non-DOS	250	304	55

Total fixed disk space is 305 cyls.
Max avail space is 150 cyls at cyl 100.

The "Create DOS Partition" screen shows a sample fixed disk with 3 partitions. Note that this is not necessarily a recommended setup. It is shown only as an example.

The line with the current fixed disk drive will only appear if you have more than one fixed disk drive.

There will be one line shown for each assigned partition.

The Partition column shows the relative number of the partition (in the order it appears on the fixed disk).

The status column shows which partition's system gets control when the system unit is started from the fixed disk. That partition's status is shown as **A** (for active), the others are shown as **N** (for not active).

The Type column shows which partition, if any, is the DOS partition.

The Start and End columns show the starting and ending cylinder numbers for a partition and the Size column shows its size in cylinders.

The next line shows you the total amount of space on the fixed disk, and the line after that shows you the size of the largest available space that you could use for a partition and where it is located on the fixed disk.

3. The following prompt appears:

Enter partition size.....: [*xxxx*]

The partition size entry defaults to the largest available space on the fixed disk. If you want your DOS partition to use the largest available space, simply press the Enter key. Otherwise, type in the size you want (in cylinders) and press the Enter key. The next prompt is:

Enter starting cylinder number..: [*xxxx*]

4. The starting cylinder number default depends on the partition size you specified above. It is the first cylinder of the largest space on the fixed disk large enough for the partition. If you want the DOS partition to be located there, press the Enter key. Otherwise, type in the starting cylinder number you prefer and press the Enter key. The cursor is placed at the bottom of the screen and you see this message:

Press Esc to return to FDISK Option [ ]

Note that the lines on the screen change to show the new active partition. The DOS partition has now been created. With your DOS diskette in drive A, press the Ctrl, Alt, and Del keys simultaneously (System Reset).

If you need the partition you just created to be *active* (startable), follow the steps in "Changing Active Partition (Option 2)."

Your DOS partition has been created but you still need to follow the instructions in "Setting Up the DOS Partition" in this chapter before you can use the DOS partition.

Changing the Active Partition (Option 2)

Select this option when you want to start a different operating system in another partition. You will see a screen similar to the following:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

Change Active Partition

Current Fixed Disk Drive: 1

Press Esc to return to Utility Options

Partition    Status    Type      Start    End    Size
    1          N      non-DOS     000     049     50
    2          N      non-DOS     050     149     100
    3          A       DOS        150     304     155

Total disk space is xxxx cylinders.

Enter the number of the partition you want
to make active.....: [ ]
```

1. Enter the number of the partition whose operating system you want to get control when the system is started from the fixed disk. The following message appears:

   Press Esc to return to FDISK Options [ ]

   Note that the lines on the screen change to show the new active partition.

2. Press the Esc key to return to the FDISK options menu and press it again to return to DOS.

If you want to start the operating system in the partition you just made active, perform the following steps:

a. Open the diskette drive A door.

b. Press and hold Ctrl and Alt, and then press Del.

The operating system in the active partition should then start.

Deleting the DOS Partition (Option 3)

Note: This option destroys all data in the DOS partition so make sure you have backed up all of your files before you proceed.

1. You will need to insert a DOS diskette and restart the system from diskette drive A if you want to continue processing under DOS.

   If you want to start a system in another fixed disk partition, you should change the active partition to that partition number before you delete the DOS partition.

You will see a screen similar to the following:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

Delete DOS Partition

Current Fixed Disk Drive: 1


Partition    Status    Type      Start   End    Size
    1          N      non-DOS    000    049     50
    2          N      non-DOS    050    099     50
    3          N      DOS        100    249     150
    4          A      non-DOS    250    304     55

Total fixed disk space is xxxx cylinders


Warning! All data in the DOS partition
will be DESTROYED. Do you wish to
continue.....? [ ]
```

2. If you have backed up all of your files and are ready to continue, type Y and press Enter. If you decide to cancel the operation, press either the Enter key or the Esc key to return to the FDISK options menu.

   If you type Y and press Enter, the partition information displayed on the screen is updated, and the following message appears:

   Press Esc to return to FDISK Options [ ]

   The DOS partition has now been deleted. You will need to start another system from the fixed disk or restart DOS from a diskette to proceed.

Displaying Partition Data (Option 4)

You can use this option to display fixed disk status information. Your screen appears similar to the following:

```
IBM Personal Computer
Fixed Disk Setup Program Version 1.00
(C)Copyright IBM Corp. 1983

Display Partition Information

Current Fixed Disk Drive: 1


Partition  Status  Type      Start  End  Size
    1        A     DOS       000    199  200
    2        N     non-DOS   200    304  105



Total fixed disk space is xxxx cylinders
```

The line with the current fixed disk drive appears only if you have more than one fixed disk drive.

One line is shown for each assigned partition.

The Partition column shows the relative number of the partition (in the order it appears on the fixed disk).

The Status column shows which partition's system gets control when the system unit is started from the fixed disk. That partition's status is shown as A (for active), the others are shown as N (non-active).

The Type column shows which partition, if any, is the DOS partition.

The Start and End columns show the starting and ending cylinder numbers for a partition, and the Size column shows its size in cylinders.

The next line shows you the total amount of space on the fixed disk.

Press the Esc key when you are ready to return to the FDISK options menu.

Selecting Next Fixed Disk Drive (Option 5)

Select this option when you want to use the DOS Fixed Disk Setup Program with the next fixed disk drive.

After you have entered the option, you see the current fixed disk drive number change on the FDISK options menu.

This option is available only if your system has more than one fixed disk drive.

Notes:

Chapter 5. Using Tree-Structured Directories

Contents

Notes:

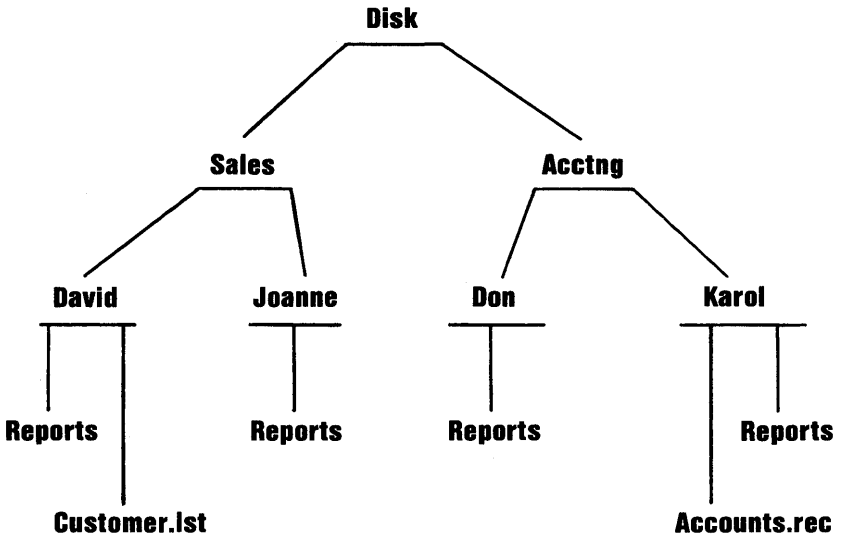
Introduction

Prior to Version 2.00, DOS used a simple directory structure that was adequate for managing files on diskettes. Each diskette contained a single directory that could hold a maximum of 64 or 112 files, depending on whether the diskette was single or dual sided.

With the added support for fixed disks in DOS Version 2.00, however, a single fixed disk can hold literally thousands of files. Keeping a large number of files in one directory becomes inefficient for both you and DOS (the larger a directory is, the longer it can take DOS to search for a file).

DOS Version 2.00 gives you the ability to better organize your disk by placing groups of related files in their own directories—all on the same disk (fixed disk or diskette).

For example, let's assume that the XYZ company has two departments (sales and accounting) that share an IBM Personal Computer. All of the company's files are kept on the computer's fixed disk. The logical organization of the file categories could be viewed like this:



With DOS Version 2.00, it is possible to create a directory structure that matches the file organization. With this ability, all of DAVID's report files can be grouped together in a single directory (called REPORTS), separated from all the other files on the disk. Likewise, all of the accounts receivable files can be in a unique directory, and so on.

Directory Types

As in previous versions of DOS, a single directory is created on each disk when you FORMAT it. That directory is called the root directory, or system directory.

A root directory on diskette can hold either 64 or 112 files—the maximum number of files in a fixed disk root directory depends on the size of the DOS partition on the disk.

In addition to containing the names of files, the root directory can also contain the names of other directories; and these in turn can contain the names of other files and directories; and so on.

Unlike the root directory, these other directories *called sub-directories* are actually files, and are therefore not restricted in size—they can contain any number of files and sub-directories, limited only by the amount of available space on the disk.

The sub-directory names are in the same format as filenames—a name of 1-to-8 characters optionally followed by a period and an extension of 1-to-3 characters. All characters that are valid for a filename are also valid for a directory name. Each directory can contain file and directory names that also appear in other directories. In other words, two or more files or directories can have the same name, as long as they are defined in separate directories.

The Current Directory

Just as DOS remembers a default drive, it can also remember a default directory for each drive on your system. This is called the *current directory*, and is the directory that DOS will search if you enter a filename without telling DOS which directory the file is in. You can change the current directory or find out what your current directory is for any drive by issuing the CHDIR command (described in Chapter 6). When DOS is started, it will automatically use the root directory as the current directory for each drive until you issue a CHDIR command.

Specifying the Path To a File

When you want DOS to create or search for a file, DOS must know three things—the drive, the name of the file, and the name of the directory containing the file.

If the file is in the current directory, you don't need to specify a directory—DOS will automatically look in the current directory.

But if the file is not in the current directory, you must supply DOS with the *path* of directory names leading to the desired directory. The path you specify can be either the path of names starting with the root directory, or the path from the current directory.

The *path* consists of a series of directory names separated by backslashes (\). If a filename is included, it must also be separated from the last directory name by a backslash.

If a path begins with a backslash, DOS starts its search from the root directory; otherwise, the search begins at the current directory.

For example, if the current directory is DAVID, and you want to find file ANNUAL.FIG in DAVID's REPORTS directory, you can specify it in either of these ways:

```
\SALES\DAVID\REPORTS\ANNUAL.FIG
```

or

```
REPORTS\ANNUAL.FIG
```

In the first case, the full path from the root directory (leading backslash) was specified. In the second case (no leading backslash), the path from the current directory was given.

Each sub-directory contains two special entries—you'll see them listed when you use the DIR command to list a sub-directory. The first contains a single period instead of a filename—it identifies this "file" as a sub-directory. The second entry contains two periods instead of a filename, and is used by DOS to locate the higher level directory that defines this directory, the *parent* of this directory. For example, in the file organization illustration shown on the first page of this chapter, the parent of the directory named JOANNE is the one named SALES.

This second special entry can be quite useful when you specify a path to DOS, because entering two periods is a shorthand way of telling DOS to *back up* one directory level. For example, if the current directory is DAVID, and you want to find file SUMMARY in JOANNE's REPORTS directory, you can specify it in either of these ways:

```
\SALES\JOANNE\REPORTS\SUMMARY
```

or

```
..\JOANNE\REPORTS\SUMMARY
```

The second case causes DOS to backup one level from the current directory (to the current directory's parent), and to continue the path from there. The double period can be used more than once in a path—it simply causes DOS to back up one level each time it is specified.

You can specify which drive to use by including a drive specifier ahead of the path and filename string. For example:

```
B:\LEVEL1\MYFILE
```

Notice that when defining a path, the drive is specified ahead of the path, rather than the filename.

Nearly all of the DOS commands that accept filenames also accept path names. For example, if you enter:

```
DIR \ACCTNG\KAROL\REPORTS
```

all of the files in KAROL's REPORTS directory are listed. Similarly, if you enter:

```
DEL \ACCTNG\KAROL\REPORTS
```

DOS assumes you want to erase all the files in KAROL's REPORT directory.

In all cases, to refer to a specific file, simply add the filename to the end of the path (separated from the path by a backslash, of course).

To refer to the root directory (if the root is not the current directory), enter one backslash. For example, if the current directory is DAVID, issuing:

```
DIR \
```

will list all the files in the root directory.

You can create as many sub-directories as you wish. However, you should ensure that the longest path you create (from the root to the last directory in a single path) can be expressed in 63 characters or less.

Directory Commands

The following commands are new in DOS Version 2.00, and are included to help you create and manage your directory structure. A brief explanation of each is presented here—for more detailed information, please refer to the individual command descriptions in Chapter 6.

System programmers and application developers should also refer to the technical appendixes (in Section 2) for descriptions of the DOS function calls.

Creating a Sub-Directory

The MKDIR (MD) command is used to create new directories. Be sure to include the appropriate drive and path, ending with the name of the new directory you want created. Also, be sure that the full path from the root to the new directory name is 63 characters or less.

Deleting a Directory

Directories can be deleted (removed) only with the RMDIR (RD) command. They cannot be deleted with the ERASE or DEL commands. A directory can be removed only if it is empty—that is, it has no files or sub-directories other than the two special entries shown as . and .. in the DIR command display. The last directory name in the specified path is removed—only one directory at a time can be deleted. The root directory and the current directory cannot be deleted. Entering CHDIR or CD with no parameters (or only a drive specification) causes DOS to display the current directory for the drive.

Displaying and Changing the Current Directory

The CHDIR (CD) command is used to tell DOS which directory path it should "remember" as the current directory. Enter just a backslash for the root, or a full path for any other directory. The current directory is where DOS will look to find files whose names are entered without path specifiers.

Displaying the Directory Structure

The TREE command will produce a report describing the entire directory structure of a disk. Included in the report are all directory paths and, optionally, the names of all files in each sub-directory.

Where DOS Looks for Commands and Batch Files

When you enter a command, DOS searches the current directory for it (if the command is not built-in). The PATH command allows you to specify a series of additional paths that DOS can search if it does not find the command in the current directory. The PATH command is described in Chapter 6.

Notes:

Chapter 6. DOS Commands

Contents

Notes:

Introduction

This chapter explains how to use the DOS commands. You can use DOS commands to:

- Compare, copy, display, erase, rename files, and format fixed disks and diskettes.
- Execute system programs, such as EDLIN and DEBUG, plus your own programs.
- Set various printer and screen options.
- Request DOS to pause.
- Transfer DOS to another diskette.
- Set options for the Asynchronous Communications Adapter.
- Cause printer output to be directed to the Asynchronous Communications Adapter.
- Recover a specific file from a damaged disk, or recover the entire disk or diskette.
- Print the contents of a graphics display screen on a printer.
- Print files on the printer while the system is doing other work.
- Backup and restore files on a fixed disk.

Types of DOS Commands

There are two types of DOS commands.

- Internal
- External

Internal commands execute immediately because they are built-in to DOS.

External commands reside on disk as program files; therefore, they must be read from disk before they execute. This means that the disk containing the command must already be in a drive, or DOS is unable to find the command. For example, if you entered the command:

```
B:GRAPHICS
```

you must be sure that the diskette containing GRAPHICS.COM is in drive B. If you entered:

```
GRAPHICS
```

then DOS will look on the default drive (the one in your system prompt) for the GRAPHICS command.

Any file with a filename extension of .COM or .EXE is considered an external command. This allows you to develop your own unique commands and add them to the system. (For example, programs such as FORMAT.COM and COMP.COM are external commands.)

When you select an external command, do not include the filename extension.

For Application Developers: DOS Version 2.00 has a provision that allows you to cause execution of DOS commands from within your application. Please refer to Appendix F for details.

Format Notation

We will use the following notation to indicate how the DOS commands should be entered:

- You must enter any words shown in capital letters. These words are called *keywords* and must be entered exactly as shown. You can, however, enter keywords in any combination of uppercase and lowercase letters. DOS automatically converts keywords to uppercase.

- You must supply any items shown in lowercase *italic* letters. For example, you should enter the name of *your* file when *filename* is shown in the format.

- Items in square brackets ([ ]) are optional. If you want to include optional information, you do not need to type the brackets, only the information inside the brackets.

- Items separated by a bar (|) mean that you can enter one of the separated items. For example:

  ON | OFF

  Means you can enter ON *or* OFF, but not both.

- An ellipsis (...) indicates that you can repeat an item as many times as you want.

- You must include all punctuation (except square brackets and vertical bars) such as commas, equal signs, question marks, colons, slashes, or backslashes where shown.

DOS Command Parameters

Parameters are items that you can include in your DOS command statements. They are used to specify additional information to the system. Some parameters are required in your commands, others are optional. If you do not include some parameters, the system provides a default value. Default values that the system provides are discussed in the detailed descriptions of the DOS commands.

Use the following parameters in your DOS command statements:

Parameter	Definition
d:	Denotes when you should specify a drive. Enter a drive letter followed by a colon to specify the drive. For example, A represents the first drive on your system, B represents the second. If you omit this parameter, DOS assumes the *default* drive.

Parameter	Definition
path	[\][*dirname*][\ *dirname*[...]] Denotes a path of directory names. Enter the directory names, separated by backslash characters. If a filename is also to be appended, it should be separated from the last directory name by a backslash. For example: **\DIR1\DIR2\FILE1** The first backslash is optional. If used, it tells DOS to begin with the *root* directory. If omitted, the directory path is assumed to begin with the *current* directory. Global filename characters are not allowed in path specifications. The longest path allowed by DOS (from root directory to the last level) is 63 characters.

Parameter	Definition
filename	Diskette filenames are 1-8 characters in length and can be followed by a filename extension. The following characters can be used for filenames: A-Z 0-9 $ & # @ ! % " ( ) - { } _ ^ ~ Any other characters are invalid. An invalid character is assumed to be a delimiter, in which case the filename is truncated. Refer also to "Reserved Device Names" in this chapter for more information about filenames.

Parameter	Definition
.ext	The optional filename extension consists of a period and 1-3 characters. When used, filename extensions immediately follow filenames. The following characters can be used for filename extensions: A-Z 0-9 $ & # @ ! % ' ' ( ) - { } _ ^ ~ Any other characters are invalid. Remember to include the extension when you refer to a file that has a filename extension; otherwise, DOS will be unable to locate the file.
filespec	[*d*:]*filename*[*.ext*] Examples: **B:myprog.COB** **A:yourprog** **DATAFILE.pas** **cobfile**

Reserved Device Names

Certain names have special meaning to DOS. DOS reserves the following names as system devices:

Reserved Name	Device
CON	Console keyboard/screen. If used as an input device, you can press the F6 key; then press the Enter key to generate an end-of-file indication, which ends CON as an input device.
AUX or COM1	First Asynchronous Communications Adapter port.
COM2	Second Asynchronous Communications Adapter port.
LPT1 or PRN	First Parallel Printer (as an output device only).
LPT2 or LPT3	Second Parallel Printer Third Parallel Printer
NUL	Nonexistent (dummy) device for testing applications. As an input device, immediate end-of-file is generated. As an output device, the write operations are simulated, but no data is actually written.

Notes:

1. When using a device name, you should assure that the device actually exists; using the name of a nonexistent device can cause unpredictable errors in DOS operation.

2. The reserved device names can be used in place of a filename.

3. Any drive specifier or filename extension entered with these device names will be ignored.

Global Filename Characters

Two special characters ? and * can be used within a filename and its extension. These special characters give you greater flexibility with the DOS commands.

The ? Character

A ? in a filename or in a filename extension indicates that any character can occupy that position. For example,

```
DIR AB?DE.XYZ
```

lists all directory entries on the default drive with filenames that have five characters, begin with AB, have any next character, are followed by DE, and have an extension of XYZ.

Here are some examples of the files that might be listed by the DIR command:

```
ABCDE XYZ
ABIDE XYZ
ABODE XYZ
```

The * Character

An * in a filename or in a filename extension indicates that any character can occupy that position and all the remaining positions in the filename or extension. For example,

```
DIR AB*.XYZ
```

lists all directory entries on the default drive with filenames that begin with AB and have an extension of XYZ. In this case, the filenames may be from 2-8 characters in length.

Here are some example files that might be listed by the DIR command:

```
ABCDE   XYZ
ABC357  XYZ
ABIDE   XYZ
ABIIOU  XYZ
ABO$$$  XYZ
AB      XYZ
```

Examples of Ways to Use ? and *

Example 1

To list the directory entries for all files named INPUT on drive A (regardless of their filename extension), enter:

```
DIR A:INPUT.???
  or
DIR A:INPUT.*
```

Example 2

To list the directory entries for all files on drive A (regardless of their filenames) with a filename extension of XYZ, enter:

```
DIR A:????????.XYZ
  or
DIR A:*.XYZ
```

Example 3

To list the directory entries for all files on drive A with filenames beginning with ABC and extensions beginning with E, enter:

```
DIR A:ABC?????.E??
  or
DIR A:ABC*.E*
```

Detailed Descriptions of the DOS Commands

This section presents a detailed description of how to use the DOS commands. The commands appear in alphabetical order; each with its purpose, format, and type. Examples are provided where appropriate.

Information Common to All DOS Commands

The following information applies to all DOS commands:

- The normal prompt from the command processor is the default drive letter plus >, such as A> unless changed by the PROMPT command.
- When a command completes, the system prompt will reappear on the screen. If no error messages are displayed before the system prompt reappears, the command has been successfully completed.
- Commands are usually followed by one or more parameters.
- Commands and parameters may be entered in uppercase or lowercase, or a combination of both.

- DOS will search the current directory of the specified or default drive to find a command or batch file whose name you have entered. If not found, DOS will continue its search in each of the directories listed in the PATH command.

- Most commands allowing you to enter filenames will also accept a path (directory) name ahead of the filename. If you do not plan to create directories of your own, you may disregard all references to path names. This will greatly simplify the command syntax for you.

- Commands and parameters *must* be separated by delimiters (space, comma, semicolon, equal sign, or the tab key). The delimiters can be different within one command. For example, you could enter:

```
COPY oldfile.rel;newfile.rel
RENAME,thisfile thatfile
```

- The three parts of filespec (d:filename.ext) must not be separated by delimiters. The colon (:) and period (.) already serve as delimiters.

- In this book, we usually use a space as the delimiter in the commands for readability.

- Also in this book, when we say "*Press any key,*" we mean "Press any *character* key."

- Files are not required to have filename extensions when you create or rename them; however, you must include the filename extension when referring to a file that *has* a filename extension.

- You can end commands while they are running by pressing Ctrl-Break. Ctrl-Break is recognized only while the system is reading from the keyboard or printing characters on the screen, unless you have used BREAK=ON in your configuration file or have issued a BREAK=ON command. Thus, the command may not end immediately when you press Ctrl-Break.

- Commands become effective only after you press the Enter key.

- Global filename characters and device names are not allowed in a command name. You may only use them in command parameters.

- For commands displaying a large amount of output, you can press Ctrl-Num Lock to suspend the display of the output. You can then press any character key to continue the display.

- You can use the control keys and the DOS editing keys described in Chapter 3 while entering DOS commands.

- Drives will be referred to as *source* drives and *target* drives. A source drive is the drive you will be transferring information *from*. A target drive is the drive you will be transferring information *to*.

- When an external command is entered, DOS first looks for it in the current directory of the default or specified drive. If not found, DOS continues searching for it in the directories listed in the most recent PATH command.

- If the characters <, >, or ¦ appear anywhere in the command line you enter, DOS will act upon them as described in "Redirection of Standard Input and Output" and "Piping of Standard Input and Output Device" in Chapter 10. Thus, the command:

  REM this is a ¦ test

  would pipe the output of the REM command (none) to a program named "test". If the program "test" does not exist, this message appears:

  Bad command or filename

ASSIGN (Drive) Command

Purpose: Instructs DOS to use a different drive from the one that was specified for disk operations.

Format: ASSIGN [*x*=*y* [...]]

Type: Internal External
***

Remarks: Use this command to tell DOS to route all requests for a disk drive to a different drive.

The first drive letter *x* is internally converted by DOS to the second drive letter *y*. This command does not require you to enter a colon after the drive letter. Entering ASSIGN with no parameters causes all drive reassignments to be reset so that normal drive assignments will resume.

ASSIGN (Drive) Command

Example: This example causes DOS to route all requests for drive A to drive C. Thus, if you issue **DIR A:**, DOS will display the directory that is on physical drive C:

```
ASSIGN A=C
```

In this example, any requests for drive A or drive B are routed by DOS to drive C:

```
ASSIGN A=C B=C
```

The command:

```
ASSIGN
```

will *undo* the reassignment so that requests for drive A will again go to physical drive A, etc.

> **Note:** This command has been included to assist you with applications that were designed to perform their disk operations specifically on drives A and B (those applications that do not allow you to specify a drive). By using a command such as:
>
> ```
> ASSIGN A=C B=C
> ```
>
> those applications can be made to use drives other than A and B, such as a fixed disk.

Reassignment of drives should *only* be used when necessary for these cases. It should never be used with the PRINT command or when running DOS in normal operations, because it can *hide* the true device type from commands and programs that require actual drive information. Also note that DISKCOPY and DISKCOMP will ignore any drive reassignments.

If you will be developing an application program, we recommend that you avoid using specific drive assignments within your program, but instead, allow the user to specify the drive(s) to be used.

BACKUP (Fixed Disk) Command

Purpose: Backs up one or more files from a fixed disk to diskettes.

Format: BACKUP [*d:*][*path*][*filename*][*.ext*]
d:[/S][/M][/A][/D:*mm-dd-yy*]

Type: Internal External
***

Remarks: Use DOS formatted diskettes only. The first parameter you specify is the fixed disk file you want to back up. The second parameter is the backup diskette drive. Files are backed up from the current directory if you do not specify a path. If you do not specify a filename or extension, then all files in the directory will be backed up.

Global filename characters are allowed in the filename. They cause all files matching the filename to be backed up onto diskettes. For example, entering:

```
BACKUP C:*.DAT A:
```

Causes each file from the current directory of fixed disk drive C that has an extension of .DAT to be backed up onto the diskette in drive A.

The parameter /S causes the files in all sub-directories to be backed up in addition to the files in the specified directory itself. This includes sub-directories at all levels beyond the specified directory.

The parameter /M indicates that only files that have been modified since the last backup should be backed up. Use this parameter to avoid backing up files that never change. The BACKUP command can tell which files have been changed because of an indicator in each file's directory entry that is set by DOS whenever the file is written to.

The parameter /A indicates that backed up files should be added to the files on the backup diskette already in the specified drive. If this parameter is omitted, then you will be prompted to insert a diskette when the backup program is loaded.

The parameter /D can be used to back up files written only on or after the specified date. See the description of the DATE command for valid date formats.

The following example backs up all of the files on fixed disk drive C:

```
BACKUP C:\ A: /S
```

BACKUP (Fixed Disk) Command

The next example backs up three different files from the default fixed disk drive onto the same set of backup diskettes:

```
BACKUP  level1\file1.dat A:
BACKUP  level1\level2\file2.dat A: /A
BACKUP  level1\level3\file3.dat A: /A
```

The next example backs up all files in the current directory that have changed since the last backup:

```
BACKUP *.* A: /M
```

After you enter the BACKUP command, you will be prompted to insert a diskette (unless you specified the /A parameter). Use DOS formatted diskettes only. BACKUP will erase existing files on the diskette before it starts backing up the fixed disk file, unless you have used the /A parameter. After BACKUP fills up a diskette, it will prompt you to insert a new diskette. You should label each diskette and record the date and diskette number.

BACKUP (Fixed Disk) Command

BACKUP displays the name of each file as it backs it up. If you want a printed copy of this list, you can use redirection of output to the printer. Refer to "Redirection of Standard Input and Output Devices" in Chapter 10 for additional information.

The BACKUP command sets the exit code as follows:

0 Normal completion

1 No files were found to backup

3 Terminated by user (Ctrl-Break)

4 Terminated due to error

These codes can be used with the batch processing IF subcommand.

> **Note:** The files on the backup diskettes are unusable in normal processing and should only be used by the RESTORE command.

Batch Commands

Purpose: Executes the commands contained in the specified file from the current directory of the designated or default drive. If the batch file is not found in the current directory, DOS searches for it in the directories listed in the PATH command.

Format: [*d*:]*filename* [*parameters*]

Type: Internal *** External

Remarks: A batch file is a file containing one or more commands that DOS executes one at a time. All batch files must have a filename extension of .BAT.

You can pass parameters to the *filename*.BAT file when the file executes. Therefore, the file can do similar work with different data during each execution.

Batch Commands

You create a batch file by using the Line Editor (EDLIN), or by using the COPY command directly from the keyboard.

Notes:

1. Do not enter the name BATCH (unless the name of the file you want to execute is BATCH.BAT).

2. Only the filename must be entered to run the batch file. Do not enter an extension.

3. The commands in the file named *filename*.BAT are executed.

4. There are seven subcommands that can be used to control batch processing: ECHO, FOR, GOTO, IF, SHIFT, PAUSE, and REM. They are explained in the following pages.

5. If you press Ctrl-Break while in batch mode, this prompt appears:

   Terminate batch job (Y/N)?

   If you press Y, the remainder of the commands in the batch file are ignored and the system prompt appears.

If you press **N**, only the current command ends and batch processing continues with the next command in the file.

6. If you remove a diskette containing a batch file being processed, DOS prompts you to insert it again before the next command can be read.

7. The last command in a batch file may be the name of another batch file. This allows you to invoke one batch file from another when the first is finished.

8. DOS will *remember* which directory your batch file was started from. Therefore, the commands within the batch file may change the current directory at will, and the batch file will continue executing.

The AUTOEXEC.BAT File

The AUTOEXEC.BAT file is a special batch file. When you start or restart DOS, the command processor searches for the AUTOEXEC.BAT file. If this file is present in the root directory of the drive DOS was started from, DOS automatically executes the file whenever you start DOS.

For example, if you want to automatically load BASIC and run a program called MENU, create an AUTOEXEC.BAT file as follows:

1. Enter:

   COPY CON: AUTOEXEC.BAT

   This statement tells DOS to copy the information from the console (keyboard) into the AUTOEXEC.BAT file on the default drive.

2. Now, enter:

   BASIC MENU

   and press Enter.

   This statement goes into the AUTOEXEC.BAT file. It tells DOS to load BASIC and to run the MENU program whenever DOS is started.

3. Press the F6 key, then press the Enter key to end copying from the keyboard to the file.

   The MENU program will now run automatically whenever you start DOS.

To run your own BASIC program, enter the name of your program in place of MENU in the second line of the example. Remember, you can enter any DOS command, or series of commands, in the AUTOEXEC.BAT file.

> **Note:** If you use AUTOEXEC.BAT, DOS does not prompt you for the current date and time unless you include DATE and TIME commands in the AUTOEXEC.BAT file.

Creating a .BAT File With Replaceable Parameters

Within a batch file you may include *dummy* parameters that can be replaced by values supplied when the batch file executes.

For example, enter:

```
A>Copy con: ASMFILE.BAT
Copy %1.MAC %2.MAC
Type %2.PRN
Type %0.BAT
```

Press Enter after entering each line.

Now, press F6; then press Enter.

The system responds with this message:

```
1 File(s) copied
A>_
```

The file ASMFILE.BAT, which consists of three commands, now resides on the diskette in the default drive.

The dummy parameters %0, %1, and %2 are replaced sequentially by the parameters you supply when you execute the file. The dummy parameter %0 is always replaced by the drive designator, if specified, and the filename of the batch file.

Notes:

1. Up to 10 dummy parameters (%0 - %9) can be specified within a batch file, more than 10 parameters can be specified on a command line (see SHIFT subcommand).

2. If you want to use % as part of a filename *within* a batch file, you must specify it twice. For example, to specify the file ABC%.EXE you must enter it as *ABC%%.EXE* in the batch file.

Executing a .BAT File With Replaceable Parameters

To execute the ASMFILE.BAT file and pass parameters, enter the batch filename followed by the parameters you want sequentially substituted for %1, %2, etc.

For example, you can enter:

```
ASMFILE A:PROG1 B:PROG2
```

ASMFILE is substituted for %0, A:PROG1 for %1, and B:PROG2 for %2.

The result is the same as if you entered each of the three commands (in the ASMFILE.BAT file) from the console with their parameters, as follows:

```
Copy A:PROG1.MAC B:PROG2.MAC
Type B:PROG2.PRN
Type ASMFILE.BAT
```

Remember that the dummy parameter %0 is always replaced by the drive designator, if specified, and the filename of the batch file.

ECHO Subcommand

Purpose: The ECHO batch processing subcommand allows or inhibits the screen display of DOS commands executed from a batch file. It does not interfere with messages produced while the commands are executing.

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

Type: Internal External
***

Remarks: Batch commands are normally displayed on the screen as they are read from the batch file. ECHO is ON after power-on or system reset. ECHO ON displays all the commands on the screen as they are executed. ECHO OFF stops the display of commands on the screen (including the REM command).

Echo *message* displays **message** on the screen regardless of the current ON or OFF state. In this way, you can cause specific messages to be displayed even when ECHO has been turned off. If ECHO is issued with no parameters, the current ECHO state (ON or OFF) is displayed.

Example: In this example, the batch file contains the following:

```
echo off
rem **** command display is off
dir a:/w
echo on
dir a:/w
```

Upon execution of the above batch file, the following display will occur:

```
echo off

Volume on drive A has no ID
Directory of A:\

filename1.ext  filename2.ext

2 file(s)    xxxxx bytes free

dir a:/w

Volume in drive A has no ID
Directory of A:\

filename1.ext    filename2.ext

2 file(s)    xxxxx bytes free
```

In the above example, the ECHO OFF is displayed. The **rem** command and **dir a:/w** are not displayed because ECHO is OFF, but the output of the **dir** is not inhibited.

FOR Subcommand

Purpose: The FOR batch processing subcommand allows iterative execution of DOS commands.

Format: FOR %%*variable* IN (*set*) DO *command*

Type: Internal External
***

Remarks: The *%%variable* is sequentially set to each member of *set* and then the *command* is evaluated and executed. If a member of *set* is an expression involving * and/or ?, then the *%%variable* is set to each matching filename from disk.

Example: In this example, if you enter the command:

```
FOR %%f IN (prog1.asm prog2.asm prog3.asm) DO dir %%f
```

The result is

```
dir prog1.asm
dir prog2.asm
dir prog3.asm
```

Note: FOR subcommands cannot be nested; that is, only one FOR subcommand can be specified on a command line. Also, path names are not accepted with filenames.

GOTO Subcommand

Purpose: The GOTO batch processing subcommand transfers control to the line following the one containing the appropriate label. A label is inserted in a batch file as a colon (:) followed by the label name.

Format: GOTO *label*

Type: Internal *** External

Remarks: The GOTO *label* causes commands to be executed beginning with the line immediately after *:label*. If *:label* is not defined, the current batch file terminates with the message **Label not found**. A label in a batch file is defined as a character string where the first 8 characters are significant (make it different).

Example: In this example, the following batch file produces an indefinite sequence of **rem looping...** and **GOTO LOOP** messages on the screen:

```
:LOOP
rem looping...
GOTO LOOP
```

Note that labels within a batch file are never displayed while the batch file is executing. In the example above, the line **:LOOP** would not be displayed. Thus, unreferenced labels provide a handy means for placing comments within your batch file that are not displayed when the file is executed.

IF Subcommand

Purpose: The IF batch processing subcommand allows conditional execution of DOS commands.

Format: IF [NOT] *condition command*

Type: Internal External
***

Remarks: The *condition* parameter is one of the following:

ERRORLEVEL *number*

string1==string2

EXIST *filespec*

When the **IF** parameter's condition is true, then the DOS command is executed. Otherwise, the DOS command is skipped, and the following command in the file is executed.

ERROELEVEL *number* is true if the previous program had an exit code of *number* or higher. The number is specified as a binary value.

String1==string2 is true when *string1* and *string2* are identical.

> **Note:** The corresponding characters of string1 and string2 must both be uppercase or lowercase to be identical.

EXIST *filespec* is true if *filespec* is found on the specified drive. Path names are not allowed with the *filespec*.

NOT *condition* is true if the *condition* is false.

Example: This example is for the IF EXIST *filespec* command:

```
if exist file1 goto abc
        .
        .
        .
:abc
command
```

Execution of a batch file containing this command with *file1* given as the %1 parameter would make the condition true provided *file1* is found on the default drive. The *goto abc* would be executed causing the system to skip to the command following the label *:abc*. If *file1* is not found, the *goto abc* would not be executed. Processing would then continue with the next command in the batch file.

The following example is for the IF string1==string2 command:

```
if %1 == Doug echo Doug was here!
```

Execution of a batch file containing this command with *Doug* given as the %1 parameter would make the condition true. The ECHO batch command would then be executed displaying **Doug was here!**. If *Doug* was not given as the %1 parameter, the condition would have been false. The ECHO batch command would not have been executed. Processing would continue with the next command in the batch file.

The following example is for the IF ERRORLEVEL *number* command:

```
myprog1
if errorlevel 1 echo myprog1 failed - end batch file execution
```

The above two commands are in a batch file; MYPROG1 is a program that sets the errorlevel when it completes its processing. In the simple case, MYPROG1 sets the errorlevel to 0 if it completed processing successfully and sets errorlevel to 1 if processing completed unsuccessfully. The batch file conditional **if errorlevel 1 echo ...** tests for the situation when MYPROG1 failed. If MYPROG1 completed processing unsuccessfully, the condition is true and the ECHO batch command is executed. The ECHO batch command displays the data (or message) immediately following *echo*. If MYPROG1 was successful, the condition would not be true and the ECHO batch command would not be executed. Processing would then continue with the next command in the batch file.

The following example is for the IF NOT EXIST *filespec* command:

```
if not exist a:%1 copy b:%1 a:
program
```

The above IF batch command demonstrates the NOT condition. The batch file that this command is in, is going to execute a program that requires a particular file to be on drive A. The IF command is executed prior to the program to make sure that the required file is on drive A. If the file does not exist on drive A, the condition is true. The *copy* will then be executed, copying the file from drive B to drive A to satisfy the requirements of the program. If the file does exist on drive A, the copy will not be executed. Processing will then continue to execute the program.

> **Note:** At the present time, only BACKUP and RESTORE commands set an ERRORLEVEL that can be tested. The facility is included to allow your own programs to set an error code that can then be interrogated by the IF ERRORLEVEL subcommand.

SHIFT Subcommand

Purpose: The SHIFT batch processing subcommand allows command lines to make use of more than 10 (%0 through %9) replaceable parameters.

Format: SHIFT

Type: Internal *** External

Remarks: Replaceable parameters are numbered %0 through %9. If you wish to use more than 10 parameters on a command line, you can use SHIFT to get at parameters past the tenth. All parameters on the command line are shifted one position to the left, with the %0 parameter being replaced by the %1 parameter, etc. Each subsequent shift command causes all the parameters to be shifted to the left by one position. For example:

```
%0 = A
%1 = B
%2 = C
%3 = D
 .
 .
 .
%9
```

The SHIFT results are:

```
%0 = B
%1 = C
%2 = D
 .
 .
 .
%9
```

Example: This example demonstrates how the SHIFT subcommand can be used in a batch file. If a batch file named **MYFILE.BAT** contains the following commands, and the default drive is A:

```
echo %0 %1 %2 %3
shift
echo %0 %1 %2 %3
shift
echo %0 %1 %2 %3
shift
echo %0 %1 %2 %3
shift
echo %0 %1 %2 %3
```

Invoke the batch file with the following parameters:

```
MYFILE PROG1  PROG2  PROG3
```

Produces the following results:

```
A>echo MYFILE  PROG1  PROG2  PROG3
MYFILE  PROG1  PROG2  PROG3

A>shift

A>echo  PROG1  PROG2  PROG3
PROG1  PROG2  PROG3

A>shift

A>echo PROG2  PROG3
PROG2  PROG3

A>shift

A>echo PROG3
PROG3
A>shift
A>echo

A>
```

PAUSE Subcommand

Purpose: Suspends system processing and displays the message **Strike a key when ready....**

Format: PAUSE [*remark*]

Type: Internal *** External

Remarks: You can insert PAUSE commands within a batch file to display messages and to give you the opportunity to change diskettes between commands. To resume execution of the batch file, press any key *except* Ctrl-Break. (Ctrl-Break ends processing.)

If you include the optional *remark*, the remark is also displayed. The optional remark can be any string of characters up to 121 bytes long.

You can control how much of a batch file you want to execute by placing PAUSE commands at strategic points in the file. At each PAUSE command, the system stops and gives you time to decide whether to end processing. To end processing, press Ctrl-Break. To continue processing, press any other key.

Example: If you enter this PAUSE command in a batch file, the following message is displayed:

```
A >PAUSE Change diskette in drive A
Strike a key when ready... _
```

This PAUSE enables you to change diskettes between commands.

REM (Remark) Subcommand

Purpose: Displays remarks from within a batch file.

Format: REM [*remark*]

Type: Internal *** External

Remarks: The remarks are displayed when the batch execution reaches the remark.

Remarks can be any string of characters up to 123 bytes long.

You can use REM commands without remarks for spacing within your batch file, for readability.

Example: If the following REM command is issued in a batch file, this remark is displayed:

```
REM This is the daily checkout program
```

> **Note:** See "Detailed Descriptions of the DOS Commands" in this chapter for considerations in using the REM command.

BREAK (Control Break) Command

Purpose: Allows you to instruct DOS to check for a control break whenever a program requests DOS to perform any functions (such as disk operations).

Format: BREAK [ON | OFF]

Type: Internal *** External

Remarks: Use this command to specify when DOS should check for a Ctrl-Break being entered at the keyboard. Normally, DOS only performs this check during screen, keyboard, printer, or auxiliary device operations.

Ctrl-Break allows you to *breakout* of a program that performs few or no screen or keyboard operations (such as a compiler).

The ON parameter causes DOS to begin checking for Ctrl-Break *whenever* a program requests any DOS function. Specifying **OFF** causes DOS to check only during screen, keyboard, printer, or Asynchronous Communications Adapter operations.

You can also turn on the extended checking by using BREAK=ON in your configuration file. Refer to "Configuring Your System" in Chapter 9.

Entering **BREAK** with no parameter causes DOS to display the current state (on or off) of Ctrl-Break checking.

CHDIR (Change Directory) Command

Purpose: Change the DOS current directory of the specified or default drive, or to display the current directory path of a drive.

Format: CHDIR [[*d*:]*path*]

or

CD [[*d*:]*path*]

Type: Internal External
***

Remarks: The current directory is where DOS looks to find files whose names were entered without specifying which directory they were in. If you do not specify a drive, the default drive is assumed. If you enter **CHDIR** or **CD** with no parameters or with only a *d*: parameter, the current directory path of the specified or default drive is displayed.

Example: In this example, the command will change the current directory of the default drive to its root directory:

```
CHDIR \
```

In this example, the command will change drive B's current directory to the path "root ⟶ LEVEL1 ⟶ LEVEL2":

```
CD B:\LEVEL1\LEVEL2
```

In this example, the command will change drive B's directory to the current directory path plus LEVEL3:

```
CD B:LEVEL3
```

Thus, if the second example had been used, the resultant path would be:

```
root⟶LEVEL1⟶LEVEL2⟶LEVEL3
```

The search for the LEVEL3 directory begins in the directory that was current when the command was issued.

CHKDSK (Check Disk) Command

Purpose: Analyzes the directories and the File Allocation Table on the designated or default drive and produces a disk and memory status report.

Format: CHKDSK [*d*:][*filename*][/F][/V]

Type: Internal External
***

Remarks: If you specify a filename, CHKDSK will display the number of non-contiguous areas occupied by the file or files. Note that CHKDSK only looks in the current directory for these files.

CHKDSK will not automatically correct errors found in the directory or file allocation table. If you want it to make the corrections, use the /F (fix) parameter. If you do *not* specify the /F parameter, CHKDSK continues to function as though it were preparing to correct the disk so that you can analyze the possible results of correction, but does not actually write the corrections on the disk.

If you use the /V parameter, CHKDSK will display a series of messages indicating its progress, and provide more detailed information about the errors it finds.

After checking the disk, CHKDSK displays any error messages, followed by a status report. A complete listing of error messages can be found in Chapter 8.

Following is an example of the status report that is displayed:

```
Volume MYDISK        Created AUG 12, 1983  10:00

  179712 bytes total disk space
   18944 bytes in 3 hidden files
     512 bytes in 1 directories
   26112 bytes in 4 user files
  134144 bytes available on disk

  196608 bytes total memory
  170736 bytes free
```

Note that in this status report, *three hidden* files were reported. These are the volume label, and the DOS system files IBMBIO.COM and IBMDOS.COM, that are hidden from the normal directory searches. Some application programs also create hidden files.

CHKDSK does not wait for you to insert a diskette. It assumes that the diskette to be checked is in the specified drive. Therefore, on a single diskette-drive system, it is especially important that the specified drive is different from the default drive, unless you are checking the DOS diskette itself.

CHKDSK (Check Disk) Command

You should run CHKDSK occasionally for each fixed disk and diskette to ensure the integrity of the file structures.

Notes:

1. All yes or no (Y/N) prompts from CHKDSK require you to press Enter after entering **Y** or **N**, to prevent accidental changes to your disk.

2. If you specified a filename, the number of non-contiguous areas occupied by the file will be reported. Badly fragmented files (many non-contiguous areas) can cause system performance to slow down when those files are accessed, since DOS can not read them sequentially. You can determine the extent of file fragmentation by using ***.*** in the filename field of the CHKDSK command.

3. If CHKDSK finds *lost* allocation units (clusters) on the disk, it asks if you wish to recover the lost data into files. If you say yes, and the **/F** parameter was used, CHKDSK recovers each chain of lost allocation units into a file whose name is in the form:

   FILE*nnnn*.CHK

Where *nnnn* is a sequential number starting with 0000. These files are created in the root directory of the specified drive. You can then look at these files to see if they have any useful information. If not, you can erase them.

4. If you redirect CHKDSK's output to a file, for example:

```
CHKDSK B:>FILE
```

It will report errors on that file. In this case, be sure not to use the /F parameter.

CLS (Clear Screen) Command

Purpose: Clears the display screen.

Format: CLS

Type: Internal External
***

Remarks: This command clears the display screen upon execution. If foreground and background colors have been selected through the "Extended Screen Control and Keyboard" functions described in Chapter 13, the colors will remain unchanged. Otherwise, the screen is set to white characters on a black background.

COMP (Compare Files) Command

Purpose: Compares the contents of the first set of specified files to the contents of the second set of specified files. Usually, you would run COMP after a COPY operation to ensure that the two sets of files are identical.

> **Note:** This command compares two sets of *files*; the DISKCOMP command compares two *entire diskettes*.

Format: COMP [*d*:][*path*][*filename*[.*ext*]]
[*d*:][*path*][*filename*[.*ext*]]

Type: Internal External
***

Remarks: The first parameter you specify is the *primary* file. The second parameter is the *secondary* file. The files that you compare may be on the same drive or on different drives. They can also be in the same directory or different directories.

Global filename characters are allowed in both filenames, and will cause all of the files matching the first filename to be compared with the corresponding files from the second filename. Thus, entering:

```
COMP A:*.ASM  B:*.BAK
```

causes each file that has an extension of .ASM from drive A to be compared with a file of the same name (but with an extension of .BAK) from drive B.

If no parameters are entered with the COMP command, or if the second parameter is missing, you are prompted for them. If either parameter only contains a drive or a path with no filename, COMP assumes a filename of *.*. You can enter a complete path with either of the two filenames. For example, the command:

```
COMP B:*.ASM C:
```

causes all .ASM files on drive B to be compared with the files of the same names and extensions on drive C.

It is also possible to compare all files in one directory with all corresponding files in another directory. For example:

```
COMP A:\LEVEL1 A:\LEVEL2
```

locates all the files in the LEVEL1 directory of drive A and compares them with the same filenames from the LEVEL2 directory on the same drive.

If no file matches the primary filename, COMP will prompt you for both the primary and secondary parameters again.

COMP (Compare Files) Command

The paths and names of the files being compared are displayed as the comparing process proceeds. An error message will follow if a file matching the second filespec cannot be found, or the files are different sizes, or a specified directory path is invalid.

During the comparison, an error message appears for any location that contains mismatching information in the two files. The message indicates the offset into the files of the mismatching bytes, and the contents of the bytes themselves (all in hexadecimal), as follows:

```
Compare error at offset XXXXXXXX
File 1 = XX
File 2 = XX
```

In this example, File 1 is the first filename entered; File 2 is the second filename entered.

After ten unequal comparisons, COMP concludes that further comparing would be useless; processing ends; and the following message is displayed:

```
10 Mismatches - ending compare
```

After a successful comparison, COMP displays:

Files compare OK

After the comparison of the two files ends, comparing will proceed with the next pair of files that match the two filenames, until no more files can be found that match the first parameter. Then COMP displays:

Compare more files (Y/N)? _

You now have the option to compare two more files or to end the comparison. If you want to compare two more files, enter Y. You will be prompted for new primary and secondary filenames.

If you want to end COMP processing, enter N. You will return to the DOS prompt. In all compares, COMP looks at the last byte of the files being compared to assure that it contains a valid end-of-file mark (Ctrl-Z, which is the hexadecimal character 1A). If found, no action is taken by COMP. If the end-of-file mark is *not* found, COMP produces the message:

EOF mark not found

This is done because some products produce files whose sizes are always recorded in the directory as a multiple of 128 bytes, even though the actual usable data in the file is usually a few bytes less than the directory size. In this case, COMP may produce **compare error** messages when comparing the few bytes beyond the last real data byte in the last block of 128 bytes (COMP always compares the number of bytes reflected in the directory). Thus, the **EOF mark not found** message indicates that the compare errors may not have occurred in the usable data portion of the file.

Notes:

1. The two sets of files you want to compare can have the same path and filenames—provided they are on different drives.

2. If you only specify a drive for the second file, it is assumed that the second filename is the same as the first filename. But, the current directory is to be used. That is, it is the same as entering:

   ```
   d:*.*
   ```

3. A comparison does not take place if the file sizes are different.

4. COMP does not wait for you to insert a diskette containing a file to be compared. Therefore, if a file to be compared is not on the same diskette as the COMP command itself, you should enter COMP with no parameters. When COMP prompts for the filenames, you can insert the desired diskette and reply with the name of the file to be compared.

COPY Command

Purpose: Copies one or more files to another disk and optionally, gives the copy a different name if you specify it in the COPY command.

COPY also copies files to the same disk. In this case, you *must* give the copies different names unless different directories are specified; otherwise, the copy is not permitted. Concatenation (combining of files) can be performed during the copying process.

You can also use COPY to transfer data between any of the system devices. (An example of how to copy information that you enter at the keyboard to a file is provided at the end of the description of COPY.)

Format: COPY [/A][/B][*d*:][*path*]*filename*[.*ext*][/A][/B]
[*d*:][*path*][*filename*[.*ext*]][/A][/B][/V]

or

COPY [/A][/B][*d*:][*path*]*filename*[.*ext*][/A][/B]
[+[*d*:][*path*]*filename*[.*ext*][/A][/B]...]
[*d*:][*path*][*filename*[.*ext*]][/A][/B][/V]

Type: Internal External
***

COPY Command

Remarks: The first file specified is the source file. The second file specified is the target file. If the second parameter is a directory (*path* with no filename), files are copied into that directory without changing their names.

The parameter /V causes DOS to verify that the sectors written on the target diskette are recorded properly. Although errors in recording data are very rare, this option has been provided for those of you who wish to verify that critical data has been correctly recorded. This option causes the COPY command to run more slowly, due to the additional overhead of verification.

The /V parameter provides the same check as does the VERIFY ON command. /V is redundant if the VERIFY ON command has been executed previously. The difference is that /V is effective during the execution of the COPY command. The VERIFY ON command is in effect continually until VERIFY OFF is entered.

The parameters /A and /B indicate the amount of data to be processed by the COPY command. Each applies to the filespec preceding it and to all remaining filespecs on the command line until another /A or /B is encountered. These parameters have the following meanings:

When used with a *source* filespec:

/A Causes the file to be treated as an ASCII (text) file. The file's data is copied up to, but not including, the first end-of-file character (Ctrl-Z, which is hex 1A) found in the file; the remainder of the file is not copied.

/B Causes the entire file (based on the directory file size) to be copied.

When used with a *target* filespec:

/A Causes a Ctrl-Z character to be added as the last character of the file.

/B Causes no end-of-file character (Ctrl-Z) to be added.

The default values are /A when concatenation is being performed (see Option 3 below), and /B when concatenation is not being performed (Options 1 and 2).

Notes:

1. When copying to or from a reserved device name, the copy is performed in ASCII (/A) mode. The first Ctrl-Z character encountered will end the copy unless /B was specified.

2. When making a copy of a file that is marked read-only, the copy will not be marked read-only.

You can use the global characters ? and * in the filename and in the extension parameters of both the original and duplicate files. If you enter a ? or * in the source *filespec*, the names of the files will be displayed as the files are being copied. For more information about global characters, refer to "Global Filename Characters" in this chapter.

COPY Command

The COPY command has three format options:

Option 1 – Copy With Same Name

Use this option to copy a file with the copied file having the *same* filename and extension as the source file. For example:

COPY [*d:*][*path*]*filename*[*.ext*]

or

COPY [*d:*][*path*]*filename*[*.ext*] *d:*[*path*]

In the first example, we want to copy a file to the current directory of the default drive. In the second example, we specify the target drive and/or directory. In both examples, because we did not specify the second filename, the copied file will have the same filename as the source file. Because we did not specify a name for the second file, the source drive and the target drive must be different unless different directories were specified or implied; otherwise, the copy is not permitted.

For example, assume the default drive is A. The command:

```
COPY B:MYPROG
```

copies the file MYPROG from drive B, to the default drive A, with no change in the filename. The command:

```
COPY *.* B:
```

copies all the files from the default drive A to drive B, with no change in the filenames or in the extensions. The filenames are displayed as the files are copied. This method is very useful if the files on drive A are fragmented. The command:

```
COPY B:\MYPROG    B:\LEVEL1
```

copies the file MYPROG from the root directory of drive B to the directory path:

root—>LEVEL1

on the same drive. The copy has the same filename as the original file. Note that the above example assumes that directory LEVEL1 exists on drive B. If it did not, then the file MYPROG would have been copied into a file named LEVEL1 in the root directory of drive B. In other words, if the second parameter specifies a directory that exists, the file (or files) will be placed in that directory, keeping the same filename. If the second parameter does not specify a directory that exists, DOS will treat it as a filename.

Option 2 – Copy With Different Name

Use this option when you want the copied file to have a different name from the file that is being copied. For example:

COPY [*d*:][*path*]*filename*[*.ext*] [*d*:][*path*]*filename*[*.ext*]

or

COPY [*d*:][*path*]*filename*[*.ext*] [*d*:][*path*]*filename*[*.ext*]

In the first example, we copied a file (first file specified), and renamed the copy (second file specified). We did not specify a drive, so the default drive was used. In the second example, we copied a file and renamed the copy also. In this example, we did specify the target drive. Because we changed the name of the file, the source drive and the target drive do not have to be different. The current directory can be the same or different.

For example:

```
COPY MYPROG.ABC B:*.XXX
```

copies the file MYPROG.ABC from the diskette in default drive A to drive B, naming the copy MYPROG.XXX. The current directory of each drive was used.

You can also use reserved device names for the copy operation. For example:

```
COPY CON: fileA
COPY CON: AUX:
COPY CON: LPT1:
COPY fileA CON:
COPY fileB AUX:
COPY fileC LPT2:
COPY AUX: LPT1:
COPY AUX: CON:
```

Also, NUL: can be used in any variation. The colon is not required when specifying a reserved device name.

Refer to "Reserved Device Names" in this chapter for information about system devices.

This example shows how to use COPY to put what you enter from the keyboard into a file:

```
A>COPY CON: fileA
Type a line and press Enter.
Type your next line and press Enter.
•
•
•
Type your last line and press Enter.
Now, press F6 and then press Enter.
```

When you press F6, and then press Enter, the COPY operation ends and saves the information you entered. In this example, the information is saved in a file named **fileA.**

> Note: This example assumes that you have not altered the meaning of F6 through the "Extended Keyboard Support" functions described in Chapter 13. If you have, then substitute the key that you have assigned Ctrl-Z for F6 in this example.

Option 3 – Copy and Combine Files

Use this option when you want to combine files while copying. That is, you can combine two or more files into one file by adding the additional files to the end of the first. The date and time recorded in the result file directory are the current date and time. The message indicating the number of files copied refers to the number of result files created.

To combine files, list any number of source files, separated by plus (+) signs in the COPY command. Use the following format:

COPY [/A][/B][*d*:][*path*]*filename*[.*ext*][/A][/B]

[+[*d*:][*path*]*filename*[.*ext*][/A][/B]...]

[*d*:][*path*]*filename*[.*ext*]][/A][/B][/V]

For example:

```
COPY A.XYZ+B.ABC+B:C.TXT  BIGFILE.TXT
```

This command creates a new file called **BIGFILE.TXT** on the default drive. The combination of **A.XYZ**, **B.ABC**, and **B:C.TXT** is put into **BIGFILE.TXT**.

If you do not specify a result *filename*, the additional files are added to the end of the first file, leaving the result in the first file. For example:

```
COPY A.ASM+B.ASM
```

In this case, COPY appends **B.ASM** to the end of **A.ASM** and leaves the result in **A.ASM**.

> **Note:** Combining files is normally performed in text (or ASCII) mode. That is, the first Ctrl-Z (hex 1A) character in the file is interpreted as an end-of-file mark. To combine binary files, use the **/B** parameter to force COPY to use the physical end-of-file (the file length shown in the DIR command).

You can also combine ASCII and binary files by using the following parameters:

- ASCII - /A
- Binary - /B

For example:

```
COPY A.XYZ+B.COM/B+B:C.TXT/A BIGFILE.TXT
```

A /A or /B takes effect on the file it is placed after, and it applies to all subsequent files on the command line until another /A or /B is found. A /A or /B on the result file causes a Ctrl-Z to be added (/A), or not to be added (/B), as the last character in the result file.

You can use the global characters ? and * in the filenames of both the files to be combined and the result file. For example:

```
COPY *.LST COMBIN.PRN
```

In this example, all files matching *.LST are combined into one file called COMBIN.PRN. Also:

```
COPY *.LST+*.REF COMBIN.PRN
```

This example combines all files matching *.LST and then all files matching *.REF into one file called COMBIN.PRN.

```
COPY *.LST+*.REF *.PRN
```

In this example, each file matching *.LST is combined with the corresponding .REF file, with the result having the same name but with extension .PRN. Thus, a file FILE1.LST would be combined with FILE1.REF to form FILE1.PRN; XYZ.LST would be combined with XYZ.REF to form XYZ.PRN; etc. Note that in this case (when multiple files are to be created), only one file from each of the source filespecs is used to create a given target file.

For more information about global characters, refer to "Global Filename Characters" in this chapter.

It is easy to enter a COPY command to combine files where one of the source files is the same as the target, yet this often cannot be detected. For example:

```
COPY *.LST ALL.LST
```

This would produce an error if ALL.LST already existed. The error would not be detected, however, until it was time for ALL.LST to be appended; by this time, ALL.LST could already have been altered.

COPY handles this situation as follows: As each input file is found, its name is compared with the target filename. If the names are the same, that one input file is skipped, and the message **Content of destination lost before copy** is displayed. Further copying proceeds normally. This allows *summing* files, with a command like:

```
COPY ALL.LST + *.LST
```

This command appends all *.LST files, except ALL.LST itself, to ALL.LST. In this case, the error message is suppressed, because this is a true *physical append* to ALL.LST.

The following are special cases. Remember to use the /B parameter whenever you use the plus (+) sign with non-ASCII files.

```
COPY B:XYZ.ASM+
```

This command copies the file XYZ.ASM to the default drive and gives it a new date and time. To simply change the date and time, leaving the file in place, you can use the following command:

```
COPY B:XYZ.ASM+, , B:
```

Note that the two commas are necessary to define the end of the *source filename*, because COPY normally expects to see another filename after the plus (+) sign.

In these special cases, if global filename characters are used in the filename or extension, then all of the matching files will be appended together into the first filename that matches. Thus, the command:

```
COPY B:*.* +, , B:
```

will not update the dates and times of all files on drive B, but will append all of drive B's files together into a single file that will replace the first file found on drive B.

> Note: When combining files, COPY considers the copying process to be successful if at least one, but not necessarily all, of the named source files is found. If none of the source files can be found, you receive the message
>
> ```
> 0 file(s) copied
> ```

DATE Command

Purpose: Permits you to enter a date or change the date known to the system. The date is recorded in the directory entry for any files you create or alter.

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

Type: Internal *** External

Remarks: If you enter a valid date with the DATE command, the new date is accepted, and the system prompt appears. Otherwise, the DATE command issues the following prompt:

```
Current date is day  mm-dd-yy
Enter new date: __
```

The system displays the day of the week in the *day* location.

Enter a new date in the form *mm-dd-yy* or *mm/dd/yy*, where:

mm is a one- or two-digit number from 1-12
dd is a one- or two-digit number from 1-31
yy is a two-digit number from 80-99 (the 19 is assumed) or a four-digit number from 1980-2099

DATE Command

You can change the date from the keyboard or from a batch file. Remember, when you start the system, it does not prompt you for the date if you use an AUTOEXEC.BAT file. You may want to include a DATE command in that file. For more information about the AUTOEXEC.BAT file, refer to "Batch Commands" in this chapter.

Notes:

1. To leave the date as is, press Enter.
2. The valid delimiters within the date are hyphens (-) and slashes (/).
3. Any date is acceptable as today's date, as long as the digits are in the correct ranges.
4. If you enter an invalid date or delimiter, you receive an **Invalid date** message.

Example: In this example, once you press Enter, the date known to the system is 7/24/82.

```
A>DATE
Current date is Mon 1-18-1982
Enter new date: 7/24/82
```

DEL Command

See "ERASE Command" in this chapter.

DIR (Directory) Command

Purpose: Lists either all the directory entries, or only those for specified files. The information provided in the display includes the volume identification and the amount of free space left on the disk. The display line for each file includes its size in decimal bytes and the date and time the file was last written to. Entries that name other directories are clearly identified with <DIR> in the file size field.

> **Note:** Directory entries for system files IBMBIO.COM and IBMDOS.COM are not listed, even if present.

Format: DIR [*d:*][*path*][*filename*[*.ext*]][/P][/W]

Type: Internal External
***

Remarks: The /P parameter causes the display to pause when the screen is full. When you are ready to continue with the directory listing, press any key.

The /W parameter produces a wide display of the directory, which lists only the filenames and directory names. Each line displayed contains five names. (This parameter is only recommended for 80-column displays.)

You can use the global characters ? and * in the filename and extension parameters. For more information about the global characters, refer to "Global Filename Characters" in this chapter.

The DIR command has two format options (the /P and /W parameters may be used with either option):

Option 1 – List All Files

Use this option to list all the files in a directory. For example:

DIR [*path*]

or

DIR *d*:[*path*]

In the first example, we want to list all directory entries on the default drive. In the second example, we want to list all directory entries on the specified drive. In both cases, if a path is specified, the listing is of files in the specified directory. Otherwise, the current directory is listed.

The directory listing might look like this:

```
A>dir

 Volume in drive A is MYDISK
 Directory of A: \

FILE1      A         10368      7-20-83   12:13p
FILE3      A          1613      5-27-83   12:14p
9X                      31      8-17-82   10:59a
LEVEL2             <DIR>        9-09-82   12:10p
FILE1                2288       9-02-82   12:25p
          5 File(s) 141312 bytes free
```

Note that if the directory being listed is not the root directory, the above example would have included two special entries. The first entry would contain a period in place of a filename. The second would contain two periods in place of a filename. The list of files shown above would follow these two entries. These special entries tell you that the directory being listed is a sub-directory, rather than the root directory.

DIR (Directory) Command

Option 2 – List Selected Files

Use this option to list selected files from a directory. For example:

DIR *filename.ext*

or

DIR *d:filename.ext*

If either *filename* or *.ext* is omitted, an * is assumed.

In the first example, we want to list all the files in the directory of the *default* drive that have the specified filename and extension. In the second example, we want to list all the files in the directory of the *specified* drive that have the specified filename and extension.

Using the previous example, if you enter:

```
dir file3.a
```

the screen might look like this:

```
A>dir file3.a

 Volume in drive A is MYDISK
 Directory of  A:\

FILE3      A      1613      5-27-83      12:14p
        1 File(s)  141312 bytes free
```

If you enter:

```
dir *.a
```

or

```
dir .a
```
(omission of *filename* defaults to *)

the screen might look like this:

```
A>dir *.a

Volume in drive A is MYDISK
Directory of A:\

FILE1    A          10368          7-20-83     12:13p
FILE3    A           1613          5-27-83     12:14p
         2 File(s)  141312 bytes free
```

If you enter:

```
dir file1
```

the screen might look like this (omission of *.ext* defaults to *):

```
A> dir file1

 Volume in drive A is MYDISK
 Directory of A:\

FILE1    A          10368          7-20-83     12:13p
FILE1               2288          9-02-82     12:25p
         2 File(s)  141312 bytes free
```

DIR (Directory) Command

To display only the entry for a file that has no extension, enter the filename followed by a period. In this case, the *.ext* does *not* default to *. For example,

```
dir file1
```

displays the entry for FILE1, but not for FILE1.A.

If you wish to display all the files in directory LEVEL2 on the above drive, you can enter:

```
dir level2
```

The screen will look like this:

```
A>dir level2

Volume in drive A is MYDISK
Directory of A:\level2

.          <DIR>          9-09-82
..         <DIR>          9-09-82
MYPROG  COM      2463      7-30-82   8:55a
        3 File(s)  141312 bytes free
```

Note that all files in directory LEVEL2 have been listed, including the two special entries found in all sub-directories. The entry marked with a single period denotes the directory being listed (LEVEL2), and the double period denotes this directory's parent directory (in this case, the root directory). Thus, if your *current* directory is LEVEL2 and you wish to see the files in its parent directory, you can enter:

```
dir ..
```

The following screen is displayed:

```
A>dir ..

Volume in drive A is MYDISK
Directory of A:\

FILE1      A          10368   7-20-83   12:13p
FILE3      A           1613   5-27-83   12:14p
9X                       31   8-17-82   10:59a
LEVEL2          <DIR>         9-09-82   12:10p
FILE1                  2288   9-02-82   12:25p
        5 File(s)  141312 bytes free
```

DISKCOMP (Compare Diskette) Command

Purpose: Compares the contents of the diskette in the first specified drive to the contents of the diskette in the second specified drive. Usually, you would run DISKCOMP after a DISKCOPY operation to ensure that the two diskettes are identical.

Notes:

1. This command is used only for comparing diskettes. If a fixed disk drive letter is specified, an error message is displayed.

2. This command compares two *entire diskettes*; the COMP command compares two *files*.

Format: DISKCOMP [*d*:][*d*:][/1][/8]

Type: Internal External
***

Remarks: You can specify the same drive or different drives in this command. If you specify the same drive, a single-drive comparison is performed. You are prompted to insert the diskettes at the appropriate time. DISKCOMP waits for you to press any key before it continues.

The /1 parameter forces DISKCOMP to compare only the first side of the diskettes, even if the diskettes and drives are dual-sided.

The /8 parameter causes DISKCOMP to compare only 8 sectors per track, even if the first diskette contains 9 sectors per track.

DISKCOMP compares all 40 tracks on a track-for-track basis and issues a message if the tracks are not equal. The message indicates the track number (0-39) and the side (0 or 1) where the mismatch was found.

After completing the comparison, DISKCOMP prompts:

Compare more diskettes (Y/N)?_

If you press Y, the next comparison is done on the same drives that you originally specified, after you receive prompts to insert the proper diskettes.

DISKCOMP (Compare Diskette) Command

To end the command, press **N**.

Notes:

1. If you omit both parameters, a single-drive comparison is performed on the default drive.

2. If you omit the second parameter, the default drive is used as the secondary drive. If you specify the default drive in the first parameter, this also results in a single-drive comparison.

3. On a single-drive system, all prompts are for drive A, regardless of any drive specifiers entered.

4. DISKCOMP usually does not issue a **Diskettes compare OK** message if you try to compare a backup diskette created by the COPY command with the diskette you copied from. The COPY operation produces a copy that contains the same information, but places the information at different locations on the target diskette from those locations used on the source diskette. In this case, you should use the COMP command to compare individual files on the diskettes.

5. If a diskette error occurs while DISKCOMP is reading the diskette, a message is produced that indicates where (track and side) the error occurred. Then DISKCOMP continues to compare the rest of the diskette. Because the remainder of the data to be compared cannot be read correctly from the indicated track and side, you can expect to receive a **Compare error** message.

6. DISKCOMP automatically determines the number of sides and sectors per track to be compared, based on the diskette that is to be read first (the first drive parameter entered).

   If the first diskette or drive can be read on only one side, or if the **/1** parameter is used, only the first side is read from both diskettes. If the first diskette contains 9 sectors per track, then DISKCOMP will compare 9 sectors per track unless you used the **/8** parameter. If the first drive and diskette are dual-sided, and /1 is not specified, a two-sided comparison is done. An error message is produced if either the second drive or the diskette is a single-sided diskette.

DISKCOPY (Copy Diskette) Command

Purpose: Copies the contents of the diskette in the source drive to the diskette in the target drive. The target diskette is formatted if necessary, during the copy.

> **Note:** This command is used only for copying diskettes. If a fixed disk drive letter is specified, an error message is displayed.

Format: DISKCOPY [*d*:] [*d*:][/1]

Type: Internal External
***

Remarks: The first parameter you specify is the source drive. The second parameter is the target drive.

The /1 parameter causes DISKCOPY to copy only the first side of the diskette, regardless of the diskette or drive type.

You can specify the same drives or different drives. If the drives are the same, a single-drive copy operation is performed. You are prompted to insert the diskettes at the appropriate times. DISKCOPY waits for you to press any key before continuing.

After copying, DISKCOPY prompts:

```
Copy another (Y/N)?_
```

If you press **Y**, the next copy is done on the same drives that you originally specified, after you are prompted to insert the proper diskettes.

To end the command, press **N**.

Notes:

1. If the target diskette has not been formatted with the same number of sides and sectors per track as the source diskette, DISKCOPY will format the target diskette during the copy operation.

2. If you omit both drive parameters, a single-drive copy operation is performed on the default drive.

3. If you omit the second parameter, the default drive is used as the target drive.

4. If you omit the second parameter and you specify the default drive as the source drive, a single-drive copy operation is performed.

5. On a single-drive system, all prompts will be for drive A, regardless of any drive letter you may enter.

6. Diskettes that have had a lot of file creation and deletion activity become **fragmented**, because diskette space is not allocated sequentially. The first free sector found is the next sector allocated, regardless of its location on the diskette.

   A fragmented diskette can cause degraded performance due to excessive head movement and rotational delays involved in finding, reading, or writing a file.

   If this is the case, it is recommended that you use the COPY command, instead of DISKCOPY, to eliminate the fragmentation.

   For example, place a freshly formatted diskette in drive B, and the diskette you wish to copy in drive A. The command:

   ```
   COPY A:*.* B:
   ```

   copies all the files from the diskette in drive A to the diskette in drive B.

7. You can run DISKCOMP after a successful DISKCOPY to ensure that the diskettes are identical.

8. If disk errors are encountered on either diskette, DISKCOPY indicates the drive, track, and side in error and proceeds with the copy. In this case, the target diskette (copy) may or may not be usable, depending on whether the affected diskette location was to contain valid data.

9. DISKCOPY automatically determines the number of sides and sectors per track to copy, based on the source drive and diskette. If only the first side of the source diskette can be read, then only the first side can be copied. If the source drive and diskette are dual-sided, both sides can be copied (unless you override it with the /1 parameter). In this case, if the target drive is single-sided, an error message will indicate that the drives are incompatible.

   If the source diskette has ever been physically formatted with 9 sectors per track, then all 9 sectors on each track will be copied.

ERASE Command

Purpose: Deletes the file with the specified filename from the specified directory on the designated drive, or deletes the file from the default drive if no drive is specified. If no path is entered, the file is deleted from the current directory.

Format: ERASE [*d:*] [*path*] [*filename*[.*ext*]]

or

DEL [*d:*] [*path*] [*filename*[.*ext*]]

Type: Internal *** External

Remarks: The shortened form, DEL, is a valid abbreviation for ERASE.

You can use the global characters ? and * in the filename and in the extension. Global characters should be used with caution, however, because multiple files can be erased with a single command. For more information about global characters, refer to "Global Filename Characters" in this chapter.

To erase all files in the current directory, enter:

```
ERASE [d:]*.*
```

To erase all files in a specific directory, enter:

ERASE [d:]path

ERASE assumes a filename of *.* if no filename is given.

Notes:

1. The system files IBMBIO.COM and IBMDOS.COM cannot be erased.

2. If you use the filespec *.* to erase all of the files in a directory or on a disk, DOS issues the following message to verify that you actually want to erase all files:

   Are you sure (Y/N)?

   If you *do* want to erase all of the files on the diskette, enter Y. Otherwise, enter N. Then press the Enter key.

3. The two special entries in each sub-directory (. and .. in place of filenames) cannot be erased.

Example: In this example, the file **myprog.1** will be erased from the current directory of drive A.

A>ERASE A:myprog.1

FORMAT Command

Purpose: Initializes the disk in the designated or default drive to a recording format acceptable to DOS; analyzes the entire disk for any defective tracks; and prepares the disk to accept DOS files by initializing the directory, File Allocation Table, and system loader.

Format: FORMAT [*d*:][/S][/1][/8][/V][/B]

Type: Internal External
***

Remarks: You must format all new diskettes (by using either the FORMAT or DISKCOPY command) and fixed disks (through FORMAT) before they can be used by DOS.

A fixed disk must also be formatted again if you change the size of its DOS partition through the FDISK command.

If you specify /S in the FORMAT command, the operating system files are also copied from the default drive to the new disk or diskette in the following order:

IBMBIO.COM
IBMDOS.COM
COMMAND.COM

If you specify **/1**, the target diskette is formatted for single-sided use, regardless of the drive type.

If you specify **/8**, the target diskette is formatted for use at 8 sectors per track. FORMAT will default to 9 sectors per track usage if you do not specify **/8**. Note that format always creates 9 physical sectors on each diskette track, but that it instructs DOS to use only 8 sectors per track if you use the **/8** parameter. The **/1** and **/8** parameters are valid only for diskettes.

If you specify **/V**, FORMAT will prompt you for a volume label which will be written on the disk.

The volume label cannot be used in place of filenames as input to any of the DOS commands. The volume label is for your use in keeping track of your diskettes. The **/V** parameter cannot be used with the **/8** parameter.

The **/B** parameter causes FORMAT to create an 8 sector per track diskette with space allocated for the IBMBIO.COM and IBMDOS.COM system modules. It does not place the system modules or the command processor on the diskette. This parameter is used to create a diskette on which any version of DOS (1.00, 1.10, or 2.00) can be placed through that version's SYS command. If the **/B** parameter is not used, only DOS Version 2.00 can be placed on the diskette through the SYS command.

The /V and /S parameters cannot be used with the /B parameter.

Notes:

1. Formatting destroys any previously existing data on the disk.

2. During the formatting process, any defective tracks are marked as *reserved* to prevent the tracks from being allocated to a data file.

3. Directory entries for IBMBIO.COM and IBMDOS.COM are marked as *hidden files*, and therefore, they do not appear in any directory searches—including the DIR command.

4. FORMAT will prompt you to enter a volume label (volume identification) if you have used the /V parameter. The label can consist of from 1 to 11 characters. All characters acceptable in filenames are acceptable in the volume label. Unlike filenames, however, the volume label does not contain a period between the eighth and ninth characters.

5. FORMAT produces a status report, that indicates:

   - Total disk space
   - Space marked as defective
   - Space currently allocated to files (when /S is used)
   - Amount of space available for your files

6. FORMAT determines the target drive type and formats the disk or diskette accordingly. For diskettes, if the diskette can be successfully read and written on only one side, the diskette is formatted for single-sided use; it can be used in either type of drive. If the target drive is dual-sided and you do not use the /1 parameter, the diskette is formatted for dual-sided use; it will not be usable in a single-sided drive.

7. Fixed disks are already physically formatted (proper recording format) when shipped by IBM. When formatting a fixed disk, FORMAT checks all locations within the DOS partition, but does not physically format them again.

8. If the /S parameter is used and the system has insufficient available memory for FORMAT to load all three system modules, it will load as many modules as it can, format the target disk and write the modules that are in memory. It must then read the remaining modules from the source disk so they can be placed on the target disk. If the source diskette has been removed from the drive, an appropriate message will prompt you to reinsert it before FORMAT can continue.

Example: By issuing the following command, the diskette in drive B is formatted and the operating system files are also copied:

```
A >FORMAT B:/S/V
```

The system displays the following message:

```
Insert new diskette for drive B:
and strike any key when ready
```

After you insert the appropriate diskette and press any key, the system displays this message:

```
Formatting...
```

while the diskette formatting is taking place.

Once the formatting is complete, the system displays this message:

```
Formatting...Format complete
System transferred

Volume label (11 character, ENTER for none)? MYDISK

  179712 bytes total disk space
   xxxxx bytes used by system
  xxxxxx bytes available on disk

Format another (Y/N)?n
```

In the above example, note that MYDISK was entered as the volume label.

Enter Y to format another diskette.

Enter N to end the FORMAT program.

When you format a fixed disk, you will see the following message instead of the prompt to insert a diskette:

```
Press any key to begin formatting x:
```

The *x* is replaced by the drive letter you typed. The messages will otherwise appear in the same manner.

Fixed disk formatting can take several minutes because of the large size that can be allocated to DOS, so don't be alarmed if it takes some time before you are prompted for the volume label. You can tell that FORMAT is working by noting that your fixed disk drive light is on.

GRAPHICS (Screen Print) Command

Purpose: Prints the contents of a graphics display screen on an IBM Personal Computer 80 cps Graphics Printer when using a Color/Graphics Monitor Adapter. This command increases the resident size of DOS in memory by 736 bytes.

Format: GRAPHICS

Type: Internal External
***

Remarks: Press the Shift-PrtSc keys to print the screen contents on the printer. If the screen is in text mode, the text is printed in under 30 seconds. If the screen is in the graphics mode, each time the PrtSc key is pressed, the following things occur:

- In the 320x200 color graphics mode, the screen contents are printed in up to four shades of gray.
- In the 640x200 color graphics mode, the screen is printed sideways on the paper. The upper right corner of the screen is printed on the upper left corner of the paper.
- Printing may take as long as three minutes.
- To invoke the screen print from a program, use the following coding example:

```
PUSH BP
INT 5
POP BP
```

MKDIR (Make Directory) Command

Purpose: Creates a sub-directory on the specified disk.

Format: MKDIR [*d*:]*path*

or

MD [*d*:]*path*

Type: Internal *** External

Remarks: If you do not specify a drive, the default drive is assumed.

Example: In this example, the command creates an entry in the root directory for a new sub-directory called LEVEL2:

```
MD\LEVEL2
```

If you have done this and wish to add another directory level, you can use either of the following two examples.

Use this example if your current directory is the root directory:

```
MD\LEVEL2\LEVEL3
```

MKDIR (Make Directory) Command

Adds an entry for sub-directory LEVEL3 in the LEVEL2 directory.

Use this example if your current directory is LEVEL2:

```
MD LEVEL3
```

This command will do the same thing as the previous example. Note that in the previous example, the first \ tells DOS to begin its directory search with the *root* directory. The absence of a leading \ in the last example causes DOS to begin at the *current* directory. Each directory may contain the names of still other directories.

> **Note:** You can create as many sub-directories as you wish, limited only by available disk space. However, you should ensure that the maximum length of any single path from the root directory to the desired level is 63 characters, including imbedded backslashes.

MODE Command

Purpose: Sets the mode of operation on a printer or on a display connected to the Color/Graphics Monitor Adapter, sets options for an Asynchronous Communications Adapter, or causes printer output to be routed to an Asynchronous Communications Adapter.

> **Technical Note:** When used in Option 1, 3, or 4, the MODE command causes printer and Asynchronous Communications Adapter intercept code to be made resident in memory. This increases the size of DOS in memory by approximately 256 bytes.

Format: MODE LPT#:[*n*][,[*m*][,P]]

or

MODE *n*

or

MODE [*n*],*m*[,T]

or

MODE COM*n*:*baud*[,*parity*[,*databits*[,*stopbits*[,P]]]]

or

MODE LPT#:=COM*n*

MODE Command

Type: Internal External
***

Remarks: A missing or invalid *n* or *m* parameter means that the mode of operation for that parameter is not changed. The MODE command has four format options:

Option 1 (For the printer)

MODE LPT#:[*n*][,[*m*][,P]]

where:

#	is 1, 2, or 3 (the printer number)
n	is 80 or 132 (characters per line)
m	is 6 or 8 (lines per inch vertical spacing)
P	specifies continuous retry on time-out errors

For example:

```
MODE LPT1:132,8
```

sets the mode of operation of printer 1 to 132 characters per line and 8 lines per inch vertical spacing. The power-on default options for the printer are 80 characters per line and 6 lines per inch.

The retry loop can be stopped by pressing Ctrl-Break. To stop time-out errors from being continuously retried when you have entered **P**, you must use MODE Option 1 without specifying **P**.

Option 2 (For switching Display Adapters, and setting the display mode of the Color/Graphics Monitor Adapter)

MODE *n*

or

MODE [*n*],*m*[,T]

where:

n	is 40, 80, BW40, BW80, CO40, CO80, or MONO
40	sets the display width to 40 characters per line (for Color/Graphics Monitor Adapter).
80	sets the display width to 80 characters per line (for Color/Graphics Monitor Adapter).
BW40	switches the active display adapter to the Color/Graphics Monitor Adapter, and sets the display mode to Black and White (disables color) with 40 characters per line.

MODE Command

BW80 switches the active display adapter to the Color/Graphics Monitor Adapter, and sets the display mode to Black and White (disables color) with 80 characters per line.

CO40 switches the active display adapter to the Color/Graphics Monitor Adapter, enables color, and sets the display width to 40 characters per line.

CO80 switches the active display adapter to the Color/Graphics Monitor Adapter, enables color, and sets the display width to 80 characters per line.

MONO switches the active display adapter to the Monochrome Display Adapter (which always has display width of 80 characters per line).

m is R or L (shift display right or left).

T requests a test pattern used to align the display.

For readability, you can shift the display one character (for 40 columns) or two characters (for 80 columns) in either direction. If you specify T in the MODE command, a prompt asks you if the screen is aligned properly. If you enter Y the command ends. If you enter N the shift is repeated followed by the same prompt. For example,

```
MODE 80,R,T
```

sets the mode of operation to 80 characters per line and shifts the display two character positions to the right. The test pattern is displayed to give you the opportunity to further shift the display without having to enter the command again.

Option 3 (For Asynchronous Communications Adapter)

MODE COM*n*:*baud*[,*parity*[,*databits*[,*stopbits*[,P]]]]

where:

n — Either 1 or 2 (Asynchronous Communications Adapter number)

baud — 110, 150, 300, 600, 1200, 2400, 4800, or 9600

Note: Only the first two characters are required; subsequent characters are ignored.

parity Either **N** (none), **O** (odd), or **E** (even) - (default = **E**)

databits Either **7** or **8** (default = **7**)

stopbits Either **1** or **2** (if baud equals 110, default = **2**; if baud does not equal 110, default = **1**)

These are the *protocol* parameters. They are used to initialize the Asynchronous Communications Adapter. When you specify the protocol, you must specify at least the baud rate. The other parameters can be omitted, with the defaults accepted, by entering only commas. For example:

```
MODE COM1:12,N,8,1,P
```

sets the mode of operation to 1200 baud rate, no parity, eight databits, and one stopbit. To use the defaults listed in the definitions above, you enter:

```
MODE COM1:12,,,,P
```

The *parity* defaults to even, the *databits* defaults to seven, and the *stopbits* defaults to one.

The P option indicates that the asynchronous adapter is being used for a serial interface printer. If you enter the **P**, time-out errors are continuously retried. You can stop the retry loop by pressing Ctrl-Break. To stop the time-out errors from being continuously retried when you have entered **P**, you must reinitialize the asynchronous adapter without entering the **P**.

Option 4 (To redirect parallel printer output to an Asynchronous Communications Adapter)

MODE LPT#:=COM*n*

where:

Either **1**, **2**, or **3** (printer number)

n Either **1** or **2** (Asynchronous Communications Adapter number)

All output directed to printer LPT# is redirected to the asynchronous adapter *n*.

Notes:

1. Before you can use MODE to redirect parallel printer output to a serial device, you must initialize the Asynchronous Communications Adapter by using Option 3 (see above). If that serial device is a printer, your serial initialization command should also include the P parameter.

2. MODE LPT#:[*n*][,*m*] disables the redirection for the printer designated by the #.

PATH (Set Search Directory) Command

Purpose: Causes specified directories to be searched for commands or batch files that were not found by a search of the current directory.

Format: PATH [[*d:*]*path*[[;[*d:*]*path*]...]]

Type: Internal *** External

Remarks: You may specify a list of drives and path names, separated by semicolons (note that path names must be specified and will not default to current directory). Then, when you enter a command that is not found in the current directory of the drive that was specified (or implied) with the command, DOS searches the named directories in the sequence you entered them. The current directory is not changed.

Entering PATH with no parameters causes DOS to display the names that were specified on a previous PATH command (that is, the search paths currently defined to DOS). Entering PATH with only a semicolon (PATH;) resets the search path to null (no extended search path). This is the default when DOS is started. In this case, DOS searches only the current directory for commands and batch files.

PATH (Set Search Directory) Command

Example: In this example, assume the program MYPROG.COM resides only in directory MYDIR on drive B, and that the default drive is drive A:

```
PATH LEVEL2; LEVEL2 LEVEL3;B: MYDIR
```

This command instructs DOS to look in the current directory of the drive specified, followed by A: LEVEL2, then A: LEVEL2 LEVEL3 then B: MYDIR until it finds the command you have entered. If the command entered is *not* found in any of the directories specified in PATH, the message **Bad command or filename** is displayed.

In the previous example, if you enter the command:

```
MYPROG
```

PATH (Set Search Directory) Command

DOS searches four directories, finding the program MYPROG in B:\MYDIR.

Notes:

1. Erroneous information in the paths, such as invalid drive specifications or imbedded delimiters, will not be detected until DOS actually needs to search the specified paths.

2. If a path is specified that no longer exists at the time DOS uses it to search for a command or batch file, DOS ignores that path and goes on to the next.

PRINT Command

Purpose: Prints a queue (list) of data files on the printer while you are doing other tasks on the computer. Up to 10 filenames can be queued for printing at one time. The first time this command is issued, it increases the resident size of DOS in memory by approximately 3200 bytes.

Format: PRINT [[*d*:][*filename*[.*ext*]][/T][/C] [/P]...]

Type: Internal External
***

Remarks: You can enter more than one filename on the command line, each with appropriate parameters. Global filename characters * and ? are allowed in the filename and extension. Only files in the current directory can be queued for printing. Once a file has been queued, you can change the current directory without affecting the printing of the files already in the print queue.

/T sets the terminate mode. All queued files are canceled from the print queue. If a file is currently being printed, the printing stops, a cancellation message is printed, the paper is advanced to the next page, and the printer's alarm sounds.

/C sets the cancel mode. Allows you to select which file or files to cancel. The preceding filename and all following filenames entered on the command line are canceled from the print queue until a /P is found on the command line, or the Enter key is pressed.

/P sets the print mode. The preceding filename and all following filenames are added to the print queue until a /C is found on the command line, or the Enter key is pressed.

Global filename characters * and ? are allowed in the filename and extension. You can enter more than one filename on the command line, each with appropriate parameters.

If no / parameters are specified and the Enter key is pressed, the files listed on the command line are queued for printing (/P is assumed).

If PRINT is entered with no parameters, PRINT displays the names of the files currently in the print queue.

The first time the PRINT command is executed after you start your system, the following message is displayed on the display screen:

Name of list device [PRN]:

PRINT Command

This allows you to specify the output list device, LPT1, LPT2, LPT3, PRN, COM1, COM2, AUX, etc. The default is PRN, and will be selected if you press Enter.

> **Note:** Be sure the device you name is physically attached to your system; naming a nonexistent device will cause unpredictable system behavior.

The files are queued for printing in the order entered. After each file is printed, the printer paper is advanced to the next page. Any tab characters found are expanded with blanks to the next 8-column boundary.

If PRINT encounters a disk error while attempting to read the file to be printed, PRINT will cause:

- The file currently printing to be canceled
- The disk error message to be printed on the printer
- The printer paper to be advanced to the next page and the alarm to be sounded
- The remaining files in the print queue to be printed

If the /T or /C parameters are used to cancel a file or files currently being printed:

- The printer alarm sounds.
- A file cancellation message prints on the printer. If /T, **All files canceled by operator.** If /C, the name of the file followed by **File canceled by operator,** where File is the name of the file.
- The printer paper advances to the next page.
- If all files in the print queue have not been canceled, printing resumes with the first file remaining in the print queue.

PRINT Command

Notes:

1. The disk containing the files being printed must remain in the specified drive until all printing is complete. Any file in the print queue must not be altered or erased until after it has been printed.

2. The printer cannot be used for any other purpose while PRINT has data to print. Any attempt to use the printer (Shift-PrtSc, LLIST, LPRINT, etc.) results in an "out-of-paper" or "time-out" indication until all files have been printed or printing is terminated (/T). Using Ctrl-PrtSc will result in a "not ready" error message. You should press Ctrl-PrtSc again and reply A to the error message.

Example: In this example, the PRINT command is being used for the first time since system was started. The command:

```
PRINT a:temp1.tst
```

has just been entered, DOS responds with:

```
Name of list device [PRN]:
```

Press the Enter key to send output to the printer.

Then it adds the file TEMP1.TST from drive A to the print queue and sends the output to the device "PRN" printer. The command:

```
PRINT /T
```

empties the print queue. Any other information on the line is ignored. The command:

```
PRINT temp.*/C
```

removes all TEMP.??? files from the print queue that have the same drive letter as the default drive. The command:

```
PRINT a:temp1.tst/C a:temp2.tst a:temp3.tst
```

removes the three files TEMP1, 2, and 3 on drive A from the print queue. The command:

```
PRINT temp1.tst/C temp2.tst/P temp3.tst
```

removes file TEMP1.TST from the print queue, adds the files TEMP2.TST and TEMP3.TST to the print queue. The command:

```
PRINT temp1.tst temp2.tst temp3.tst/C
```

adds files TEMP1.TST and TEMP2.TST to the print queue, then removes TEMP3.TST from the print queue.

RECOVER Command

Purpose: Recovers files from a disk that has developed a defective sector. You can recover the file that contains the bad sector (minus the data in the bad sector), or all the files on the disk can be recovered if the directory has been damaged.

Format: RECOVER [*d*:][*path*]*filename* [.*ext*]

or

RECOVER *d*:

Type: Internal External
***

Remarks: The file named by *filename* is the file to be recovered. If you do not specify a drive, the default drive is used. If you do not specify a path, the current directory is used. The size of the recovered file is a multiple of the DOS allocation unit size. In most cases, this is larger than the original file size. Text files will normally require re-editing to remove unwanted data from the end of the recovered file before they can be used for normal processing.

In the second format shown, RECOVER assumes the directory is damaged, and recovers all files on the specified disk.

If the global filename characters * and ? are used in the filename or extension, only the first file that matches the filespec will be recovered. RECOVER only recovers one file at a time when a filespec is entered.

Example: For this example assume the disk file to be recovered is MYPROG:

```
RECOVER A:MYPROG
```

This command causes the disk file MYPROG on drive A to be read sector-by-sector, skipping the bad sectors. The bad sectors are allocated in a system table, thus preventing future allocations of that sector. The filename is not changed.

The following example shows how to recover the contents of an entire disk from drive A:

```
RECOVER A:
```

This command causes the disk file allocation table on drive A to be scanned for chains of allocation units. A new root directory is created for each chain of allocation units in the form:

FILE*nnnn*.REC

Where *nnnn* is a sequential number starting with 0001. Each FILE*nnnn*.REC points to one of the recovered files on the disk.

> **Note:** This form of the RECOVER command should only be used if the directory of the disk has become unusable. Because RECOVER has no way to know whether the data in the directory is valid or not, it *must* assume that the entire directory is invalid, and therefore recovers all files into filenames of the form shown above, including any files for which there may still have been valid directory entries.

RENAME (or REN) Command

Purpose: Changes the name of the file specified in the first parameter to the name and extension given in the second parameter. If a valid drive is specified in the second parameter, the drive is ignored.

Format: REN[AME] [*d*:][*path*]*filename*[.*ext*] *filename*[.*ext*]

Type: Internal External
***

Remarks: You can use the abbreviated form REN for the RENAME command. You can also use the global characters ? and * in the parameters. For more information about global characters, refer to "Global Filename Characters" in this chapter. A path can be specified only with the first filename; the file will remain in the same directory after its name has been changed.

RENAME (or REN) Command

Example: The command:

```
RENAME B:ABODE HOME
```

renames the file ABODE on drive B to HOME.

The command:

```
REN B:ABODE *.XY
```

renames the file ABODE on drive B to ABODE.XY.

The command:

```
REN B: LEVEL2 MYPROG.COM MYPROG1.COM
```

renames the file MYPROG.COM in directory LEVEL2 on drive B to filename MYPROG1.COM.

RESTORE (Fixed Disk) Command

Purpose: Restores one or more files from diskettes to a fixed disk.

Format: RESTORE *d:* [*d:*][*path*][*filename*][*.ext*][/S][/P]

Type: Internal External
***

Remarks: The files being restored must have been placed on the diskettes by the BACKUP command. The first parameter you specify is the backup diskette drive. The second parameter is the fixed disk file you want to restore.

Files are restored to the current directory if you do not specify a path. If you do not specify a filename or extension, then all files backed up from the directory will be restored.

Global filename characters are allowed in the filename, and will cause all of the files matching the filename to be restored. For example, entering:

```
RESTORE A: C:*.DAT
```

restores each file from the backup diskettes with an extension of .DAT that had been backed up from the current directory.

RESTORE (Fixed Disk) Command

The parameter /S causes backed up files in all sub-directories to be restored in addition to the files in the specified directory itself. This includes sub-directories at all levels beyond the specified directory.

The parameter /P will cause RESTORE to prompt you before restoring files that have changed since they were last backed up, or that are marked read-only. You can then choose to restore the file or not. Read-only is a file attribute that an application can set by interfacing with DOS internally. The two DOS system files (IBMBIO.COM and IBMDOS.COM) are marked read-only when they are created by the FORMAT and SYS commands.

The following example restores all files on the backup diskettes to fixed disk drive C:

```
RESTORE A: C:\ /S
```

The next example restores three different files from the backup diskettes to the default fixed disk drive:

```
RESTORE A: \level1\file1.dat
RESTORE A: \level1\level2\file2.dat
RESTORE A: \level1\level3\file3.dat
```

RESTORE (Fixed Disk) Command

When RESTORE prompts you to insert the backup diskette, make sure you insert the first diskette that might contain the file you want to restore. If you are not sure, insert backup diskette number one. If the file is not on the diskette you inserted, RESTORE will prompt you to insert the next diskette.

If you used global filename characters, RESTORE will prompt you to insert the next diskette after it has restored all files on the backup diskette that match the specified filename.

The RESTORE command sets the ERRORLEVEL (see Batch Commands) as follows:

0 Normal completion

1 No files were found to restore

3 Terminated by user (Ctrl-Break or ESC)

4 Terminated due to error

These codes can be used with the batch processing IF subcommand to control subsequent error level processing.

RMDIR (Remove Directory) Command

Purpose: Removes a sub-directory from the specified disk.

RMDIR [*d*:]*path*

or

RD [*d*:]*path*

Type: Internal External
***

Remarks: The directory must be empty before it can be removed with the exception of the "." and ".." entries. The last directory name in the path is the directory to be removed.

Example: In this example, the command:

```
RD B:\LEVEL2\LEVEL3
```

removes the entry for LEVEL3 from directory LEVEL2.

> **Note:** The root directory and the current directory cannot be removed.

SYS (System) Command

Purpose: Transfers the operating system files from the default drive to the specified drive.

Format: SYS *d*:

Type: Internal External
***

Remarks: The directory of the disk in the specified drive must be completely empty, or the disk must have been formatted by a FORMAT *d*:/S or FORMAT *d*:/B command to contain directory entries for the DOS files IBMBIO.COM and IBMDOS.COM. This is necessary because DOS startup requires these files to occupy the first two directory entries, and because IBMBIO.COM must reside on consecutive sectors on the disk.

> **Note:** SYS lets you transfer a copy of DOS to an application program diskette designed to use DOS, but sold without it. In this case, the space required for the DOS files has already been allocated, although the DOS files are not actually present. The SYS command will transfer the files to the allocated space.

TIME Command

Purpose: Permits you to enter or change the time known to the system. Whenever you create or add to a file, the time is recorded in the directory. You can change the time from the console or from a batch file.

Format: TIME [*hh:mm:ss.xx*]

Type: Internal *** External

Remarks: If you enter a valid time with the TIME command, the time is accepted, and the system prompt appears. Otherwise, the TIME command displays the following prompt:

```
Current time is hh:mm:ss.xx
Enter new time: _
```

where:

hh is a one- or two-digit number from 0-23 (representing hours)
mm is a one- or two-digit number from 0-59 (representing minutes)
ss is a one- or two-digit number from 0-59 (representing seconds)
xx is a one- or two-digit number from 0-99 (representing hundredths of a second)

Notes:

1. To leave the time as is, press Enter.

2. If you enter any information (for example, just the hours, and press Enter), the remaining fields are set to zero.

3. Any time is acceptable as long as the digits are within the defined ranges.

4. The valid delimiters within the time are the colon (:) separating the hours, minutes, and seconds, and the period (.) separating the seconds and the hundredths of a second.

5. If you specify an invalid time or delimiter, you receive an **Invalid time** message.

Example: In this example, once you press Enter, the time known to the system is changed to 13:55:00.00.

```
A>TIME
Current time is 00:25:16.65
Enter new time: 13:55 _
```

TREE (Display Directory) Command

Purpose: Displays all of the directory paths found on the specified drive, and optionally lists the files in each sub-directory.

Format: TREE [*d*:] [/F]

Type: Internal External
***

Remarks: If no drive is specified, the default drive is assumed.

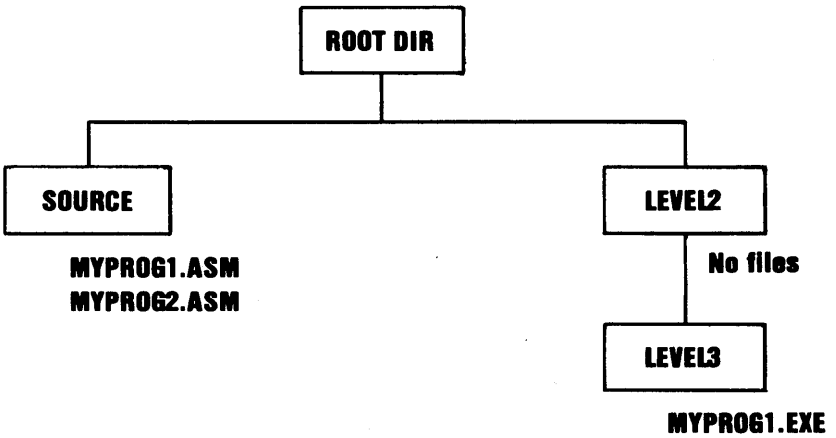
For each directory found, its full path name will be displayed, along with the names of any directories defined within it (these are called sub-directories in the output). If the **/F** parameter is used, the names of all files in each sub-directory will also be displayed.

Example: In this example, the command:

```
TREE B:/F >TREE.LST
```

causes all directories on drive B to be listed. The output will be placed in file TREE.LST in the current directory of drive B, and will contain the names of all sub-directories and files at each directory level.

Following is an example of a directory path listing. If the disk called MYDISK in drive A had the following directory structure:



then TREE would display:

```
DIRECTORY PATH LISTING FOR VOLUME MYDISK

Path: \SOURCE

Sub-directories: None

Files:      MYPROG1 .ASM
            MYPROG2 .ASM


Path: \LEVEL2

Sub-directories: LEVEL3

Files:      None


Path: \LEVEL2\LEVEL3

Sub-directories: None

Files:      MYPROG1 .EXE
```

The following example lists all the sub-directories and filenames from drive A on the printer:

```
TREE A:/F >PRN
```

TYPE Command

Purpose: Displays the contents of the specified file on the screen.

Format: TYPE [*d*:][*path*]*filename*[.*ext*]

Type: Internal External
***

Remarks: The data is unformatted except that tab characters are expanded to an eight-character boundary; that is, columns 8, 16, 24, etc.

TYPE Command

Notes:

1. Press Ctrl-PrtSc if you want the contents of a file to be printed as it is being displayed. You can also redirect the output to a file or the printer.

2. Text files appear in a legible format; however, other files, such as object program files, may appear unreadable due to the presence of nonalphabetic or nonnumeric characters.

3. You must specify a filespec.

4. Global filename characters are not allowed in the filename or extension. If global filename characters are used in the filename or extension, the message **File not found** will appear.

Example: In this example, the file MYPROG.ONE on the diskette in drive B is displayed on the screen:

```
TYPE B:myprog.one
```

VER (Version) Command

Purpose: Displays the DOS version number that you are working with on the display screen.

Format: VER

Type: Internal *** External

Remarks: The DOS version consists of a single-digit major version number, followed by a period, followed by a two-digit minor revision level.

Example:

```
A>VER
IBM Personal Computer DOS Version 2.00
```

VERIFY Command

Purpose: Verifies that the data written on a disk has been correctly recorded.

Format: VERIFY [ON | OFF]

Type: Internal *** External

Remarks: VERIFY ON remains on until it is turned off through the SET VERIFY System Call or a VERIFY OFF command.

When ON, DOS performs a verify operation following each disk write operation, to verify that the data just written can be read without error. Because of the extra time required to perform the verification, the system runs slower when programs write data to disk.

Entering VERIFY with no parameters causes DOS to display the current state (on or off) of the verify feature.

Example: This example causes the verify feature to be turned on:

```
A>VERIFY ON
```

This example displays the current status:

```
A>VERIFY
VERIFY is on

A>
```

VOL (Volume) Command

Purpose: Displays the disk volume identification of the specified drive.

Format: VOL [*d*:]

Type: Internal External
***

Remarks: If you do not specify a drive, the default drive is assumed.

Example:
```
A>VOL
Volume in drive A is MYDISK

A>
```

Summary of DOS Commands

The following chart is provided for quick reference. The section called "Format Notation" at the beginning of this chapter explains the notation used in the format of the commands.

> **Note:** In the column labeled **Type**, the **I** stands for Internal and the **E** stands for External.

Command	Type	Purpose	Format
(Batch)	I	Executes batch file	[*d*:]*filename* [*parameters*]
ECHO	I	Inhibits screen display	ECHO [ON | OFF *message*]
FOR	I	Interactive execution of commands	FOR %%*variable* IN (*set*) DO *command*
GOTO	I	Transfers control to line following the label	GOTO *label*
IF	I	Conditional execution of commands	IF [NOT] *condition command*
PAUSE	I	Provides a system wait	PAUSE [*remark*]

Figure 1 (Part 1 of 2). DOS Batch Processing Commands

Command	Type	Purpose	Format
REM	I	Displays remarks	REM [*remark*]
SHIFT	I	Shift command lines	SHIFT

Figure 1 (Part 2 of 2). DOS Batch Processing Commands

Command	Type	Purpose	Format
ASSIGN	E	Routes requests to a different drive	ASSIGN [*x*=*y* [...]]
BACKUP	E	Backs up fixed disk files	BACKUP [*d*:][*path*] [*filename*][.*ext*]*d*: [/S][/M][/A] [D:*mm-dd-yy*]
BREAK	I	Checks for control break	BREAK [ON | OFF]
CHDIR	I	Change current directory	CHDIR [[*d*:]*path*] or CD [[*d*:]*path*]
CHKDSK	E	Checks disk and reports status	CHKDSK [*d*:][*filename*][.*ext*] [/F][/V]
CLS	I	Clears the display screen	CLS

Figure 2 (Part 1 of 6). DOS Commands

Command	Type	Purpose	Format
COMP	E	Compares files	COMP [*d*:][*path*][*filename* [*.ext*]] [*d*:][*path*][*filename* [*.ext*]]
COPY	I	Copies files	COPY [/A][/B] [*d*:][*path*]*filename* [*.ext*][/A][/B] [*d*:][*path*][*filename* [*.ext*]][/A][/B][/V] or COPY [/A][/B] [*d*:][*path*]*filename* [*.ext*][/A][/B] [+[*d*:][*path*]*filename* [*.ext*][/A][/B]...] [*d*:][*path*][*filename* [*.ext*]][/A][/B][/V]
DATE	I	Enter date	DATE [*mm-dd-yy*]
DIR	I	Lists filenames	DIR [*d*:][*path*] [*filename*[*.ext*]] [/P][/W]
DISKCOMP	E	Compares diskettes	DISKCOMP [*d*:] [*d*:] [/1][/8]
DISKCOPY	E	Copies diskettes	DISKCOPY [*d*:] [*d*:] [/1]

Figure 2 (Part 2 of 6). DOS Commands

Command	Type	Purpose	Format
ERASE	I	Deletes files	ERASE [*d*:][*path*][*filename*] [*.ext*] or DEL [*d*:][*path*][*filename*] [*.ext*]
FORMAT	E	Formats diskette	FORMAT [*d*:][/S] [/1][/8][/V][/B]
GRAPHICS	E	Prints graphics display screen	GRAPHICS
MKDIR	I	Creates a sub-directory	MKDIR [*d*:]*path* or MD [*d*:]*path*

Figure 2 (Part 3 of 6). DOS Commands

Command	Type	Purpose	Format
MODE	E	Sets mode on printer/display	MODE LPT#:[*n*] [,[*m*][,P]] or MODE *n* or MODE [*n*],*m*[,T] or MODE COM*n*: *baud*[,*parity*[,*databits* [,*stopbits*[,P]]]] or MODE LPT#:= COM*n*
PATH	I	Searches directories for commands or batch files	PATH [[*d*:]*path*[;[*d*:] *path*]...]]
PRINT	E	Queues and prints data files	PRINT [[*d*:][*filename*[.*ext*]] [/T][/C][/P]...]

Figure 2 (Part 4 of 6). DOS Commands

Command	Type	Purpose	Format
RECOVER	E	Recovers files from disk or diskette	RECOVER [*d*:][*path*]*filename* [*.ext*] or RECOVER *d*:
RENAME	I	Renames files	REN[AME] [*d*:][*path*]*filename* [*.ext*]*filename*[*.ext*]
RESTORE	E	Restores diskette files to fixed disk	RESTORE *d*: [*d*:][*path*][*filename*] [*.ext*][/S][/P]
RMDIR	I	Removes a sub-directory	RMDIR [*d*:]*path* or RD [*d*:]*path*
SYS	E	Transfers DOS	SYS *d*:
TIME	I	Enter time	TIME [*hh:mm:ss.xx*]
TREE	E	Displays all directory paths	TREE [*d*:][/F]
TYPE	I	Displays file contents	TYPE [*d*:][*path*]*filename* [*.ext*]
VER	I	Displays version number	VER

Figure 2 (Part 5 of 6). DOS Commands

Command	Type	Purpose	Format
VERIFY	I	Verifies data	VERIFY [ON | OFF]
VOL	I	Displays volume identification	VOL [*d*:]

Figure 2 (Part 6 of 6). DOS Commands

Chapter 7. The Line Editor (EDLIN)

Contents

Notes:

Introduction

In this chapter, you will learn how to use the Line Editor (EDLIN) program.

You can use the Line Editor (EDLIN) to create, change, and display source files or text files. Source files are unassembled programs in source language format. Text files appear in a legible format.

EDLIN is a line text editor that you can use to:

- Create new source files and save them
- Update existing files and save both the updated and original files
- Delete, edit, insert, and display lines
- Search for, delete, or replace text within one or more lines

The text of files created or edited by EDLIN is divided into lines of varying length, up to 253 characters per line.

Line numbers are generated and displayed by EDLIN during the editing process, but are not actually present in the saved file.

When you insert lines, all line numbers following the inserted text advance automatically by the number of lines inserted. When you delete lines, all line numbers following the deleted text decrease automatically by the number of lines deleted. Consequently, line numbers always go consecutively from 1 through the last line number.

> **Note:** EDLIN will erase the original backup copy (.BAK) of the file when you issue an **E** (end edit) command, or if the disk space is required during the editing session to satisfy a **W** (write lines) command.

How to Start the EDLIN Program

To start EDLIN, enter:

EDLIN [*d:*][*path*]filename[.*ext*][/B]

Editing an Existing File

If the specified file exists on the designated or default drive, the file is loaded into memory until memory is 75% full. If the entire file is loaded, the following message and prompt is displayed:

```
End of input file
*_
```

You can then edit the file.

> **Note:** If you have not used the /B parameter, EDLIN will stop loading the file when the first Ctrl-Z is encountered in the file's text. If you wish to edit a file that is known to contain embedded Ctrl-Z characters (end-of-file marks), you should use the /B parameter. EDLIN will then process the entire file regardless of any embedded end-of-file marks.

Notice that the prompt for EDLIN is an asterisk (*).

If the entire file cannot be loaded into memory, EDLIN loads lines until memory is 75% full, then displays the * prompt. You can then edit the portion of the file that is in memory.

To edit the remainder of the file, you must write some of the edited lines to diskette in order to free memory so that you can load unedited lines from diskette into memory. Refer to the Write Lines and Append Lines commands in this chapter for the procedure you will use.

Editing a New File

If the specified file does not exist on the drive, a new file is created with the specified name. The following message and prompt are displayed:

```
New file
*_
```

You can now create a new file by entering the desired lines of text. To begin entering text, you must enter an I command to insert lines.

When you have completed the editing session, you can save the original and updated (new) files by using the End Edit command. The End Edit command is discussed in this chapter in the section called "The EDLIN Commands." The original file is renamed to an extension of .BAK, and the new file has the filename and extension you specified in the EDLIN command.

> **Note:** You cannot edit a file with a filename extension of .BAK with EDLIN because the system assumes it is a backup file. If you find it necessary to edit such a file, rename the file to another extension; then start EDLIN and specify the new name.

The EDLIN Command Parameters

Parameter	Definition
line	Denotes when you must specify a line number. There are three possible entries that you can make using this parameter: 1. Enter a decimal integer from 1-65529. If you specify a number greater than the number of lines that are in memory, the line will be added after the last line that exists. Line numbers must be separated from each other by a comma or space. OR 2. Enter a pound sign (#) to specify the line after the last line in memory. Entering a # has the same effect as specifying a number greater than the number of lines in memory.

Parameter	Definition
line	OR 3. Enter a period (.) to specify the current line. The current line indicates the location of the last change to the file, but it is not necessarily the last line displayed. The current line is marked by an asterisk (*) between the line number and the first character of text in the line. For example: 10:*FIRST CHARACTER OF TEXT
n	Denotes when you must specify lines. Enter the number of lines that you want to write to diskette or load from diskette. You only use this parameter with the Write Lines and Append Lines commands. These commands are meaningful only if the file to be edited is too large to fit in memory.
string	Denotes when you must enter one or more characters to represent text to be found, replaced, deleted, or to replace other text. You only use this parameter with the Search Text and Replace Text commands.

The EDLIN Commands

This section describes the EDLIN commands and tells how to use them. The commands are in alphabetical order; each with its purpose, format and remarks. Examples are provided where appropriate.

Information Common to All EDLIN Commands

The following information applies to all EDLIN commands:

- With the exception of the Edit Line command, all commands are a single letter.
- With the exception of the End Edit and Quit Edit commands, commands are usually preceded and/or followed by parameters.
- Enter commands and string parameters in uppercase or lowercase, or a combination of both.
- Separate commands and parameters with delimiters for readability; however, a delimiter is only required between two adjacent line numbers. Remember, delimiters are spaces or commas.
- Commands become effective only after you press the Enter key.

- Stop commands by pressing the Ctrl-Break keys.

- For commands producing a large amount of output, press Ctrl-Num Lock to suspend the display so that you can read it before it scrolls away. Press any other character to restart the display.

- Use the control keys and DOS editing keys, described in Chapter 3, while using EDLIN. They are very useful for editing *within a line*, while the EDLIN commands can be used for editing operations on *entire lines*.

- The prompt from EDLIN is an asterisk (*).

- It is possible to refer to line numbers relative to the current line. Use a Minus (-) sign and a number to indicate a line before the current line. Use a Plus (+) sign and a number to indicate a line after the current line. For example:

  -10,+10L

  This command displays 10 lines before the current line, the current line, and 10 lines after the current line.

- Multiple commands can be entered on one command line. When you enter the command to edit a single line using [*line*], you must use a semicolon to separate the commands on the line. In the case of the Search or Replace command the [*string*] can be terminated by Ctrl-Z (F6) instead of the Enter key. Otherwise, one command can follow another without any special delimiting characters. For example:

  15;-5,+5L

  edits line 15 and then displays lines 10 through 20 on the screen.

- Control characters can be inserted into the text, or can be used in the strings for the Search text and Replace text commands. To enter a control character, press Ctrl-V, then enter the desired control character in uppercase. For example, the sequence Ctrl-V, followed by Z generates the control character Ctrl-Z.

Append Lines Command

Purpose: Adds the specified number of lines from disk to the file being edited in memory. The lines are added at the end of the current lines in memory.

Format: [*n*]A

Remarks: This command is only meaningful if the file being edited is too large to fit in memory. As many lines as possible are read into memory for editing when you start EDLIN.

To edit the remainder of the file that will not fit into memory, you must write edited lines in memory to disk before you can load unedited lines from disk into memory by using the Append Lines command. Refer to the Write Lines command for information on how to write edited lines to disk.

Notes:

1. If you do not specify the number of lines, lines are appended to memory until available memory is 75% full. No action is taken if available memory is already 75% full.

2. The message **End of input file** is displayed when the Append Lines command has read the last line of the file into memory.

Copy Lines Command

Purpose: Copies the lines in the specified range to the line number specified by the third parameter. The new data is placed ahead of the line that was specified in the third parameter. This third parameter is not optional. The operation is repeated the number of times specified in *count*.

Format: [*line*],[*line*],*line*[,*count*]C

Remarks: The parameter *count* defaults to 1. To repeat text specify the number of times the operation is to be performed in *count*. If the first parameter or the second parameter is omitted, the default is the current line. This effectively copies the current line to the specified line. The file is renumbered accordingly. The first of the copied lines becomes the current line. For example:

```
1,5,8C
```

copies lines 1 through 5 to line 8. Line 8 becomes the current line.

> **Note:** The line numbers must not overlap or an error is reported. Also, the characters – and + are not allowed in the *count* field.

Delete Lines Command

Purpose: Deletes a specified range of lines.

Format: [*line*][,*line*]D

Remarks: The line following the deleted range becomes the current line, even if the deleted range includes the last line in memory. The current line and any following lines are renumbered.

Default values are supplied if either one or both of the parameters are omitted.

If you omit the first parameter, as in:

,*line*D

deletion starts with the current line and ends with the line specified by the second parameter. The beginning comma is required to indicate the omitted first parameter.

Delete Lines Command

If you omit the second parameter, as in:

***line*D**

or

***line*,D**

only the one specified line is deleted. If you omit both parameters, as in:

D

only the current line is deleted, and the line that follows becomes the current line.

Example: Assume that you want to edit the following file. The current line is line 29.

```
 1: This is a sample file
 2: used to demonstrate
 3: line deletion
 4: and dynamic
 5: line number generation.
 •
 •
 •
25: See what happens
26: to the lines
27: and line numbers
28: when lines are
29:*deleted.
```

Delete Lines Command

If you want to delete a range of lines, from 5-25, enter:

```
5,25D
```

The result is:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: and dynamic
5:*to the lines
6: and line numbers
7: when lines are
8: deleted.
```

Lines 5-25 are deleted from the file. Lines 26-29 are renumbered to 5-8. Line 5 becomes the current line. If you want to delete the current and the following line, enter:

```
,6D
```

The result is:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: and dynamic
5:*when lines are
6: deleted.
```

Lines 5-6 are deleted from the file. Lines 7-8 are renumbered to 5-6. Line 5 is still the current line, but now it has different text.

Delete Lines Command

If you want to delete a single line, say line 2, enter:

```
2D
```

The result is:

```
1: This is a sample file
2:*line deletion
3: and dynamic
4: when lines are
5: deleted.
```

Line 2 is deleted. Lines 3-6 are renumbered to 2-5. The new line 2 becomes the current line. If you want to delete only the current line, enter:

```
D
```

The result is:

```
1: This is a sample file
2:*and dynamic
3: when lines are
4: deleted.
```

The current line, line 2, is deleted. Lines 3-5 are renumbered to 2-4. The new line 2 becomes the current line.

Edit Line Command

Purpose: Allows you to edit a line of text. You must enter the line number of the line to be edited, or enter a period (.) to indicate the current line.

Format: [*line*]

Remarks: If you just press Enter, you specify that the line after the current line is to be edited.

The line number and its text are displayed and the line number is repeated on the line below.

You can use the control keys and the editing keys, described in Chapter 3, to edit the line, or you can replace the entire line by typing new text.

When you press the Enter key, the edited line is placed in the file and becomes the current line.

If you decide not to save the changed line, press either Esc or Ctrl-Break instead of Enter. The original line remains unchanged. Pressing the Enter key with the cursor at the beginning of the line has the same effect as pressing Esc or Ctrl-Break.

If the cursor is in any position other than the beginning or the end of a line, pressing Enter erases the rest of the line.

Edit Line Command

Example: Assume that you want to edit line 6. The following display would appear on the screen:

```
*6
   6: This is a sample unedited line.
   6: _
```

The first line is your request to edit line 6, followed by the two-line display response.

If you want to move the cursor to the letter **u**, press F2 and enter:

```
u
```

The result is:

```
*6
   6: This is a sample unedited line.
   6: This is a sample_
```

If you want to delete the next two characters and keep the remainder of the line, press Del twice; then press F3.

The result is:

```
*6
   6: This is a sample unedited line.
   6: This is a sample edited line._
```

Edit Line Command

Now you can take one of the following actions:

- Press Enter to save the changed line.
- Extend the changed line by typing more text. You are automatically in insert mode when the cursor is at the end of a line.
- Press F5 to do additional editing to the changed line without changing the original line.
- Press Esc or Ctrl-Break to cancel the changes you made to the line. The original contents of the line will be preserved.

End Edit Command

Purpose: Ends EDLIN and saves the edited file.

Format: E

Remarks: The edited file is saved by writing it to the drive and filename specified when you started EDLIN.

The original file, the one specified when EDLIN was started, is given a .BAK filename extension. A .BAK file will not be created if there is no original file; that is, if you created a new file instead of updating an old file during the editing session.

End Edit Command

EDLIN returns to the DOS command processor, which displays the command prompt.

Notes:

1. Be sure your disk has enough free space to save the entire file. If your disk does not have enough free space, only a portion of the file is saved. The portion in memory that is not written to disk is lost. In this case, your original file will not be renamed to .BAK, and the portion of data that was written to disk will have a filename extension of $$$.

2. EDLIN appends a carriage return, line feed sequence to the end of the file if they were not already present, to delimit the last line of text in the file. Also, a Ctrl-Z character is added as the last character in the saved file. This serves as an end-of-file mark.

Insert Lines Command

Purpose: Inserts lines of text immediately *before* the specified line. When you create a new file, you must enter the Insert Lines command before text can be inserted.

Format: [*line*]I

Remarks: If you do not specify line, or if you specify line as a period (.), the insert is made immediately before the current line.

If the line number you specify is greater than the highest existing line number, or if you specify # as the line number, the insertion is made after the last line in memory.

EDLIN displays the appropriate line number so that you can enter more lines, ending each line by pressing Enter. During the insert mode of operation, successive line numbers appear automatically each time Enter is pressed.

You must press Ctrl-Break to discontinue the insert mode of operation.

The line that follows the inserted lines becomes the current line, even if the inserted lines are added to the end of the lines in memory. The current line and any remaining lines are renumbered.

Insert Lines Command

Example: Assume that you want to edit the following file. Line 3 is the current line:

```
1: This is a sample file
2: used to demonstrate
3:*line deletion
4: and dynamic
5: line number generation.
```

If you want to insert text before line 4, the entry and immediate response look like this:

```
*4I
   4:* _
```

Now, if you want to insert two new lines of text, enter:

```
*4 I
   4:*First new line of text
   5:*Second new line of text
   6:*
```

and press Ctrl-Break.

The original lines 4 and 5 are now renumbered to lines 6 and 7.

Insert Lines Command

If you display the file with a List Lines command, the file looks like this:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: First new line of text
5: Second new line of text
6:*and dynamic
7: line number generation.
```

If the two lines that were inserted had been placed at the beginning of the file, the screen would look like this:

```
1: First new line of text
2: Second new line of text
3:*This is a sample file
4: used to demonstrate
5: line deletion
6: and dynamic
7: line number generation.
```

If the two lines that were inserted had been placed immediately before the current line (3 I or . I or I), the screen would look like this:

```
1: This is a sample file
2: used to demonstrate
3: First new line of text
4: Second new line of text
5:*line deletion
6: and dynamic
7: line number generation.
```

If the two inserted lines had been placed at the end of the file (6 I or # I), the screen would look like this:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: and dynamic
5: line number generation.
6: First new line of text
7: Second new line of text
```

List Lines Command

Purpose: Displays a specified range of lines.

The current line remains unchanged.

Format: [*line*][,*line*]L

Remarks: Default values are provided if either one or both of the parameters are omitted.

If you omit the first parameter, as in:

,*line*L

the display starts 11 lines before the current line and ends with the specified line. The beginning comma is required to indicate the omitted first parameter.

> **Note:** If the specified line is more than 11 lines before the current line, the display is the same as if you omitted both parameters. (An example is provided in this section showing both parameters omitted.)

List Lines Command

If you omit the second parameter, as in:

***line*L**

or

***line*,L**

a total of 23 lines are displayed, starting with the specified *line*.

If you omit both parameters, as in:

L

a total of 23 lines are displayed - the 11 lines before the current line, the current line, and the 11 lines after the current line. If there aren't 11 lines before the current line, then extra lines are displayed after the current line to make a total of 23 lines.

List Lines Command

Example: Assume that you want to edit the following file. Line 15 is the current line.

```
 1: This is a sample file
 2: used to demonstrate
 3: line deletion
 4: and dynamic
 5: line number generation.
 •
 •
 •
15:*This is the current line (note the asterisk)
 •
 •
 •
25: See what happens
26: to the lines
27: and line numbers
28: when lines are
29: deleted.
```

If you want to display a range of lines, from 5-25, enter:

```
5,25L
```

The screen looks like this:

```
 5: line number generation.
  •
  •
  •
15:*This is the current line (note the asterisk)
  •
  •
  •
25: See what happens
```

If you want to display the first three lines, enter:

```
1,3L
```

The screen looks like this:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
```

If you want to display 23 lines of the file, starting with line 3, enter:

```
3L
```

The screen looks like this:

```
 3: line deletion
 4: and dynamic
 5: line number generation.
 •
 •
 •
15:*This is the current line (note the asterisk)
 •
 •
 •
25: See what happens
```

If you want to display 23 lines centered around the current line, enter:

```
L
```

The screen looks like this:

```
 4: and dynamic
 5: line number generation.
 •
 •
 •
15:*This is the current line (note the asterisk)
 •
 •
 •
25: See what happens
26: to the lines
```

Move Lines Command

Purpose: Moves the range of lines specified by the first two *line* parameters ahead of the line specified in the third *line* parameter. The third parameter is not optional.

Format: [*line*],[*line*],*line*M

Remarks: Use this command to move a block of data from one location in the file to another. If the first or second line parameter is omitted, it will default to the current line. After the move, the first of the moved lines becomes the current line. The lines are renumbered according to the direction of the move. For example:

```
,+25,100M
```

moves the data from the current line plus 25 lines to line 100. If the arguments overlap an entry error is reported.

Page Command

Purpose: Lists the specified block of lines.

Format: [*line*][,*line*]P

Remarks: If the first *line* parameter is omitted, it defaults to the current line plus one. If the second *line* parameter is omitted, 23 lines are listed. The new current line becomes the last line displayed by the Page command and is marked with an asterisk. This command pages through a file displaying 23 lines at a time. It differs from the List Lines command in that it changes the current line.

Quit Edit Command

Purpose: Quits the editing session without saving any changes you may have entered.

Format: Q

Remarks: EDLIN prompts you to make sure you really don't want to save the changes.

Enter **Y** if you want to quit the editing session. No editing changes are saved and no .BAK file is created. Refer to the End Edit command for information about the .BAK file.

Enter **N**, or any other character, if you want to continue the editing session.

Example:

```
Q
Abort edit (Y/N)?_
```

Replace Text Command

Purpose: Replaces all occurrences of the first string in the specified range of lines with the second string.

Notes:

1. If you omit the second string, Replace Text deletes all occurrences of the first string within the specified range of lines. If you omit both strings, EDLIN will re-use the search string entered with the most recent (previous) S or R command, and the Replace Text string entered with the last R command.

2. This command uses the F6 key as normally setup by DOS. If you have changed the meaning of the F6 key through "Extended Keyboard Control" (see Chapter 13), you should press Ctrl-Z where F6 is referred to below.

EDLIN displays the changed lines each time they are changed. The last line changed becomes the current line.

Format: [*line*][,*line*][?]R[*string*][<F6>*string*]

Replace Text Command

Remarks: You can specify the optional parameter ? to request a prompt (O.K.?) after each display of a modified line. Press Y or the Enter key if you want to keep the modification.

Enter any other character if you don't want the modification. In either case, the search continues for further occurrences of the first string within the range of lines, including multiple occurrences within the same line.

Defaults occur if either one or both of the *line* parameters are missing.

If you omit the first *line*, the search begins with the line after the current line. If you omit the second *line*, the search ends with the last line in memory. If you omit both *line* parameters, the system will search from the line following the current line to the last line in memory.

> **Note:** The first string begins with the character in the position immediately following the R, and continues until you press F6 or Ctrl-Z (or the Enter key if the second string is omitted).
>
> The second string begins immediately after you press F6 or Ctrl-Z and continues until you press Enter.

Replace Text Command

Example: Assume that you want to edit the following file. Line 7 is the current line.

```
1: This is a sample file
2: used to demonstrate
3: the Replace and Search Text commands.
4: This includes the
5: optional parameter ?
6: and required string
7:*parameter.
```

To replace all occurrences of **and** with **or** in the lines in memory, enter:

```
1,7 Rand
```

Then press F6, type **or**, and press Enter.

The result is:

```
3: The Replace or Search Text commands
6: or required string
```

Line 6 becomes the current line in the file, because line 6 was the last line changed. Notice that lines 1, 2, 4, 5, and 7 are not displayed because they were not changed.

Greater selectivity can be achieved by requesting a prompt (by using the ? parameter) after each display of a modified line. If you request a prompt, the screen looks like this:

```
*1,7? Rand (Press F6, type or, and press Enter)
      3: the Replace or Search Text commands
O.K.?  Y
      3: the Replace or Search Text commands
O.K.?  N
      6: or required string
O.K.?  Y
*
```

Lines 3 and 6 are displayed like this:

```
3: the Replace or Search Text commands.
6: or required string
```

Search Text Command

Purpose: Searches a specified range of lines in order to locate a specified string.

Format: [*line*][,*line*][?]S[*string*]

Remarks: The first line to contain the specified string is displayed and the search ends (unless you use the ? parameter). The first line found that contains the specified string becomes the current line.

> **Note:** The Search command always searches for the exact same character in text. That is, it searches for UPPERCASE if you enter UPPERCASE, and lowercase if you enter lowercase.

You should specify the optional parameter ? if you would like a prompt (O.K.?) after each display of a line containing the specified string.

Search Text Command

If you do not enter a string, the **S** command will use the last search string that was entered on a Replace or Search command. If the specified string is not found, the search ends and the message **Not found** is displayed. The current line remains unchanged. If you enter **Y** or press the Enter key, the line that matches the specified string becomes the current line and the search ends. Enter any other character to continue the search until another string is found, or until all lines within the range are searched. Once all the lines within the range are searched, the **Not found** message is displayed.

Search Text Command

The system provides default values if you omit the first, second, or both line parameters. If you omit the first line parameter, the system defaults to the line following the current line. If you omit the second line parameter, the system defaults to the last line in memory. If you omit both line parameters, the system searches from the line following the current line to the last line in memory.

Notes:

1. The string begins with the character in the position immediately following the S and continues until you end the string by pressing the Enter key.

2. If you wish to place more than one command on a line containing a Search Text command, the Search Text command should end in a Ctrl-Z (F6), and the next command should begin in the following character position.

Search Text Command

Example: Assume that you want to edit the following file. Line 7 is the current line.

```
1: This is a sample file
2: used to demonstrate
3: the Search Text command.
4: This includes the
5: optional parameter ?
6: and required string
7:*parameter.
```

If you want to search for the first occurrence of **and** in the file, enter:

```
1,7 Sand
   or
1, Sand
   or
1Sand
```

The result is:

```
      3: the Search Text command.
*
```

The **and** is part of the word **command.** Notice that line 3 becomes the current line in the file.

Search Text Command

Perhaps this is not the **and** you were looking for. To continue the search, simply enter the letter S and press Enter. The search will continue with the line following the current line (the line just found).

The screen looks like this:

```
*1,7 Sand
     3: the Search Text command.
*S
     6: and required string
*
```

Line 6 now becomes the current line in the file.

You can also search for strings by requesting a prompt (by means of the ? parameter) after each display of a matching line. In this case, the screen looks like this:

```
*1,7 ? Sand
     3: the Search Text command.
O.K.? N
     6: and required string
O.K.? Y
*
```

Transfer Lines Command

Purpose: Transfers (merges) the contents of a specified file into the file currently being edited.

Format: [*line*]T[*d*:]*filename*

Remarks: The *filename* contents will be inserted ahead of the *line* in the file being edited. If *line* is omitted, then the current line is used.

> **Note:** The file being merged is read from the current directory of the specified or default drive. If a path was specified when you issued the EDLIN command, then that path will be the current directory for that drive for the duration of the EDLIN session, and any Transfer Lines commands for that drive must be satisfied from the same directory.

Write Lines Command

Purpose: Writes a specified number of lines to diskette from the lines that are being edited in memory. Lines are written beginning with the line number 1.

Format: [*n*]W

Remarks: This command is only meaningful if the file you are editing is too large to fit in memory. When you start EDLIN, EDLIN reads lines into memory until memory is 75% full.

To edit the remainder of your file, you must write edited lines in memory to diskette before you can load additional unedited lines from diskette into memory by using the Append Lines command.

> **Note:** If you do not specify the number of lines, lines are written until 25% of available memory is used. No action is taken if available memory is already less than 25% used. All lines are renumbered so that the first remaining line becomes number 1.

Summary of EDLIN Commands

The following chart is provided for quick reference.

> **Note:** The section called "Format Notation" in Chapter 6 explains the notation used in the format of the following commands.

Command	Format
Append Lines	[*n*]A
Copy Lines	[*line*],[*line*],*line*,[,*count*]C
Delete Lines	[*line*][,*line*]D
Edit Line	[*line*]
End Edit	E
Insert Lines	[*line*]I
List Lines	[*line*][,*line*]L
Move Lines	[*line*],[*line*],*line*M
Page	[*line*][,*line*]P
Quit Edit	Q
Replace Text	[*line*][,*line*][?]R[*string*][<F6>*string*]

Figure 3 (Part 1 of 2). EDLIN Commands

Command	Format
Search Text	[*line*][,*line*][?]S[*string*]
Transfer Lines	[*line*]T *filename*
Write Lines	[*n*]W

Figure 3 (Part 2 of 2). EDLIN Commands

Notes:

Chapter 8. Messages

Contents

Notes:

Introduction

This chapter contains two parts; first, *device errors* (the message that DOS uses to indicate errors while reading or writing to devices on your system), and second, *Other messages* (the remainder of the DOS messages) in alphabetical order. Each message is indicated here by **bold** type, and the description follows the message.

The first word of the description of each message is the name of the program or command that generated the message.

Device Error Messages

When an error is detected while reading or writing any of the devices (disk drives, printer, etc.) on your system, DOS will display a message in the following format:

<*type*> **error reading** <*device*>
Abort, Retry, Ignore?
or
<*type*> **error writing** <*device*>
Abort, Retry, Ignore?

In these messages, *<device>* is the name of the device in error, such as **PRN**, or **B:**, and *<type>* is one of the following error types:

Bad call format

A device driver was passed an incorrect length request header. If this occurs, contact the dealer you purchased the device driver from.

Bad command

A device driver has issued an invalid command to *<device>*.

Bad unit

A device driver has been passed an invalid sub-unit number. If this occurs, contact the dealer you purchased the device driver from.

Data

The data was unable to be read or written correctly. This usually means a disk has developed a defective spot.

Disk

An error of a type not described above has occurred.

No paper

The indicated printer is either out of paper or is not turned on.

Non-DOS disk

The file allocation table contains invalid information, and needs to be re-formatted.

Not ready

The named device is not ready, and cannot accept or transmit data.

Read fault

DOS was unable to successfully read the data.

Sector not found

The sector containing the data could not be located on the disk, usually occurring when a defective spot develops on the disk.

Seek

The fixed disk or diskette drive was unable to locate the proper track on the disk.

Write fault

DOS was unable to successfully write the data to the device, from the device.

Write protect

An attempt was made to write on a write protected diskette.

Warning
If any of these messages appear for a diskette drive, DO NOT change diskettes.

The system now waits until *one* of the following responses is made, Enter:

- A for Abort. The system ends the program that requested the disk read or write.

- R for Retry. The system tries the disk read or write operation again.

- I for Ignore. The system pretends the error did not occur and continues the program.

To recover from an error condition, the responses are generally made in the following order:

R to retry the operation because the error may not occur again.

A to abort the program.

I to ignore the error condition and continue the program. (This response is not recommended because data is lost when you use it.)

Note: One of these messages will appear if you attempt to use a dual-sided diskette in a single-sided drive, or if you attempt to use a 9 sector per track diskette on a pre-version 2.00 level of DOS.

Other Messages

The following messages are in alphabetical order.

A

About to generate .EXE file
Change disks <hit ENTER>

LINK. This message is displayed when you specify the /PAUSE parameter. Insert your Runfile diskette into the appropriate drive and press Enter.

Access denied

DEBUG. An attempt was made to write to a file which is marked read-only.

All files canceled by operator

PRINT. This message appears on the printer when you cancel the printing of all queued files via the /T parameter.

All specified file(s) are contiguous

CHKDSK. The file or files you named are all written sequentially on the disk.

Allocation error for file, size adjusted

CHKDSK. A filename precedes this message. An invalid sector number was found in the file allocation table. The file was truncated at the end of the last valid sector.

Ambiguous switch: z

LINK. The characters specified by *z* do not uniquely identify a linker parameter. Use more characters from the parameter name.

Amount read less than size in header

EXE2BIN. The program portion of the file was smaller than indicated in the file's header. You should re-compile or re-assemble, and re-LINK the program.

An internal failure has occurred

LINK. An error has occurred in the linker program. Report the conditions under which the message appeared to your Authorized IBM Personal Computer Dealer.

Attempt to access data outside of segment bounds

LINK. An object file is probably invalid.

Attempted write-protect violation

FORMAT. The diskette being formatted cannot be written on because it is write-protected. You are prompted to insert a new diskette and press a key to restart formatting.

B

Backup file sequence error

RESTORE. A file to be restored was backed up onto more than one diskette. You did not insert the diskette with the first part of the file. Rerun RESTORE and start with the correct diskette.

Bad command or file name

DOS. The command just entered is not a valid command to DOS. You should check your spelling and re-enter the command. If the command name is correct, check to see that the default drive contains the external command or batch file you are trying to execute.

Bad numeric parameter

LINK. The value you specified with the /STACK parameter is not a valid numeric constant.

Bad or missing Command Interpreter

DOS. The disk that DOS is being started from does not contain a copy of COMMAND.COM, or an error occurred while the disk was being loaded. If System Reset fails to solve the problem, copy COMMAND.COM from a backup diskette to the root directory of the disk that failed.

This message also appears if COMMAND.COM has been removed from the directory it was in originally when DOS was started; or if the COMSPEC= parameter in the environment points to a directory not containing COMMAND.COM and DOS is attempting to reload the command processor.

Bad or missing <filename>

DOS. This message appears only at startup, and indicates that a device driver named in a DEVICE= <filename> parameter in the CONFIG.SYS file was not found, or set a break address that was out of bounds for the machine size, or an error occurred while the driver was being loaded. That driver is not installed by DOS.

Batch file missing

DOS. DOS was unable to locate the batch file it had been processing. The file has probably been erased or renamed by one of the steps within it. Batch processing stops and the DOS prompt appears.

BF

DEBUG. Bad flag. An invalid flag code setting was specified. Try the Register (R F) command again with the correct code.

BP

DEBUG. Breakpoints. More than ten breakpoints were specified for the Go command. Try the Go (G) command again with ten or fewer breakpoints.

BR

DEBUG. Bad register. An invalid register name was specified. Try the Register (R) command again with a correct register name.

BREAK is On/Off

DOS. This message indicates that BREAK is on or off.

C

Cannot do binary reads from a device

COPY. You have used the /B parameter with a device name while attempting to copy from the device. The copy cannot be performed in binary mode because COPY must be able to detect end-of-file from the device. You should omit the /B parameter or use the /A parameter after the device name.

Cannot edit .BAK file – rename file

EDLIN. .BAK files are considered to be backup files, with more up-to-date versions of the files assumed to exist. Therefore, .BAK files shouldn't be edited.

If it is necessary to edit the .BAK file, either rename the file, or copy it and give the copy a different name.

Cannot find file *object file*
Change diskette <hit ENTER>

LINK. The linker could not locate the specified object module on the drive. Insert the diskette with the specified module on it and press Enter.

Cannot find library *library file*
Enter new drive letter:

LINK. The specified library could not be found on the drive. Enter the letter for the drive the library is on.

Cannot load COMMAND, system halted

DOS. While attempting to reload the command processor, DOS determined that the area in which it keeps track of available memory has been destroyed, or the command processor could not be found in the path specified by the COMSPEC environment parameter. You should restart DOS.

Cannot nest response file

LINK. You used *@filespec* within an automatic response file. Automatic response files cannot be nested.

Cannot open list file

LINK. The directory or disk is full.

Cannot open overlay

LINK. The directory or disk is full.

Cannot open response file

LINK. The automatic response file could not be found.

Cannot open temporary file

LINK. The directory or disk is full.

Cannot start COMMAND, exiting

DOS. While attempting to load a second copy of the command processor, either the FILES= parameter in the configuration file was found to contain too small a value, or there is insufficient available memory to contain the new copy of COMMAND.COM.

COMn: bbbb,p,d,s,t initialized

MODE. The Asynchronous Communications Adapter has been initialized. The values represent:

n adapter (COM1 or COM2)

bbbb baud rate

p parity

- **e** even
- **o** odd
- **n** none

s stop bits (1 or 2)

t type of serial device

- **p** serial printer (serial timeouts will be retried)
- **–** other serial device (serial timeouts will not be retried)

Compare error at offset xxxxxxxx

COMP. The files being compared contain different values at the displayed offset (in hexadecimal) into the file. The differing values are also displayed in hexadecimal.

Compare error(s) on
Track xx, side xx

DISKCOMP. One or more locations on the indicated track and side contain differing information between the diskettes being compared.

Compare more diskettes (Y/N)?

DISKCOMP. If you wish to compare another pair of diskettes, enter **Y**, and DISKCOMP will prompt you to insert the required diskettes. If you do not want to compare any more diskettes, enter **N**.

Compare more files (Y/N)?

COMP. If you wish to compare the contents of two more files, enter **Y**, and COMP will prompt you for the names of the files to compare. If you do not wish to compare more files, enter **N**.

Comparing *x* sectors per track, n side(s)

DISKCOMP. The **n** will be either 1 or 2, indicating the number of sides that DISKCOMP will compare on the two diskettes. This number is determined by the number of sides DISKCOMP was able to successfully read from the first track of the first diskette. The *x* indicates the number of sectors per track found on the first diskette (8 or 9). If you use **/8**, then the number 8 will appear.

**Contains invalid cluster,
file truncated**

CHKDSK. The file whose name precedes this message contains an invalid pointer to the data area. The file is truncated at the last valid data block if the /F parameter was used.

Contains *xxx* non-contiguous blocks

CHKDSK. The file whose name precedes this message is not written sequentially on disk—it is written in *xxx* pieces on different areas of the disk. This is an information only message, and does not indicate a problem on the disk. Since fragmented files take longer to read, you should consider copying badly fragmented files to another disk. This will record the file sequentially, resulting in better system performance when the file is read.

Convert directory to file (Y/N)?

CHKDSK. The directory whose name appears ahead of this message contains enough invalid information that it is no longer usable as a directory. If you reply Y, CHKDSK will convert the directory to a file so that you may examine it with DEBUG. If you reply N, the entry is not changed.

Convert lost chains to files (Y/N)?

CHKDSK. If you wish to recover the data in the "lost" blocks found by CHKDSK, reply Y. If you reply N, CHKDSK will free the blocks up so they can be allocated to new files.

Copy another (Y/N)?

DISKCOPY. If you wish to copy another entire diskette, enter **Y**; DISKCOPY will prompt you to insert the required diskette. If you do not wish to make another copy, enter **N**.

Copy complete

DISKCOPY. The source diskette contents have been successfully copied to the target diskette.

Copying *x* sectors per track, n side(s)

DISKCOPY. The **n** will be either 1 or 2, indicating the number of sides that DISKCOPY has successfully read from the first track of the source diskette. The *x* will be 8 or 9, indicating the number of sectors per track found on the source diskette.

D

DF

DEBUG. Double flag. Conflicting codes were specified for a single flag. A flag can be changed only once per Register (R F) command.

Disk boot failure

DOS. An error occurred while trying to load DOS into memory. If subsequent attempts to start the system also fail, place a backup DOS diskette in drive A and restart your system.

Disk error writing FAT *x*

CHKDSK. A disk error was encountered which CHKDSK was attempting to update the file allocation table (FAT) on the specified drive. *X* will be 1 or 2, depending on which of the 2 copies of the file allocation table could not be written. If this message appears twice, for FAT's 1 and 2, the disk should be considered unusable.

Disk full – write not completed

EDLIN. An End Edit command ended abnormally because the disk does not have enough free space to save the entire file.

Some of the file may be saved on disk, but the portion in memory not saved is lost.

Disk not compatible

FORMAT. The drive you specified cannot be formatted by the DOS FORMAT command. It is not supported by the IBM device interfaces that FORMAT requires.

Disk unsuitable for system disk

FORMAT. A defective track was detected where the DOS files were to reside. The diskette can be used only for data.

Diskettes compare OK

DISKCOMP. The two diskettes just compared contain identical information.

Diskette is not a backup diskette

BACKUP and RESTORE. The diskette was not created by BACKUP. The first file on a backup diskette is always BACKUPID.@@@. Rerun with the correct diskette.

Divide overflow

DOS. A program attempted to divide a number by zero, or the program had a logic error that caused an internal malfunction. The system simulates Ctrl-Break processing.

Do you see the leftmost 9? (Y/N)

MODE. ,R,T was specified. Respond Y or N. This prompt is repeated until you respond Y.

Do you see the rightmost 9? (Y/N)

MODE. ,L,T was specified. Respond Y or N. This prompt is repeated until you respond Y.

Do you wish to use the entire fixed disk for DOS (Y/N).....?[ ]

FDISK. When the "Create DOS Partition" option is used on the current fixed disk and the fixed disk has never been set up, this question is asked. If you enter Y, the entire current fixed disk will be used for DOS and it will be made active. If you enter N, you will be asked to enter the limits of the DOS partition you want to create.

Dup record too complex

LINK. Problem resides in object module created from an assembler source program. A single DUP requires 1024 bytes before expansion. Debug the source program; then rerun LINK.

Duplicate filename or file not found

RENAME. You tried to rename a file to a filename that already exists on the diskette, or the file to be renamed could not be found on the specified (or default) drive.

E

Enter the number of the partition you want to make active.....: [ ]

FDISK. The "Change Active Partition" option is requesting that you enter the number of the partition you want to make active. Type the number of the partitions that you want to make active on the current fixed disk. They are displayed above the prompt. Then press the Enter key.

Enter partition size.....: [dddd]

FDISK. The "Create DOS Partition" option requests that you enter the size of the partition you wish to create. The number shown in the brackets is the default size. If you only press Enter, that size will be used as the partition size.

Enter primary file name

COMP. Enter the filespec of the first of two files to be compared.

Enter starting cylinder number..: [dddd]

FDISK. The "Create DOS Partition" option is requesting that you enter the starting cylinder number for the DOS partition you are creating. The value in the brackets is the default value. It is the starting cylinder of the largest piece of free space on the current fixed disk. Type a number and press Enter, or just press Enter to use the default value.

Enter 2nd file name or drive id

COMP. Enter the filespec of the second of two files to be compared, or just enter the drive designator if the filename is the same as the primary filename.

Entry error

EDLIN. Correct the syntax error on the last command.

Entry has a bad attribute (or size or link)

CHKDSK. This message may begin with one or two periods, indicating which entry in the subdirectory was in error. CHKDSK will attempt to correct the error if the /F parameter was specified.

EOF mark not found

COMP. An unsuccessful attempt was made to locate the end of valid data in the last block of the files being compared. This message usually occurs when comparing nontext files; it should not occur when comparing text files. For more details, see the COMP command in Chapter 5.

Error found, F parameter not specified
Corrections will not be written to disk

CHKDSK. You have not used the /F parameter. CHKDSK will perform its analysis as though it were going to correct any errors detected, so that you can see the results of its analysis, but it will not actually write the corrections on the disk.

Error in EXE file

DOS. An error was detected in the relocation information placed in the file by the LINK program. This may be due to a modification to the file.

Error in EXE/HEX file

DEBUG. The file contained invalid records or characters.

Error loading operating system

Startup. A disk error occurred while attempting to load your operating system from fixed disk. If the situation persists after several attempts to re-start the system, you should start DOS from your DOS diskette and use the SYS command to transfer a new copy of DOS to your fixed disk.

Error reading fixed disk.

FDISK. The FDISK program was unable to read the startup record of the current fixed disk after five tries. Try the FDISK program again. If after several tries you are unable to proceed, consult the *Guide to Operations* book "Problem Determination" section and see your IBM Personal Computer Dealer.

Error writing fixed disk.

FDISK. The FDISK program was unable to write the startup record of the current fixed disk after five tries. Try the FDISK program again. If after several tries you are unable to proceed, consult the *Guide to Operations* book "Problem Determination" section and see your IBM Personal Computer Dealer.

Error writing to device

Commands. DOS was unable to write the requested number of bytes to the device. This indicates that you tried to send more data to the device than the device was expecting.

Errors on list device indicate that it may be off-line. Please check.

PRINT. The device being used for background printing is offline. This message only appears when the device is offline and you enter a new PRINT command.

EXE and HEX files cannot be written

DEBUG. The data would require a backwards conversion that DEBUG doesn't support.

EXEC failure

Commands. An error was encountered while reading a command from disk, or the FILES= command in the configuration file (CONFIG.SYS) does not specify a large enough value. You should increase that value and restart DOS.

F

File allocation table bad, drive *x*
Abort, Retry, Ignore?

DOS. See the message **Disk error reading drive *x*** under "Device Error Messages" at the beginning of this chapter. If this error persists, the disk is unusable and should be formatted again.

File AND File

COMP. This message indicates the full path and filenames of the two files currently being compared.

File canceled by operator

PRINT. This message appears on the printer after you cancel the printing of a file to serve as a reminder that the printout is incomplete.

File cannot be copied onto itself

DOS. A request is made to COPY a file and place the copy (with the same name) in the same directory on the same disk as the original. You should change the name given to the copy, put it in a different directory or put it on another disk.

File creation error

DOS and commands. An unsuccessful attempt was made to add a new filename to the directory or to replace a file that was already there. If the file was already there, it was marked read-only and therefore could not be replaced. Otherwise run CHKDSK to determine if the directory is full, or if some other condition caused the error.

File is cross-linked:
on cluster *xx*

CHKDSK. This message will appear twice for each cross-linked cluster number, naming the two files in error. The same data block is allocated to both files. No corrective action is taken automatically, so you must correct the problem. For example, you can:

- Make copies of both files (use COPY command).
- Delete the original files (use ERASE command).
- Review the files for validity and edit as necessary.

File is currently being printed
File is in queue

PRINT. These messages appear together when you issue a PRINT command with no parameters, or individually when you queue the first or a subsequent file for printing. This message is provided for your information.

File not found

DOS and commands. A file named in a command or command parameter does not exist in the directory in the specified (or default) drive.

Files are different sizes

COMP. The sizes of the files to be compared do not match. The comparison cannot be done because one of the files contains data which the other does not.

Files compare OK

COMP. The two files just compared contain identical information.

First cluster number is invalid, entry truncated

CHKDSK. The file whose name precedes this message contains an invalid pointer to the data area. The file is truncated to a zero-length file if the /F parameter was specified.

Fixed disk already has a DOS partition.

FDISK. You chose the "Create DOS Partition" option and the current fixed disk which already has a DOS partition.

Fixup offset exceeds field width

LINK. An assembler instruction refers to an address with a NEAR attribute instead of a FAR attribute. Edit assembler source program and process again.

Fixups needed – base segment (hex):

EXE2BIN. The source (.EXE) file contained information indicating that a load segment is required for the file. Specify the absolute segment address at which the finished module is to be loaded.

WARNING
We do not recommend using such a program as a .COM file because the program is dependent upon being loaded at a specific memory location.

FOR cannot be nested

Batch. More than one FOR subcommand was found on one command line in the batch file. Only one FOR subcommand is allowed per command line.

Format failure

FORMAT. A disk error was encountered while creating the target diskette. The diskette is unusable.

Formatting while copying

DISKCOPY. The target diskette was found to contain unformatted tracks. DISKCOPY will format the remainder of the target diskette as it copies data. If this message is followed by the message **Incompatible drive types**, you have tried to copy a dual sided diskette to a drive that does not have dual sided capability; processing will end, and the target diskette will not contain any useful data.

I

Illegal Device Name

MODE. The specified printer must be LPT1:, LPT2:, or LPT3:; the specified Asynchronous Communications Adapter must exist and be COM1: or COM2:. There must be no more than one blank between MODE and its parameters.

Incompatible diskette or drive types

DISKCOMP. The first diskette was successfully read on both sides, but the second diskette could only be read on the first side. Either the second drive or diskette is single sided or the first diskette contains 9-sectors per track and the second diskette contains only 8-sectors per track.

Incompatible drive types

DISKCOMP. The source diskette and drive are dual sided, but the target drive has only single sided capability. The target diskette contains no useful data.

Incompatible system size

SYS. The target diskette contained a copy of DOS that is smaller than the one being copied. The system transfer does not take place. A possible solution might be to format a blank diskette (use the FORMAT /S command) and then copy any files to the new diskette.

Incorrect DOS version

Commands. The command you just entered requires a different version of DOS than the one you are running.

Insert backup diskette *xx* in drive *x*:
Strike any key when ready

RESTORE. Insert the new backup diskette in sequence. RESTORE will continue when you press a key.

Insert backup diskette *xx* in drive *x*:
Warning! Diskette files will be erased
Strike any key when ready

BACKUP. Insert the next diskette to be used for the backup. Use DOS formatted diskettes only. BACKUP will continue when you press a key.

Insert COMMAND.COM disk in drive *x*:
and strike any key when ready

DOS. DOS is attempting to reload the command processor, but COMMAND.COM is not on the drive that DOS was started from. Insert the DOS diskette in the indicated drive and press any key.

Insert disk to be recovered into
drive *x*: and press any key when ready

RECOVER. Insert the diskette to be recovered in the indicated drive and press any character key.

Insert disk with batch file
and strike any key when ready

DOS. The diskette that contained the batch file being processed was removed. The batch processor is trying to find the next command in the file. Processing will continue when you insert the diskette in the appropriate drive and press a key.

Insert DOS disk in *x*:
and strike any key when ready

SYS and FORMAT. FORMAT or SYS is trying to load the DOS files, but the indicated drive *x*: does not contain the DOS diskette.

Insert DOS diskette in drive A:
Press any key when ready . . .

FDISK. You have successfully created the DOS partition on the current fixed disk. Insert the DOS diskette into drive A: and press any key. This will restart your IBM Personal Computer. The current fixed disk will now be assigned a fixed disk letter and you can now FORMAT the fixed disk.

Insert first diskette in drive *x*
Insert second diskette in drive *x*

DISKCOMP. Insert the first (or second) diskette of the two diskettes to be compared into the indicated drive. One or both of these messages will be followed by the message **Strike any key when ready**. When you press a key, the comparison will continue.

Insert source diskette in drive *x*
Insert target diskette in drive *x*

DISKCOPY. Insert the appropriate diskette into the indicated drive, and press any key when prompted. The copying process will continue.

Insufficient disk space

DOS and commands. The disk does not contain enough free space to contain the file being written. If you suspect this condition is invalid, run CHKDSK to determine the status of the disk.

Insufficient memory

Commands. The amount of available memory is too small to allow these commands to function. You should change the BUFFERS= parameter in the CONFIG.SYS file to a smaller value (if you have specified BUFFERS=), restart the system and try the command again. If the message still appears, then your system does not have enough memory to execute the command.

Insufficient room in root directory
Erase files from root and repeat CHKDSK.

CHKDSK. You have instructed CHKDSK to create files from the "lost" data blocks it has found, but the root directory is full, so all of the lost chains could not be recovered into files. You should copy some of the recovered files to another disk for further examination, delete them from the disk you are checking, then run CHKDSK again to recover the remainder of the lost data.

Insufficient space on disk

DEBUG. A Write command was issued to a disk that doesn't have enough free space to hold the data being written. If you are writing to diskette, you may insert a diskette that does have enough free space and re-issue the Write command. Otherwise, you should erase files from the disk and run DEBUG again.

Intermediate file error during pipe

DOS. DOS was unable to create one or both of its intermediate files because the default drive's root directory was full, or DOS is unable to locate the piping files, or the disk contains insufficient available space to hold the data being piped. You should erase some files from the default drive's root directory and re-issue the command that failed. If it still fails, one of the programs in the command line has erased one or both of the piping files. You should correct the program and re-issue the command line.

Invalid baud rate specified

MODE. The baud rate you specify must be 110, 150, 300, 600, 1200, 2400, 4800, or 9600 (or the first two characters of the number).

Invalid characters in volume label

FORMAT. One or more of the characters you entered in the volume label is not a valid filename character, or the name contained a period (volume labels contain 1 to 11 valid characters without a period).

Invalid COMMAND.COM in drive *n*

DOS. While trying to reload the command processor, the copy of COMMAND.COM on the disk was found to be an incorrect version. You are prompted to insert the correct DOS diskette and press any key to continue.

Invalid date

DATE. An invalid date or delimiter was entered. The only valid delimiters in a date entry are hyphens (-) and slashes (/).

Invalid device

CTTY. The device name you specified is an invalid name to DOS.

Invalid directory

DOS and commands. One of the directories in the specified path does not exist.

Invalid drive in search path

DOS. An invalid drive specifier was found in one of the paths specified in the PATH command. This message appears when DOS attempts to locate a command or batch file, not at the time you issued the erroneous PATH command.

Invalid drive specification

DOS and commands. An invalid drive specification was just entered in a command or one of its parameters.

Invalid format file

LINK. A library is in error.

Invalid number of parameters

Commands. You have specified too few or too many parameters for the command you issued.

Invalid numeric parameter

LINK. Numeric value not in digits.

Invalid object module

LINK. Object module(s) incorrectly formed or incomplete (as when the language processor was stopped in mid-process).

Invalid parameter

DOS and commands. One or more of the parameters entered for these commands was not valid. If the program expects a drive specifier, be sure to enter a colon following the drive letter. In other cases, be sure the character following the slash (/) is valid for the program being run.

Invalid parameters

MODE. No parameters were entered, or the first parameter character was other than L, or C, or the first parameter was other than **40, 80, BW40, BW80, CO40, CO80, MONO, L, R**, or the display adapter the parameter refers to is not present in the machine.

Invalid partition table

Startup. While attempting to start DOS from your fixed disk, the start-up procedures detected invalid information in the disk's partition information. You should start DOS from diskette and use the FDISK command to examine and correct the fixed disk partition information.

Invalid path

TREE. Tree was unable to use a directory whose name was found in another directory. You should run CHKDSK to determine what is wrong with the directory structure.

Invalid path, not directory or directory not empty

RMDIR. The specified directory was not removed because one of the names you specified in the path was not a valid directory name, or the directory you specified still contains entries for files or other sub-directories. You cannot remove a directory unless it is empty (with the exception of the . and .. entries). Also you cannot remove a current directory.

Invalid path or file name

COPY. You specified a directory or file name that does not exist.

Invalid sub-directory

CHKDSK. Invalid information was detected in the sub-directory whose name precedes this message. CHKDSK will attempt to correct the error. For more specific information about the nature of the error, run CHKDSK with the /V parameter.

Invalid switch: *z*

LINK. The characters indicated by *z* do not form a valid linker parameter.

Invalid time

TIME. An invalid time or delimiter was entered. The only valid delimiters are the colon between the hours and minutes, and the minutes and seconds; and a period between the seconds and hundredths of a second.

L

Label not found

Batch. A GOTO command named a label which does not exist in the batch file.

Line too long

EDLIN. Upon replacing a string, the replacement causes the line to expand beyond the 253-character limit. The Replace Text command is ended abnormally.

Split the long line into shorter lines; then issue the Replace Text command again.

List output is not assigned to a device

PRINT. The device you named to be the PRINT list device is not recognized as a valid device. You should re-issue the PRINT command and reply with a valid list device name when prompted.

LPT#: not redirected.

MODE. The parallel printer will now receive its own output, even if this printer's output had previously been redirected to a serial device. This indicates cancellation of any previous redirection which may have been in effect, because you have set the printer width or vertical spacing.

LPT#: redirected to COM*n*:

MODE. Any request that would normally have gone to the parallel printer LPT# (#=1, 2, or 3) is sent instead to the serial device COM*n* (*n*=1 or 2).

LPT#: set for 80

MODE. An attempt was made to set the printer line length to 80 characters by requesting standard type format. If the attempt was unsuccessful, an error message will follow this message on the screen.

LPT#: set for 132

MODE. An attempt was made to set the printer line length to 132 characters by requesting compressed type format. If the attempt was unsuccessful, an error message will follow this message on the screen.

M

**Maximum available space is *xxxx*
cylinders at cylinder *xxxx*.**

FDISK. The "Create DOS Partition" option displays the largest available piece of space on the current fixed disk. These numbers are also used as the defaults for the two prompts that will follow.

**Memory allocation error
Cannot load COMMAND, system halted**

DOS. A program has destroyed the area in which DOS keeps track of available memory. You should restart DOS.

Missing operating system

Startup. While attempting to start DOS from fixed disk, the startup procedures determined that the DOS partition was marked as being "bootable" (startable), but that it doesn't contain a copy of DOS. You should start DOS from diskette and use FORMAT with the /S parameter to place a copy of DOS on the fixed disk. You might want to backup your files before doing the FORMAT.

Must specify destination line number

EDLIN. A Move or Copy command was entered without a destination line number. Reenter the command with a valid destination line number.

N

No DOS partition to delete.

FDISK. You chose the "Delete DOS Partition" option when there was no DOS partition on the current fixed disk to be deleted.

Name of list device [PRN]:

PRINT. This message appears the first time you start print after DOS has been restarted. Reply with the reserved device name which is to receive the printed output, or simply press Enter if the first parallel printer [PRN] is to be used.

No free file handles
Cannot start COMMAND, exiting

DOS. An attempt to load a second copy of the command processor has failed because there are currently too many files open. You should increase the number in the FILES= command in the configuration file (CONFIG.SYS), and restart DOS.

No fixed disks present.

FDISK. The FDISK program was run on an IBM Personal Computer that:

- Does not have a fixed disk.
- Has a fixed disk in the expansion unit and the expansion unit is not powered on.
- Has a fixed disk that is not properly installed.

No object modules specified

LINK. You did not name any object modules in the command line or in response to the prompt. The linker needs some files to link.

No partitions to make active.

FDISK. You chose the "Change Active Partition" option when there were no partitions on the current fixed disk to be made active. You can use the "Create DOS Partition" option to create a partition and then the "Change Active Partition" option to make it the active partition.

No path

PATH. There is currently no alternate path for DOS to search to find commands and batch files if it does not find them in the specified (or default directory).

No room for system on destination disk

SYS. The destination diskette did not already contain the required reserved space for DOS; therefore, the system cannot be transferred. A possible solution would be to format a blank diskette (use the FORMAT /S command), and then copy any other files to the new diskette.

No room in directory for file

EDLIN. The directory on the specified disk is full. Your editing changes are lost. You should assure that your disk has auditable directory entries and run EDLIN again.

No space for a xxxx cylinder partition.

FDISK. You entered a "Partition Cylinder Size" that is larger than the largest piece of free space on the disk. Enter a smaller number.

No space for a xxxx cylinder partition at cylinder xxxx.

FDISK. You requested a partition to be created at a place on the current fixed disk and it does not have space at that place to create a DOS partition.

No space to create a DOS partition

FDISK. You chose the "Create DOS Partition" option on the current fixed disk which has no space to create a DOS partition.

No sub-directories exist

TREE. The specified drive contains only a root directory. Therefore, there is no directory path to display.

Non-System disk or disk error
Replace and strike any key when ready

Startup. There is no entry for IBMBIO.COM or IBMDOS.COM in the directory; or a disk read error occurred while starting up the system. Insert a DOS diskette in drive A: and restart your system.

Not enough room to merge the entire file

EDLIN. A Transfer command was unable to merge the entire contents of the specified file because of insufficient memory. Part of the file was merged.

Not found

EDLIN. Either the specified range of lines does not contain the string being searched for by the Replace Text or Search Text commands; or if a search is resumed by replying N to the OK? prompt, no further occurrences of the string were found.

O

Out of environment space

DOS. DOS was unable to accept the SET command you just issued because it was unable to expand the area in which the environment information is kept. This normally occurs where you try to add to the environment after loading a program which makes itself resident (PRINT, MODE, or GRAPHICS for example).

Out of space on list file

LINK. This error usually occurs when there is not enough disk space for the List file.

Out of space on run file

LINK. This error usually occurs when there is not enough disk space for the Run file (.EXE).

Out of space on VM.TMP

LINK. No more disk space remained to expand the VM.TMP file.

P

Parameters not compatible

FORMAT. You attempted to use two parameters that are not compatible with each other (/B and /V for example).

Parameter not compatible with fixed disk

FORMAT. You specified the /1 or /8 parameter while formatting a fixed disk. Neither of these parameters is valid for a fixed disk.

Press any key to begin formatting *x*:

FORMAT. The fixed disk (drive *x*) is about to be formatted. Formatting will lose track of all previously existing data on the disk. If you do *not* want the disk formatted, press Ctrl-Break. If you do want the disk formatted, press a character key.

Print queue is empty

PRINT. There are currently no files being processed by PRINT.

Print queue is full

PRINT. You attempted to add more than the limit of 10 files to the print queue. You will have to wait until a file is printed before you can add another file to the print queue.

Printer error

MODE. The MODE command (option 1) was unable to set the printer mode because of an I/O error, out of paper (or POWER OFF), or time out (not ready) condition.

Printer lines per inch set

MODE. An attempt has been made to set the printer vertical spacing to the specified 6 or 8 lines per inch. If the attempt was unsuccessful, an error message will follow this message on the screen.

Probable non-DOS disk.
Continue (Y/N)?

CHKDSK. The file allocation table identification byte contains invalid information. Either the disk was not formatted by DOS or has become badly damaged. If you did not use the /F parameter, and you reply Y, CHKDSK will indicate its possible corrective actions without actually changing the disk. We recommend doing this first, before you consider using the /F switch and replying Y.

Processing cannot continue,

CHKDSK. This message will be followed by another message which explains why CHKDSK cannot continue. Normally, this condition is caused by insufficient memory.

Program size exceeds capacity of LINK

LINK. Load module is too large for processing.

Program too big to fit in memory

DOS. The file containing the external command cannot be loaded because it is larger than the available free memory. You should reduce the number in the BUFFERS= parameter in your CONFIG.SYS file (if you have specified BUFFERS=), restart your system and re-issue the command. If the message reappears, your system does not have enough memory to execute the command.

R

Requested stack size exceeds 64K

LINK. Specify a size $\leq$ 64K bytes when the STACK SIZE: prompt appears.

Resident part of PRINT installed

PRINT. The message appears the first time you use the PRINT command. A program has been loaded into memory to handle subsequent PRINT commands. Available memory for your applications has been reduced by approximately 3200 bytes.

Resident portion of MODE loaded

MODE. When MODE is invoked for a non-screen-setting function it is sometimes necessary to load a portion of code to be made permanently resident.

S

Sector size too large in file <filename>

Startup. The device driver named in <filename> specifies a device sector size larger than the devices previously defined to DOS.

Segment size exceeds 64K

LINK. Attempted to combine identically named segments which resulted in a segment requirement of greater than 64K bytes. The addressing limit is 64K bytes.

Stack size exceeds 65535 bytes

LINK. The size specified for the stack must be less than or equal to 65535.

Symbol defined more than once

LINK. The Linker found two or more modules that define a single symbol name.

Symbol table capacity exceeded

LINK. Very many, very long names were entered. The names exceeded approximately 50K bytes. Use shorter and/or fewer names.

Syntax error

DOS. The command you entered is improperly formatted. Check to make sure you have used the correct format for this command.

T

Target diskette may be unusable

DISKCOPY. This message follows an unrecoverable read, write, or verify error message. The copy on the target diskette may be incomplete because of the unrecoverable I/O error.

Target diskette write protected
Correct, then strike any key

DISKCOPY. You are trying to produce a copy on a diskette that is write-protected.

Terminate batch job (Y/N)?

DOS. This message appears when you press Ctrl-Break while DOS is processing a batch file. Press Y to stop processing the batch file. Pressing N only ends the command that was executing when Ctrl-Break was pressed; processing resumes with the next command in the batch file.

The current active partition is *x*.

FDISK. The "Change Active Partition" option displays the active partition on the current fixed disk.

The last file was not restored

RESTORE. You stopped RESTORE before it completely restored the last file listed or there was not enough room on the fixed disk. RESTORE then deleted the partially restored file.

There was/were
number **errors detected**

LINK. This message is displayed for your information at the end of the link session.

Too many external symbols in one module

LINK. The limit is 256 external symbols per module.

Too many groups

LINK. The limit is 10, including DGROUP.

Too many libraries specified

LINK. The limit is eight libraries.

Too many overlays

LINK. The limit is 64.

Too many public symbols

LINK. The limit is 1024 public symbols.

Too many segments or classes

LINK. The limit is 247 (segments and classes taken together).

Total disk space is *xxxx* cylinders.

FDISK. The total space on the current fixed disk is displayed.

Track 0 bad-disk unusable

FORMAT. Track 0 is where the boot record, file allocation table, and directory must reside. The disk is unusable.

Tree past this point not processed

CHKDSK. CHKDSK is unable to continue processing the directory path currently being examined because track 0 is bad.

U

Unable to create directory

MKDIR. The directory you wish to create already exists, or one of the directory path names you specified could not be found, or you attempted to add a directory to the root directory and it is full, or a file already exists by that name in that directory.

Unable to write BOOT

FORMAT. The first track of the diskette or DOS partition is bad. The BOOT record could not be written on it. The diskette or DOS partition is not usable.

Unexpected end-of-file on library

LINK. This is probably caused by an error in the library file.

Unexpected end of file on VM.TMP

LINK. The diskette containing VM.TMP has been removed.

Unrecognized command in CONFIG.SYS

Startup. An invalid command was detected in the configuration file CONFIG.SYS. You should edit the file, correct the invalid command and restart DOS.

Unrecoverable format error on target
Target diskette unusable

DISKCOPY. An unrecoverable error was encountered while formatting the target diskette. The diskette contains no usable data.

Unrecoverable read error on drive *x*
Track *xx*, side *x*

DISKCOMP. Four attempts were made to read the data from the diskette in the specified drive. The data could not be read from the indicated track and side.

Unrecoverable read error on source
Track *xx*, side *x*

DISKCOPY. Four attempts were made to read the data from the source diskette. DISKCOPY continues copying, but the copy may contain incomplete data.

Unrecoverable verify error on target
Track *xx*, side *x*

DISKCOPY. Four attempts were made to verify the write operation to the target diskette. DISKCOPY continues copying, but the copy may contain incomplete data.

Unrecoverable write error on target
Track *xx*, side *x*

DISKCOPY. Four attempts were made to write the data to the target diskette. DISKCOPY continues copying, but the copy may contain incomplete data.

Unresolved externals: list

LINK. The external symbols listed were not defined in the modules or library files that you specified. If this error occurs, do not attempt to run the executable file created by the linker.

V

VM.TMP is an illegal filename and has been ignored

LINK. VM.TMP cannot be used for an object filename. This message is only a warning.

Volume label (11 characters, ENTER for none) ?

FORMAT. You are requested to enter a 1 to 11 character volume label which will be written on the disk being formatted. If you do not want a volume label on the disk, press only the ENTER key.

W

Warning! All data in the DOS partition will be DESTROYED. Do you wish to continue.....? [d]

FDISK. The "Delete DOS Partition" option is warning you that if you continue, all data in the DOS partition on the current fixed disk will be destroyed. If you press Enter, the DOS partition will NOT be destroyed. If you do wish to delete the DOS partition, type Y and press Enter.

Warning-directory full

RECOVER. There is insufficient directory space to recover more files. You should copy some of the files to another disk, erase them from this disk and run RECOVER again.

Warning! Diskette is out of sequence
Replace the diskette or continue
Strike any key when ready

RESTORE. The backup diskette is not the next one in sequence. Replace the diskette unless you are sure no files on the diskette(s) you skipped would be restored. RESTORE will continue when you press a key. This message will be repeated if you try to skip a diskette which contains part of a file being restored.

Warning! File *xx*
is a read-only file
Replace the file (Y/N)?

RESTORE. The indicated file is read-only. Enter Y if you want to replace it or N if you do not. RESTORE will continue after you press ENTER. You will see this message only if you specified the /P option.

Warning! File *xx*
was changed after it was backed up
Replace the file (Y/N)?

RESTORE. The indicated file on the fixed disk has a later date and time than the corresponding file on the backup diskette. Enter Y if you want to replace it with the backed up version or N if you do not. RESTORE will continue after you press ENTER. You will see this message only if you specified the /P option.

Warning! No files were found to back up

BACKUP. No fixed disk files were found that matched the backup file specification.

Warning! No files were found to restore

RESTORE. No backup diskette files were found that matched the restore file specification.

Warning: no stack segment

LINK. None of the object modules specified contain a statement allocating stack space.

WARNING—Read error on EXE file

EXE2BIN. An error occurred while reading the input file. EXE2BIN will attempt to continue, but the result file may be unusable.

X

***x* is not a choice. Enter a choice.**

FDISK. You entered *x* which is not a choice for this question.

***x* is not choice. Enter Y or N.**

FDISK. You entered *x* which is not a choice for this question. Enter Y or N.

xxxxxxxxx bytes disk space freed

CHKDSK. Diskette space marked as allocated was not allocated. Therefore, the space was freed and made available.

xxxx error on file _yyyy_

PRINT. This message appears on the printer. While attempting to read data from file _yyyy_ for printing, a disk error of type _xxxx_ was encountered. Printing of that file is stopped.

xxx lost clusters found in _yyy_ chains

CHKDSK. CHKDSK located _xxx_ blocks of the data area which were marked as allocated, but were not associated with a file. These clusters are assumed to contain "lost" data, and CHKDSK will ask whether you wish to free them, or to recover each chain into a separate file.

***** Backing up files to diskette _xx_ *****

BACKUP. This message will be followed by a list of files that were backed up on the indicated diskette.

***** Files were backed up _xx/xx/xxxx_ *****

RESTORE. The files on the backup diskette were backed up on the indicated date.

***** Restoring files from diskette *xx* *****

RESTORE. This message will be followed by a list of files that were restored from the indicated diskette.

—More—

MORE. The screen is full and there is more data waiting to be displayed. Press any character to see the next screen full.

10 Mismatches—ending compare

COMP. Ten mismatched locations were detected in the files being compared. COMP assumes that the files are so different that further comparisons would serve no purpose.

Section 2. Advanced DOS Features and Technical Reference

Contents

Notes:

Chapter 9. Configuring Your System

Contents

Notes:

Introduction

Each time DOS is started, it searches the root directory of the drive (from which it was started) for a special configuration file named CONFIG.SYS. If found, it reads the file and interprets the text commands within it.

Configuration Commands

The following commands can be included in the configuration file. If you add or change any of the configuration file commands, they will become effective the *next* time DOS is started.

BREAK Command

BREAK=ON/OFF

This command should only be used once in the configuration file. The default value is OFF, and causes DOS to check for Ctrl-Break being entered at the keyboard only when DOS is performing screen, keyboard, printer or Asynchronous Communication Adapter operations. With this setting, it may not be possible to cancel an executing program by using Ctrl-Break unless the program causes DOS to perform one of those four operations. Specifying ON causes DOS to check for Ctrl-Break whenever it performs *any* function for a program. This allows you to "break" out of programs that perform few (or no) screen, keyboard, printer, or auxiliary device operations (such as compilers). The ON/OFF state set in the configuration file can later be changed by issuing a BREAK command (see Chapter 6).

BUFFERS Command

BUFFERS=*xx*

Where *xx* is a number between 1 and 99. This is the number of disk buffers that DOS should allocate in memory when it starts up. The default value is 2, and this value will remain in effect until DOS is restarted with a different value specified in the configuration file.

What Is a Buffer

A disk buffer is a block of memory that DOS uses to hold data being read from, or written to a disk (fixed disk or diskette), when the amount of data being transferred is not an exact multiple of the sector size. For example, if an application reads a 128-byte record from a file, DOS will read the entire sector into one of its buffers, locate the correct 128-byte record in the buffer, and move the record from the buffer into the application's area of memory. It then marks that buffer as having been used recently. On the next request to transfer data, DOS will attempt to use a different buffer. In this way, all of the buffers will eventually contain the most recently-used data. The more buffers DOS has, the more data will be in memory.

Read/Write Requests

Each time DOS is requested to read or write a record that is not an exact multiple of the sector size, it first looks to see if the sector containing that record is already in a buffer. If not, it must read the sector as described above. But if the data is already in a buffer, then DOS can simply transfer the record to the application's area without the need to read the sector from the disk—this saves time. This savings is realized on both reading and writing records, since DOS must first read a sector before it can insert a record your application is attempting to write.

Random/Sequential Applications

For applications that read and write records in a random fashion (such as many Basic and data base applications), the likelihood of finding the correct record already in a buffer increases if DOS has more buffers to work with. This can greatly speed up the performance of those applications.

For applications doing sequential reads and writes, however (read an entire file, write an entire file), there is little advantage to having a large number of buffers allocated.

Because all applications are different, there is no specific number of buffers that will serve all applications equally well. If your applications do little random reading and writing of records, the system default of 2 buffers (if you do not specify BUFFERS= in your configuration file) should be sufficient.

However, if you use data-base type applications, or run programs that perform a lot of random reads and writes of records, you will want to increase the number of DOS buffers. The "best" number of buffers for your particular application can only be determined by using different values until the best performance is achieved. For most data base applications, a value between 10 and 20 buffers will usually provide the best results.

Beyond that point, the system may appear to start running slower - this is because, with a very large number of buffers it can take DOS longer to search all the buffers for the record than it would take to read the record from disk.

Size of Your Computer

The final consideration in determining the number of buffers to allocate is the memory size of your computer. Since each additional buffer increases the resident size of DOS by 528 bytes, the amount of memory available to the application is reduced by that amount. Therefore, additional buffers may actually cause some applications to slow down, since there is less memory in which the application itself can keep data—this could result in more frequent reads and writes than would otherwise be necessary.

In summary, the optimum number of buffers must be determined by *you*, based on:

1. The types of applications most often used
2. The memory size of your computer
3. Your analysis of system performance when using your applications with different numbers of buffers allocated.
4. For computers with fixed disks, we recommend a minimum of BUFFERS=3.

DEVICE Command

DEVICE=[*d:*][*path*]*filename*[*.ext*]

This command allows you to specify the name of a file containing a device driver. During startup, DOS loads the file into memory as an extension of itself, and gives it control as described in "Installable Device Drivers" in Chapter 14. Please refer to that section for technical information about installable device drivers.

Loading Standard Device Drivers

The standard device drivers loaded by DOS support the standard screen, keyboard, printer, auxiliary device, diskette, and fixed disk devices. A clock driver is also loaded (see Chapter 14). You don't need to specify any DEVICE= commands for DOS to support these devices.

Replacing Standard Device Drivers

If you wish to use the "Extended Screen and Keyboard Control" features described in Chapter 13, you should create the file CONFIG.SYS on the disk you will be starting DOS from. The file should contain the command DEVICE=ANSI.SYS. This command causes DOS to replace the standard screen and keyboard support with the extended functions.

Installing Your Own Device Driver

For systems programmers and application developers—if you have written device drivers that you want DOS to load when it starts, include a DEVICE= command in the CONFIG.SYS file for each driver to be loaded.

FILES Command

FILES=*xx*

The maximum value for *xx* is 99.

Beginning with DOS Version 2.00, there is no need for an application to construct a special control block (FCB) in order to access a file. Instead, the program can simply specify an ASCII string consisting of drive specifier, complete directory path name and filename when opening or creating a file. DOS will locate the correct drive, directory and file, and will create and return a *handle*—merely a 16-bit binary value.

Accessing a File

All file accesses (reads, writes, close) can then be performed by telling DOS which handle to use. When an application opens a file in this manner, DOS constructs a control block in its own memory on behalf of the application, in an area that was set aside when DOS started. The size of this area (and consequently, the maximum number of files that can be concurrently open), depends on the value specified in the FILES= command.

The default value is FILES=8; that is, no more than 8 files can be open at the same time. There is no effect on the number of files that can be concurrently open using the traditional (OPEN FCB) functions. This default value is sufficient for the majority of operating environments. However, if applications are installed that result in error messages indicating an insufficient number of handles, the FILES= command should be used to provide DOS with additional handles.

Number of Files Opened

The value specified in FILES= becomes the new maximum number of files that DOS allows to be concurrently open.

Note that this value is the maximum number of files allowed for the entire system. The maximum number of files that a process can have concurrently open is 20 (this number includes the 5 predefined handles for standard input, output, error, auxiliary, and standard printer).

If you specify FILES= in your configuration file, the size of the resident portion of DOS increases by *39* bytes for each additional file above the default value of 8. Consequently, the memory available to the application is reduced by the same amount. See function calls hex 3C through hex 46 in Appendix D for descriptions of the new file-handling functions.

SHELL Command

SHELL=[*d:*][*path*]*filename*[*.ext*]

This command allows you to specify the name and location of a top-level command processor that DOS initialization will load in place of COMMAND.COM.

System programmers who develop their own top-level command processor should remember to include provisions for handling interrupts hex 22, hex 23 and hex 24, and for reading and executing commands.

> **Note:** Because the internal commands, batch processor, and EXEC function call (program loader) reside in COMMAND.COM. These functions will not be available to the user unless they are duplicated in your command processor.

Notes:

Chapter 10. Advanced DOS Commands

Contents

Notes:

Introduction

This chapter explains how to use the advanced DOS commands. You can use advanced DOS commands to:

- Set options for the Asynchronous Communications Adapter.
- Define a remote device as your primary console.
- Sort text data.
- Search files for occurrences of specified strings of text.
- Display a screen full of data at a time.
- Set new system prompt.
- Set the system environment.
- Convert .EXE files to .COM files.

Redirection of Standard Input and Output Devices

The DOS standard input and output device redirection feature allows a program to receive its input from a source other than the keyboard (standard input), or direct its output to a device other than the display screen (standard output).

The standard input and output devices can be redirected to or from files or other devices by the following DOS command line parameters:

>[*d*:][*path*]*filename*

Causes *filename* to be created (or truncated to zero length) and then assigns standard output to that file. All output that would normally have gone to the screen from the command is placed in the file.

>>[*d*:][*path*]*filename*

Causes *filename* to be opened (created if necessary) and positions the write pointer at the end of the file so that all output is appended to the file.

<[*d*:][*path*]*filename*

Causes standard input to be assigned to *filename*. All input to the program comes from this file instead of from the keyboard.

CAUTION
When using this method of providing input to a program, be sure *all* of the program's input is in the file. If the program attempts to obtain more input after end-of-file is reached, DOS is unable to supply the input, and processing will stop. You can return to the DOS prompt by entering Ctrl-Break.

Note: If an application does not use DOS function calls to perform standard input and/or output (for example you put text directly into the video buffer), then redirection will not work for that application.

Example: In this example, the output of the DIR command is sent to the printer:

```
DIR >PRN
```

In this example, the output of the DIR command is sent to file DIRLIST:

```
DIR >DIRLIST
```

In the following example, program MYPROG will receive its input from file INPUT.TXT, instead of from the keyboard:

```
MYPROG <INPUT.TXT
```

Piping of Standard Input and Output

The DOS piping feature allows the screen output of one program to be used as the keyboard input to another program. DOS uses temporary files to hold the input and output data being piped. These temporary files are created in the root directory of the default drive and have the form:

```
%PIPEx.$$$
```

The programs being piped must use care not to cause the piping files to be erased or modified.

Piping is the chaining of programs with automatic redirection of standard input and output (refer to "Redirection of Standard Input and Output Devices" in this chapter for additional information). The names of the programs to be chained are separated by the vertical bar (¦) character on the command line.

The following are typical examples of using the piping feature for a program that does all of its input and output to the standard devices (screen and keyboard). For example, if the program named SORT read all of it's standard input, sorted it, and then wrote it to the standard output device, the command:

```
DIR¦SORT
```

would generate a sorted directory listing. This causes all standard output generated by the DIR command to be sent to the standard input of the SORT program.

To send the sorted directory to a file, you would type:

```
DIR ¦SORT>FILE
```

If you wish the file to contain only the directory entries for sub-directories, you could enter:

```
DIR ¦FIND  "DIR" ¦ SORT>FILE
```


DOS Filters

A filter is a program or command that reads data from a standard input device, modifies the data, then writes the result to a standard output device. Thus, the data has been "filtered" by the program. For example, one of the filters on your DOS diskette is called SORT. SORT reads input from the standard input device (normally the keyboard), sorts the lines of data, then writes the sorted results to the standard output device (normally the screen). With the redirection capabilities described earlier in this chapter, you can cause SORT to receive its input from some other source, and to send its output to a different destination. For example,

```
SORT <MYFILE >RESULT
```

will cause SORT to read the file MYFILE, sort the lines within it, and write the sorted output to file RESULT.

By using the piping feature, you can cause a filter to receive its input from the output of another command, or to send its output to the input of another command. For example,

```
DIR ¦ SORT
```

causes the output listing from the DIR command to be used by SORT as its input. The listing will be sorted and the result displayed on the screen.

There are three filters on your DOS diskette, and they are described as individual commands in this chapter. They are:

SORT Sorts text data.

FIND Searches files for occurrences of specified strings of text.

MORE Displays a screen full of data at a time, then pauses with the message —More—.

You can easily add your own filter to the filters that have been supplied; just write a program that reads its input from the standard input device, and writes its output to the standard output device.

> Note: If an application does not use DOS function calls to perform standard input and/or output (for example you put text directly into the video buffer), then filters will not work for that application.

Detailed Descriptions of Advanced DOS Commands

This section presents a detailed description of how to use the advanced DOS commands. The commands appear in alphabetical order; each with its purpose, format, and type. Examples are provided where appropriate. For "Information Common to All DOS Commands" refer to Chapter 6.

Invoking a Secondary Command Processor

If you wish to invoke a secondary command processor, the following syntax should be used:

COMMAND [*d*:][*path*] [/P] [/C *string*]

Where *d:path* will be the directory searched for the command processor to be loaded, /P causes the new copy to become permanent in memory, and /C *string* allows you to pass a command line (string) as a parameter. The command line will be interpreted and acted upon as if you had entered it as a normal command. For example, COMMAND /C DIR B: causes a secondary command processor to be loaded, and it executes the command DIR B:.

Issuing COMMAND without any parameters causes a new copy of the command processor to be loaded, and this new copy will inherit the environment (will never be able to change the environment via a SET command).

When a secondary command processor has been loaded, you can cause it to return to the previous level of command processor by issuing the special command EXIT. If you used the /P parameter, it will not return to the previous level (refer to Appendix F for additional information).

CTTY (Change Console) Command

Purpose: Changes the standard input and output console to an auxiliary console, or restores the keyboard and screen as the standard input and output devices.

Format: CTTY *device-name*

Type: Internal *** External

Remarks: Defines the device to be used as the primary console. Specifying AUX, COM1, or COM2 causes DOS to use that device as the primary console. Specifying CON resets the standard input and output device to the primary console.

Example: In this example, the command causes DOS to use the AUX device for its screen and keyboard operations:

```
CTTY AUX
```

CTTY (Change Console) Command

In this example, the command reverses the previous assignment, causing DOS to switch back to the standard screen and keyboard for its operations:

```
CTTY CON
```

Notes:

1. The CTTY command accepts the name of *any* character-oriented device to allow you to install your own device drivers, and to specify their device names. You must be certain that the named device is capable of both input and output operations. For example, you should not specify the name of a printer, because DOS will attempt to read from that device.

2. The CTTY command only accepts programs that use DOS function calls. Other programs, such as BASIC (that do not use DOS function calls), will not be able to use the CTTY command to change the standard input and output device.

EXE2BIN Command

Purpose: Converts .EXE files that have no segment fixup to a form that is compatible with .COM programs. This results in a saving of diskette space and faster program loading.

Format: EXE2BIN [*d:*][*path*]*filename*[*.ext*]]

[*d:*][*path*][*filename*[*.ext*]]

Type: Internal External
***

Remarks: The file named by *filespec* is the input file. If no extension is specified, it defaults to .EXE. The input file is converted to .COM file format (memory image of the program) and placed in the output file, [*d:*]*filename*[*.ext*]. If you do not specify a drive, the drive of the input file is used. If you do not specify an output filename, the input filename is used. If you do not specify a filename extension in the output filename, the new file is given an extension of .BIN. If you do not specify a path, the current directory is used.

The input must be in valid .EXE format as produced by the linker. The *resident*, or actual code and data, part of the file must be less then 64K. There must be no STACK segment.

EXE2BIN Command

Two kinds of conversions are possible, depending on the specified initial CS:IP:

- If CS:IP is not specified in the program (the .EXE file contains 0:0), a pure binary conversion is assumed. If segment fixups are necessary (the program contains instructions requiring segment relocation), you are prompted for the fixup value. This value is the absolute segment at which the program is to be loaded.

  In this case, the resultant program is usable only when loaded at the absolute memory address specified by a user application. The DOS command processor will not be capable of properly loading the program.

- If CS:IP is specified as 0000:100H, it is assumed that the file is to be run as a COM file, with the location pointer set at 100H by the assembler statement ORG; the first 100H bytes of the file are deleted. No segment fixups are allowed, as COM files must be segment relocatable; that is, they must assume the entry conditions explained in Appendixes B-K. In this case, once the conversion is complete, you may rename the resultant file to a .COM extension. Then, the command processor is capable of loading and executing the program in the same manner as the .COM programs supplied on your DOS diskette.

If CS:IP does not meet one of these criteria, or if it meets the COM file criterion but has segment fixups, the following message is displayed:

```
File cannot be converted
```

This message is also displayed if the file is not a valid .EXE file.

To produce standard COM files with the assembler, you must both use the assembler statement ORG to set the location pointer of the file at 100H and specify the first location as the start address. (This is done in the END statement.) Also, the program must not use references that are defined only in other programs. For example, with the IBM Personal Computer MACRO Assembler:

```
ORG 100H
START:
  •
  •
  •
END START
```

EXE2BIN resides on your DOS Supplemental Program diskette.

FIND Filter Command

Purpose: This filter sends to the standard output device all lines from the filenames specified in the command line that contain the specified string.

Format: FIND [/V][/C][/N]*string*[[*d:*][*path*]*filename*[.*ext*]...]

Type: Internal External
***

Remarks: The /V parameter causes all lines *not* containing the *string* to be displayed.

The /C parameter causes FIND to display only a count of the number of matching occurrences of *string* in each file, without displaying the matching lines from the file.

The /N parameter causes the relative line number of each matching line to be displayed ahead of the line from the file.

The string should be enclosed in double quotes. Two quotes in succession are taken as a single quote.

Global filename characters are not allowed in the filenames or extensions.

Examples: **A>FIND "Fool's Paradise" book1.txt book2.txt book3**

will output all lines from the book1.txt, book2.txt, and book3 (in that order) that contain the string "Fool's Paradise". Or,

A>DIR B: ¦ FIND /V "DAT"

will output the names of all the files in drive B that do not contain the string DAT.

MORE Filter Command

Purpose: This filter reads data from the standard input device, and sends one screen-full of data to the standard output device, and then pauses with the message —More—.

Format: MORE

Type: Internal External
***

Remarks: Pressing any character key causes another screen-full of data to be written to the standard output device. This process continues until all input data is read.

Example: In this example, the command line will display the contents of file TEST.ASM one screen-full at a time. When the screen is full, the message —More— appears on the bottom line. You can press any key to see the next screen-full:

```
MORE <TEST.ASM
```

PROMPT (Set System Prompt) Command

Purpose: Sets a new system prompt.

Format: PROMPT [*prompt-text*]

Type: Internal External
***

Remarks: All text on the PROMPT command line is taken by DOS to be the new system prompt. If no parameter is specified, the normal DOS prompt is assumed. Special meta-strings can be imbedded in the text in the form $*c*.

Where *c* is one of the following:

$	The "$" character.
t	The time.
d	The date.
p	The current directory of the default drive.
v	The version number.
n	The default drive.
g	The ">" character.
l	The "<" character.
b	The " " character.
q	The "=" character.
h	A backspace and erasure of the previous character.
e	The ESCape character.
–	The CR LF sequence (go to beginning of new line on the display screen).

PROMPT (Set System Prompt) Command

Any other character is treated as a null character—no action is taken on it by the PROMPT command.

Example: In this example, the command would set the normal DOS prompt:

```
PROMPT $n$g
```

In this example, the command would set ABC as the system prompt:

```
PROMPT ABC
```

In this example, the command would set a two line prompt that displays:

```
Time = (current time)

Date = (current date)

PROMPT Time = $t$ __ Date = $d
```

If you wish to create a prompt that begins with any of the DOS command delimiters (such as semicolon, blank, etc.), you can precede that character with a null meta-string. In this case, the character will be treated as the first character of the prompt, rather than as a delimiter between the word PROMPT and its parameter. For example:

```
PROMPT $A;ABC
```

causes the PROMPT command to interpret the $A as a null character, because A is not one of the defined characters in the above list. All characters following the null character will become the new system prompt.

SET (Set Environment) Command

Purpose: This command inserts strings into the command processor's environment. The entire series of strings in the environment is made available to all commands and applications.

Format: SET [*name*=[*parameter*]]

Type: Internal *** External

Remarks: The entire string (beginning with *name*) is inserted into a block of memory reserved for environment strings. Any lowercase letters in the name are converted to uppercase letters when added to the environment; the remainder of the line is inserted as you entered it. If the name already existed in the environment, it is replaced with the new *parameter*.

If the SET command is entered with no *name* specified, then the current set of environment strings will be displayed.

If a *name* is specified, but the *parameter* is not specified, then the current occurrence of *name=parameter* is removed from the environment.

SET (Set Environment) Command

The environment (series of names and parameters) is made available to all DOS commands and applications (see the "Program Segment Prefix description" in Appendix E). You can display the current environment contents by entering a SET command with no parameters. You can select the strings in the environment. For example, entering:

```
SET abc=xyz
```

will add the string ABC=xyz to the other strings already in the environment (note the conversion of abc to uppercase ABC). In this way, it is possible for you to enter keywords and parameters that are not meaningful to DOS, but can be found and interpreted by applications that are designed to examine the environment.

Example: This example will add the string PGMS= LEVEL2 to the environment. When an application program receives control, it could search the environment for the name PGMS, and use the supplied parameter as the directory name to use for its files:

```
SET PGMS=\LEVEL2
```

SET (Set Environment) Command

The following example would remove PGMS= LEVEL2 from the environment:

SET PGMS=

Notes:

1. DOS automatically adds any PROMPT or PATH commands to the environment when you enter them. You do not need to use the SET command to add either of these two commands to the environment.

2. One of the strings in the environment (placed there by DOS when it starts up) will always be a COMSPEC = parameter. That parameter describes the path that DOS uses to reload the command processor when necessary.

3. If you have *not* loaded a program that remains resident (such as MODE, PRINT, GRAPHICS, etc.), DOS expands the environment string area to hold additional strings. If you *have* loaded a program that remains resident, DOS is unable to expand the environment area beyond 127 bytes or if the environment area has already expanded beyond 127 bytes when you load a program that is to remain resident, DOS is unable to expand the environment area beyond that point. The message **Out of environment space** appears if you issue a SET command that would cause the combined environment strings to exceed 127 bytes.

SORT Filter Command

Purpose: This filter command reads data from the standard input device, sorts the data, then writes the data to the standard output device.

Format: SORT [/R] [/+*n*]

Type: Internal External
***

Remarks: Sorts are done using the ASCII collating sequence. Tab characters are not expanded with blanks.

The /R parameter will reverse the sort, for example make "Z" come before "A."

The /+ *n* parameter is an integer that starts the sort with column *n*. If no parameters are specified, the sort starts with column 1. The maximum file size that can be sorted is 63K.

Example: In this example, the command line will read the file UNSORT.TXT, do a reverse sort, then write the output to file SORT.TXT:

```
A>SORT /R <UNSORT.TXT >SORT.TXT
```

In the next example, the command line causes the output of the directory command to be piped to the SORT filter. The SORT filter will sort starting with column 14 (this is the column the file size starts in), then send the output to the console. Thus, a directory sorted by file size will be the result:

```
A>DIR ¦ SORT /+14
```

Summary of Advanced DOS Commands

The following chart is provided for quick reference. The section called "Format Notation" at the beginning of Chapter 6 explains the notation used in the format of the commands.

> **Note:** In the column labeled **Type**, the I stands for Internal and the E stands for External.

Command	Type	Purpose	Format
CTTY	I	Change to an auxiliary console	CTTY *device-name*
EXE2BIN	E	Converts .EXE files to .COM format	EXE2BIN [*d*:][*path*]*filename*[*.ext*]] [*d*:][*path*][*filename*[*.ext*]]
FIND	E	Searches files for strings of text	FIND [/V] [/C] [/N] *string*[[*d*:][*path*] *filename*[*.ext*]...]
MORE	E	Displays a screen full of data	MORE
PROMPT	E	Set new prompt	PROMPT [*prompt-text*]
SET	I	Inserts strings into the command processor's environment	SET [*name*=[*parameter*]]
SORT	E	Sorts text data	SORT [/R] [/+*n*]

Figure 4. DOS Advanced Commands

Notes:

Chapter 11. The Linker (LINK) Program

Contents

Introduction

The linker (LINK) program is a program that:

- Combines separately produced object modules
- Searches library files for definitions of unresolved external references
- Resolves external cross-references
- Produces a printable listing that shows the resolution of external references and error messages
- Produces a relocatable load module

The LINK program resides on your DOS Supplemental Program Diskette. In this chapter, we show you how to use LINK. You should read all of this chapter before you start LINK.

Files

The linker processes the following input, output, and temporary files:

Input Files

Type	Default .ext	Override .ext	Produced by
Object	.OBJ	Yes	Compiler[1] or MACRO Assembler
Library	.LIB	Yes	Compiler
Automatic Response	(None)	N/A*	User

Figure 5. Input files used by the linker

*N/A - Not applicable.

[1] One of the optional compiler packages available for use with the IBM Personal Computer DOS.

Output Files

Type	Default .ext	Override .ext	Used by
Listing	.MAP	Yes	User
Run	.EXE	No	Relocatable loader (COMMAND.COM)

Figure 6. Output files used by the Linker

VM.TMP (Temporary File)

LINK uses as much memory as is available to hold the data that defines the load module being created. If the module is too large to be processed with the available amount of memory, the linker may need additional memory space. If this happens, a temporary file called VM.TMP is created on the DOS default drive.

When the overflow to the VM.TMP file has begun, the linker displays the following message:

```
VM.TMP has been created
Do not change diskette in drive x
```

If the VM.TMP file has been created on diskette, you should not remove the diskette until LINK ends. When LINK ends, it erases the VM.TMP file.

If the DOS default drive already has a file by the name of VM.TMP, it will be deleted by LINK and a new file will be allocated; the contents of the previous file are destroyed. Therefore, you should avoid using VM.TMP as one of your own filenames.

Definitions

Segment, *group*, and *class* are terms that appear in this chapter and in some of the messages in Chapter 8. These terms describe the underlying function of LINK. An understanding of the concepts that define these terms provides a basic understanding of the way LINK works.

Segment

A *segment* is a contiguous area of memory up to 64K bytes in length. A segment may be located anywhere in memory on a *paragraph* (16-byte) boundary. Each of the four segment registers defines a segment. The segments can overlap. Each 16-bit address is an offset from the beginning of a segment. The contents of a segment are addressed by a segment register/offset pair.

The contents of various portions of the segment are determined when machine language is generated.

Neither size nor location is necessarily fixed by the compiler or assembler because this portion of the segment may be combined at link time with other portions forming a single segment.

A program's ultimate location in memory is determined at load time by the relocation loader facility provided in COMMAND.COM, based on whether you specify the /HIGH parameter. The /HIGH parameter is discussed later in this chapter.

Group

A *group* is a collection of segments that fit together within a 64K-byte segment of memory. The segments are named to the group by the assembler or compiler. A program may consist of one or more groups.

The group is used for addressing segments in memory. The various portions of segments within the group are addressed by a segment base pointer plus an offset. The linker checks that the object modules of a group meet the 64K-byte constraint.

Class

A *class* is a collection of segments. The naming of segments to a class affects the order and relative placement of segments in memory. The class name is specified by the assembler or compiler. All portions assigned to the same class name are loaded into memory contiguously.

The segments are ordered within a class in the order that the linker encounters the segments in the object files. One class precedes another in memory only if a segment for the first class precedes all segments for the second class in the input to LINK. Classes are not restricted in size. The classes are divided into groups for addressing.

Command Prompts

After you start the linker session, you receive a series of four prompts. You can respond to these prompts from the keyboard, respond to these prompts on the command line, or you can use a special diskette file called an *automatic response file* to respond to the prompts. An example of an automatic response file is provided in this chapter.

LINK prompts you for the names of the object, run, list, and library files. When the session is finished, LINK returns to DOS and the DOS prompt is displayed. If linking is unsuccessful, LINK displays a message.

The prompts are described in order of their appearance on the screen. Defaults are shown in square brackets ([ ]) after the prompt. In the response column of the table, square brackets indicate optional entries. **Object Modules** is the only prompt that requires a response from you.

PROMPT	RESPONSES
Object Modules [.OBJ]:	[*d:*][*path*]*filename*[*.ext*] [+[*d:*][*path*]*filename*[*.ext*]]...
Run File [filename.EXE]:	[*d:*][*path*][*filename*[*.ext*]]
List File [NUL.MAP]:	[*d:*][*path*][*filename*[*.ext*]]
Libraries [.LIB]:	[*d:*][[*path*]*filename*[*.ext*]] [+[*d:*][[*path*]*filename*[*.ext*]]]...

Notes:

1. If you enter a filename without specifying the drive, the default drive is assumed. If you enter a filename without specifying the path, the default path is assumed. The libraries prompt is an exception—the linker will look for the libraries on the default drive and if not found, look on the drive specified by the compiler.

2. You can end the linker session prior to its normal end by pressing Ctrl-Break.

Detailed Description of the Command Prompts

The following detailed descriptions contain information about the responses that you can enter to the prompts.

Object Modules [.OBJ]:

Enter one or more file locations for the object modules to be linked. Multiple file locations must be separated by single plus (+) signs or blanks. If the extension is omitted from any filename, LINK assumes the filename extension .OBJ. If an object module has a different filename extension, the extension must be specified. Object filenames can not begin with the @ symbol (@ is reserved for using an automatic response file).

LINK loads segments into classes in the order encountered.

If you specify an object module on a diskette drive, but LINK cannot locate the file, it displays the following prompt:

Cannot find file *object module*
change diskette <hit ENTER>

If you specify an object module on a non-removable media (like a fixed disk), the linker session will end with the following message:

Cannot find file ***object module***

You should insert the diskette containing the requested module. This permits .OBJ files from several diskettes to be included. On a single-drive system, diskette exchanging can be done safely *only* if VM.TMP has *not* been opened. As explained in the discussion of the VM.TMP file earlier in this chapter, a message will indicate if VM.TMP has been opened.

IMPORTANT: If a VM.TMP file has been opened on a diskette, you should *not* remove the diskette containing the VM.TMP file. Remember, once a VM.TMP file is opened on a diskette, the diskette it resides on cannot be removed.

After a VM.TMP file has been opened, if you specified an object module on the same disk that VM.TMP is on and LINK cannot find it, the linker session ends with the message:

Cannot find file ***object module***

Run File [filename.EXE]:

The file specification you enter is created to store the run (executable) file that results from the LINK session. All run files receive the filename extension .EXE, even if you specify another extension. If you specify another extension, your specified extension is ignored.

The default filename for the run file prompt is the first filename specified on the object module prompt.

You can specify just a drive letter, or a path on the run file prompt. This changes the place where the run file *filename*.EXE is placed.

List File [NUL.MAP]:

The linker list file is sometimes called the linker *map*.

The list file is not created unless you specifically request it. You can request it by overriding the default with a drive letter, path, or *filename*[.*ext*]. If you do not include a filename extension, the default extension .MAP is used. If you do not enter anything, the DOS reserved filename NULL specifies that no list file will be created.

The list file contains an entry for each segment in the input (object) modules. Each entry also shows the offset (addressing) in the run file.

You can specify just a drive letter or a path on the list file prompt. This changes the place where the list file is placed.

> **Note:** If the list file is allocated to a file on diskette, that diskette must not be removed until the LINK has ended.

If you specify an object module on the same diskette drive as the diskette drive to which the list file is allocated, and LINK cannot find the object module, the linker session ends with the message:

Cannot find file ***object module***

To avoid generating the list file on a diskette, you can specify the display or printer as the list file device. For example:

List File [NUL.MAP]: CON

If you direct the output to your display, you can also print a copy of the output by pressing Ctrl-PrtSc.

Libraries [.LIB]:

You may either list the file locations for your libraries, or just press the Enter key. If you press the Enter key, LINK defaults to the library provided as part of the Compiler package.

The LINK program will look for the Compiler package library on the default drive. If it cannot find the library there, then it will look for the library on the drive specified by the Compiler package. For linking objects from just the MACRO Assembler, there is no automatic default library search.

If you answer the library prompt, you specify a list of drive letters and [*path*]*filename.ext* separated by plus signs (+) or spaces. You can enter from one to eight library file locations. Specifying a drive letter tells linker to look on that drive instead of the Compiler package supplied drive for all subsequent libraries on the library prompt. The automatically searched library file specifications are conceptually placed at the end of the response to the library prompt.

LINK searches the library files in the order in which they are listed to resolve external references. When LINK finds the module that defines the external symbol, the module is processed as another object module.

If two or more libraries have the same filename, regardless of the location, only the first library in the list is searched.

When LINK cannot find a library file, it displays a message like this:

Cannot find library A:*library file*
Enter new drive letter:

The drive that the indicated library is located on must be entered.

The following library prompt responses may be used:

Libraries [.LIB]: B:

Look for compiler .LIB on drive B.

Libraries [.LIB]: B:USERLIB

Look for USERLIB.LIB on drive B and compiler .LIB on drive A.

Libraries [.LIB]: A:LIB1+LIB2+B:LIB3+A:

Look for LIB1.LIB and LIB2.LIB on drive A, LIB3.LIB on drive B, and compiler.LIB on drive A.

Linker Parameters

At the end of any of the four linker prompts, you may specify one or more parameters that instruct the linker to do something differently. Only the / and first letter of any parameter are required.

/DSALLOCATION

The /DSALLOCATION (/D) parameter directs LINK to load all data defined to be in DGROUP at the *high end* of the group. If the /HIGH parameter is specified, (module loaded high), this allows any available storage below the specifically allocated area within DGROUP to be allocated dynamically by your application and still be addressable by the same data space pointer.

> **Note:** The maximum amount of storage which can be dynamically allocated by the application is 64K (or the amount actually available) minus the allocated portion of DGROUP.

If the /DSALLOCATION parameter is not specified, LINK loads all data defined to be in the group whose group name is DGROUP at the *low end* of the group, beginning at an offset of 0. The only storage thus referenced by the data space pointer should be that specifically defined as residing in the group.

All other segments of any type in any GROUP other than DGROUP are loaded at the low end of their respective groups, as if the /DSALLOCATION parameter were not specified.

For certain compiler packages, /DSALLOCATION is automatically used.

/HIGH

The /HIGH (/H) parameter causes the loader to place the run image as high as possible in storage. If you specify the /HIGH parameter, you tell the linker to cause the loader to place the run file as high as possible without overlaying the transient portion of COMMAND.COM, which occupies the highest area of storage when loaded. If you do not specify the /HIGH parameter, the linker directs the loader to place the run file as low in memory as possible.

The /HIGH parameter is used with the /DSALLOCATION parameter.

/LINE

For certain IBM Personal Computer language processors, the /LINE (/L) parameter directs LINK to include the line numbers and addresses of the source statements in the input modules in the list file.

/MAP

The /MAP (/M) parameter directs LINK to list all public (global) symbols defined in the input modules. For each symbol, LINK lists its value and segment-offset location in the run file. The symbols are listed at the end of the list file.

/PAUSE

The /PAUSE (/P) parameter tells LINK to display a message to you as follows:

```
About to generate .EXE file
Change disks <hit ENTER>
```

This message allows you to insert the diskette that is to contain the run file.

/STACK:size

The *size* entry is any positive decimal value up to 65536 bytes. This value is used to override the size of the stack that the MACRO Assembler or compiler has provided for the load module being created. If you specify a value greater than 0 but less than 512, the value 512 is used.

If you do not specify /STACK (/S), the original stack size provided by the MACRO Assembler or compiler is used.

If the size of the stack is too small, the results of executing the resulting load module are unpredictable.

At least one input (object) module must contain a stack allocation statement, unless you plan to use the EXE2BIN program. This is automatically provided by compilers. For the MACRO Assembler, the source must contain a SEGMENT command that has the combine type of STACK. If a stack allocation statement was not provided, LINK returns a **Warning: No Stack statement** message.

How to Start the Linker Program

Before You Begin

- Make sure the files you will be using for linking are on the appropriate disks.
- Make sure you have enough free space on your disks to contain your files and any generated data.

You can start the linker program by using one of three options:

Option 1 – Console Responses

From your keyboard, enter:

```
LINK
```

The linker is loaded into memory and displays a series of four prompts, one at a time, to which you must enter the requested responses. (Detailed descriptions of the responses that you can make to the prompts are discussed in this chapter.)

If you enter a wrong response, such as an incorrectly spelled filename, you must press Ctrl-Break to exit LINK, then restart LINK. If the response in error has been typed but you haven't pressed Enter yet, you may delete the wrong characters (on that line only).

An example of a linker session using the console response option is provided in this chapter in the section called "Example Linker Session."

As soon as you have entered the last filename, the linker begins to run. If the linker finds any errors, it displays the errors on the screen as well as in the listing file.

> **Note:** After any of these responses, before pressing Enter, you may continue the response with a comma and the answer to what would be the next prompt, without having to wait for that prompt. If you end any with the semicolon (;), the remaining responses are all assumed to be the default. Processing begins immediately with no further prompting.

Option 2 – Command Line

From your keyboard, enter:

LINK *objlist,runfile,mapfile,liblist* [*parm*]...;

objlist is a list of object modules separated by spaces or plus signs (+).

runfile is the name you want to give the run file.

mapfile is the name you want to give the linker map.

liblist is a list of the libraries to be used, separated by plus signs (+) or spaces.

parm is an optional linker parameter. Each parameter must begin with a slash (/).

The linker is loaded and immediately performs the tasks indicated by the command line.

When you use this command line, the prompts described in Option 1 are not displayed if you specified an entry for all four files or if the command line ends with a semicolon.

If an incomplete list is given and no semicolon is used, the linker prompts for the remaining unspecified files.

Each prompt displays its default, which may be accepted by pressing the Enter key, or overridden with an explicit filename or device name. However, if an incomplete list is given and the command line is terminated with a final semicolon, the unspecified files default without further prompting. The *parms* are never prompted for, but may be added to the end of the command line or to any file specification given in response to a prompt.

Certain variations of this command line are permitted.

Examples:

LINK module

The object module is MODULE.OBJ. A prompt is given, showing the default of MODULE.EXE. After the response is entered, a prompt is given showing the default of NUL.MAP. After the response is given, a prompt is displayed showing the default extension of .LIB.

LINK module;

If the semicolon is added, no further prompts are displayed. The object module of MODULE.OBJ is linked, the run file is put into MODULE.EXE, and no list file is produced.

LINK module,,;

This is similar to the preceding example, except the list file is produced in MODULE.MAP.

LINK module,,

Using the same example, but without the semicolon, MODULE.OBJ is linked, and the run file is produced in MODULE.EXE, but a prompt is given with the default of MODULE.MAP.

LINK module,,NUL;

No list file is produced. The run file is in MODULE.EXE. No further prompts are displayed.

Option 3 – Automatic Responses

It is often convenient to save responses to the linker for use at a later time. This is especially useful when long lists of object modules need to be specified.

Before using this option, you must create the automatic response file. It contains several lines of text, each of which is the response to a linker prompt. These responses must be in the same order as the linker prompts that were discussed earlier in this chapter. If desired, a long response to the object module or libraries prompt may be contained on several lines by using a plus sign (+) to continue the same response onto the next line.

To specify an automatic response file, you enter a file specification preceded by an @ symbol in place of a prompt response or part of a prompt response. The prompt is answered by the contents of the diskette file. The file specification may not be a reserved DOS filename.

From your keyboard, enter:

LINK @[*d*:][*path*]*filename*1[*ext*]

Use of the filename extension is optional and may be any name. There is no default extension.

Use of this option permits the command that starts LINK to be entered from the keyboard or within a batch file without requiring any response from you.

Example

Automatic Response File – RESP1

```
MODA+MODB+MODC+
MODB+MODE+MODF
```

```
runfile/p
printout
```

Command line

```
LINK @RESP1+mymod,@RESP2;
```

Notes:

1. The plus sign at the end of the first line in RESP1 causes the modules listed in the first two lines to be considered as the input object modules. After reading RESP1, the linker returns to the command line and sees +mymod, so it includes MYMOD.OBJ in the first list of object modules as well.

2. Each of the above lines ends when you press the Enter key.

Example Linker Session

This example shows you the type of information that is displayed during a linker session.

Once we enter:

```
b:link
```

in response to the DOS prompt, the system responds with the following messages and prompts, which we answer as shown:

```
IBM Personal Computer Linker
Version 2.00 (C)Copyright IBM Corp. 1981, 1982, 1983

Object Modules [.OBJ]: example
Run File [EXAMPLE.EXE]: /map
List File [NUL.MAP]: prn/line
Libraries [.LIB]:
```

Notes:

1. By specifying /map, we get both an alphabetic listing and a chronological listing of public symbols.

2. By responding prn to the list file prompt, we send our output to the printer.

3. By specifying the /LINE parameter, LINK gives us a listing of all line numbers for all modules. (The /LINE parameter can generate a large amount of output.)

4. By just pressing Enter in response to the libraries prompt, an automatic library search is performed.

Once LINK locates all libraries, the linker map displays a list of segments in the relative order of their appearance within the load module. The list looks like this:

Start	Stop	Length	Name	Class
00000H	00028H	0029H	MAINQQ	CODE
00030H	000F6H	00C7H	ENTXQQ	CODE
00100H	00100H	0000H	INIXQQ	CODE
00100H	038D3H	37D4H	FILVQQ__CODE	CODE
038D4H	04921H	104EH	FILUQQ__CODE	CODE
•				
•				
•				
074A0H	074A0H	0000H	HEAP	MEMORY
074A0H	074A0H	0000H	MEMORY	MEMORY
074A0H	0759FH	0100H	STACK	STACK
075A0H	07925H	0386H	DATA	DATA
07930H	082A9H	097AH	CONST	CONST

The information in the Start and Stop columns shows a 20-bit hex address of each segment relative to location zero. Location zero is the beginning of the load module. The addresses displayed are not the absolute addresses of where these segments are loaded. To find the absolute address of where a segment is actually loaded, you must determine where the segment listed as being at relative zero is actually loaded; then add the absolute address to the relative address shown in the linker map. The procedure you use to determine where relative zero is actually located is discussed in this chapter, in the section called "How to Determine the Absolute Address of a Segment."

Now, because we specified the /MAP parameter, the public symbols are displayed by name and by value. For example:

Address	Publics by Name
0492:0003H	ABSNQQ
06CD:029FH	ABSRQQ
0492:00A3H	ADDNQQ
06CD:0087H	ADDRQQ
0602:000FH	ALLHQQ
•	
•	
•	
0010:1BCEH	WT4VQQ
0010:1D7EH	WTFVQQ
0010:1887H	WTIVQQ
0010:19E2H	WTNVQQ
0010:11B2H	WTRVQQ

Address	Publics by Value
0000:0001H	MAIN
0000:0010H	ENTGQQ
0000:0010H	MAINQQ
0003:0000H	BEGXQQ
0003:0095H	ENDXQQ
•	
•	
•	
F82B:F31CH	CRCXQQ
F82B:F31EH	CRDXQQ
F82B:F322H	CESXQQ
F82B:F5B8H	FNSUQQ
F82B:F5E0H	OUTUQQ

The addresses of the public symbols are in the *segment:offset* format, showing the location relative to zero as the beginning of the load module. In some cases, an entry may look like this:

```
F8CC:EBE2H
```

This entry appears to be the address of a load module that is almost one megabyte in size. Actually, the area being referenced is relative to a segment base that is pointing to a segment below the relative zero beginning of the load module. This condition produces a pointer that has effectively gone negative. The memory map which follows illustrates this point.

When LINK has completed, the following message is displayed:

```
Program entry point at 0003:0000
```

How to Determine the Absolute Address of a Segment

The linker map displays a list of segments in the relative order of their appearance within the load module. The information displayed shows a 20-bit hex address of each segment relative to location zero. The addresses that are displayed are not the absolute addresses of where these segments are actually located. To determine where relative zero is actually located, we must use DEBUG. DEBUG is described in detail in Chapter 12.

Using DEBUG,

1. Load the application. Note the segment value in CS and the offset within that segment to the entry point as shown in IP. The last line of the linker map also describes this entry point, but uses relative values, not the absolute values shown by CS and IP.

2. Subtract the relative entry as shown at the end of the map listing from the CS:IP value. For example, let's say CS is at 05DC and IP is at zero.

   The linker map shows the entry point at 0100:0000. (0100 is a segment ID or paragraph number; 0000 is the offset into that segment.)

   In this example, relative zero is located at 04DC:0000, which is 04DC0 absolute.

If a program is loaded low, the relative zero location is located at the end of the Program Segment Prefix, in the location DS plus 100H.

Messages

All messages, except for the warning messages, cause the LINK session to end. Therefore, after you locate and correct a problem, you must rerun LINK.

Messages appear both in the list file and on the display unless you direct the list file to CON, in which case the display messages are suppressed.

All of the linker messages are included in Chapter 8.

Chapter 12. The DEBUG Program

Contents

Introduction

This chapter explains how to use the DEBUG program.

The DEBUG program can be used to:

- Provide a controlled testing environment so you can monitor and control the execution of a program to be debugged. You can fix problems in your program directly, and then execute the program immediately to determine if the problems have been resolved. You do not need to reassemble a program to find out if your changes worked.

- Load, alter, or display any file.

- Execute *object files*. Object files are executable programs in machine language format.

How to Start the DEBUG Program

To start DEBUG, enter:

DEBUG [*d*][*path*][*filename*[.*ext*]][*parm1*][*parm2*]

If you enter *filename*, the DEBUG program loads the specified file into memory. You may now enter commands to alter, display, or execute the contents of the specified file.

If you do *not* enter a filename, you must either work with the present memory contents, or load the required file into memory by using the Name and Load commands. Then you can enter commands to alter, display, or execute the memory contents.

The optional parameters, *parm1* and *parm2*, represent the optional parameters for the named *filespec*. For example,

DEBUG DISKCOMP.COM A: B:

In this command, the A: and B: are the parameters that DEBUG prepares for the DISKCOMP program.

When the DEBUG program starts, the registers and flags are set to the following values for the program being debugged:

- The segment registers (CS, DS, ES, and SS) are set to the bottom of free memory; that is, the first segment after the end of the DEBUG program.

- The Instruction Pointer (IP) is set to hex 0100.

- The Stack Pointer (SP) is set to the end of the segment, or the bottom of the transient portion of the program loader, whichever is lower. The segment size at offset 6 is reduced by hex 100 to allow for a stack of that size.

- The remaining registers (AX, BX, CX, DX, BP, SI, and DI) are set to zero. However, if you start the DEBUG program with a filespec, the CX register contains the length of the file in bytes. If the file is greater than 64K, the length is contained in registers BX and CX (the high portion in BX).

- The flags are set to their cleared values. (Refer to the Register command.)

- The default disk transfer address is set to hex 80 in the code segment.

All of available memory is allocated; therefore, any attempt by the loaded program to allocate memory will fail.

Notes:

1. If a file loaded by DEBUG has an extension of .EXE, DEBUG does the necessary relocation and sets the segment registers, stack pointer, and Instruction Pointer to the values defined in the file. The DS and ES registers, however, point to the Program Segment Prefix at the lowest available segment. The BX and CX registers contain the size of the program (smaller than the file size).

   The program is loaded at the high end of memory if the appropriate parameter was specified when the linker created the file. Refer to ".EXE File Structure and Loading" in Appendix H for more information about loading .EXE files.

2. If a file loaded by DEBUG has an extension of .HEX, the file is assumed to contain ASCII representation of hexadecimal characters, and is converted to binary while being loaded.

The DEBUG Command Parameters

Parameter	Definition
address	Enter a one- or two-part designation in one of the following formats: • An alphabetic segment register designation, plus an offset value, such as: **CS:0100** • A segment address, plus an offset value, such as: **4BA:0100** • An offset value only, such as: **100** (In this case, each command uses a default segment.)

Parameter	Definition
address (cont.)	**Notes:** 1. In the first two formats, the colon is required to separate the values. 2. All numeric values are *hexadecimal* and may be entered as 1-4 characters. 3. The memory locations specified in address must be valid; that is, they must actually exist. Unpredictable results will occur if an attempt is made to access a nonexistent memory location.
byte	Enter a one or two character *hexadecimal* value.
drive	Enter a single digit (for example, 0 for drive A or 1 for drive B) to indicate which drive data is to be loaded from or written to. (Refer to the Load and Write commands.)
filespec	Enter a one- to three-part file specification consisting of a drive designation, filename, and filename extension. All three fields are optional. However, for the Name command to be meaningful, you should at least specify a drive designator or a filename. (Refer to the Name command.)

Parameter	Definition
list	Enter one or more byte and/or string values. For example, **E CS:100 F3 'XYZ' 8D 4 "abcd"** has five items in the list (that is, three byte entries and two string entries having a total of 10 bytes).
portaddress	Enter a 1-4 character *hexadecimal* value to specify an 8- or 16-bit port address. (Refer to the Input and Output commands.)
range	Enter either of the following formats to specify the lower and upper addresses of a range: • *address address* For example: **CS:100 110** Note: Only an offset value is allowed in the second address. The addresses must be separated by a space or comma.

Parameter	Definition
range (cont.)	• *address* L *value* where *value* is the number of bytes in *hexadecimal* to be processed by the command. For example: **CS:100 L 11** Notes: 1. The limit for *range* is hex 10000. To specify that *value* within four *hexadecimal* characters, enter 0000 (or 0). 2. The memory locations specified in *range* must be valid; that is, they must actually exist. Unpredictable results will occur if an attempt is made to access a non-existent memory location.
registername	Refer to the Register command.

Parameter	Definition
sector sector	Enter 1-3 character *hexadecimal* values to specify: 1. The starting relative sector number 2. The number of sector numbers to be loaded or written In DEBUG, relative sectors are obtained by counting the sectors on the disk surface. The sector at track 0, sector 1, head 0 (the first sector on the disk) is relative sector 0. The numbering continues for each sector on that track and head, then continues with the first sector on the next head of the same track. When all sectors on all heads of the track have been counted, numbering continues with the first sector on head 0 of the next track. **Note:** This is a change from the sector mapping used by DOS Version 1.10. The maximum number of sectors that can be loaded or written with a single command is hex 80. A sector contains 512 bytes. (Refer to the Load and Write commands.)

Parameter	Definition
string	Enter characters enclosed in quotation marks. The quotation marks can be either single (') or double ("). The ASCII values of the characters in the string are used as a list of byte values. Within a string, the *opposite* set of quotation marks can be used freely as characters. However, if the *same* set of quotation marks (as the delimiters) must be used within the string, then the quotation marks must be doubled. The doubling does not appear in memory. For example: 1. 'This "literal" is correct' 2. 'This ' 'literal' ' is correct' 3. 'This 'literal' is not correct' 4. 'This ""literal"" is not correct' 5. "This 'literal' is correct" 6. "This ""literal"" is correct" 7. "This "literal" is not correct" 8. "This ' 'literal' ' is not correct"

Parameter	Definition
string (cont.)	In the second and sixth cases above, the word *literal* is enclosed in one set of quotation marks in memory. In the fourth and eighth cases above, the word *literal* is not correct unless you really want it enclosed in two sets of quotation marks in memory.
value	Enter a 1-4 character *hexadecimal* value to specify: • The numbers to be added and subtracted (refer to the Hexarithmetic command), or • The number of instructions to be executed by the Trace command, or • The number of bytes a command should operate on. (Refer to the Dump, Fill, Move, Search, and Unassemble commands.)

The DEBUG Commands

This section presents a detailed description of how to use the commands to the DEBUG program. The commands appear in alphabetical order; each with its format and purpose. Examples are provided where appropriate.

Information Common to All DEBUG Commands

The following information applies to the DEBUG commands:

- A command is a single letter, usually followed by one or more parameters.
- Commands and parameters can be entered in uppercase or lowercase, or a combination of both.
- Commands and parameters may be separated by delimiters. Delimiters are only required, however, between two consecutive hexadecimal values. Thus, these commands are equivalent:

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

- Press Ctrl-Break to end commands.

- Commands become effective only after you press the Enter key.

- For commands producing a large amount of output, you can press Ctrl-NumLock to suspend the display to read it before it scrolls away. Press any other character to restart the display.

- You can use the control keys and the DOS editing keys, described in Chapter 3, while using the DEBUG program.

- If a syntax error is encountered, the line is displayed with the error pointed out as follows:

```
d cs:100 CS:110
         error
```

  In this case, the Dump command is expecting the second address to contain only a hexadecimal offset value. It finds the S, which is not a valid hexadecimal character.

- The prompt from the DEBUG program is a hyphen (-).

- The DEBUG program resides on your DOS Supplemental Program diskette.

Assemble Command

Purpose: To assemble IBM Personal Computer Macro Assembler language statements directly into memory.

Format: A[*address*]

Remarks: All numeric input to the Assemble command is in hexadecimal. The assembly statements you enter are assembled into memory at successive locations, starting with the address specified in *address*. If no address is specified, the statements are assembled into the area at CS:0100, if no previous Assemble command was used, or into the location following the last instruction assembled by a previous Assemble command. When all desired statements have been entered, press Enter when you are prompted for the next statement, to return to the DEBUG prompt.

DEBUG responds to invalid statements by displaying:

```
^ Error
```

and redisplaying the current assemble address.

Assemble Command

DEBUG supports standard 8086/8088 assembly language syntax (and the 8087 instruction set), with the following rules:

- All numeric values entered are hexadecimal and can be entered as 1-4 characters.
- Prefix mnemonics must be entered in front of the opcode to which they refer. They can also be entered on a separate line.
- The segment override mnemonics are CS:, DS:, ES:, and SS:.
- String manipulation mnemonics must explicitly state the string size. For example, MOVSW must be used to move word strings and MOVSB must be used to move byte strings.
- The mnemonic for the far return is RETF.

- The assembler will automatically assemble short, near, or far jumps and calls depending on byte displacement to the destination address. These can be overridden with the NEAR OR FAR prefix. For example:

```
0100:0500 JMP 502          ; a 2 byte short jump
0100:0502 JMP NEAR 505     ; a 3 byte near jump
0100:0505 JMP FAR 50A      ; a 5 byte far jump
```

  The NEAR prefix can be abbreviated to NE, but the FAR prefix cannot be abbreviated.

- DEBUG cannot tell whether some operands refer to a word memory location or a byte memory location. In this case, the data type must be explicitly stated with the prefix "WORD PTR" or "BYTE PTR". DEBUG will also accept the abbreviations "WO" and "BY". For example:

```
NEG BYTE PTR [128]
DEC WO [SI]
```

- DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV AX,21          ;Load AX with 21H
MOV AX,[21]        ;Load AX with the
                    contents of memory
                    location 21H
```

- Two popular pseudo-instructions have also been included. The DB opcode will assemble byte values directly into memory. The DW opcode will assemble word values directly into memory. For example:

```
DB   1,2,3,4,"THIS IS AN EXAMPLE"
DB   'THIS IS A QUOTE: " '
DB   "THIS IS A QUOTE: ' "

DW   1000,2000,3000",BACH"
```

- All forms of the register indirect commands are supported. For example:

```
ADD   BX,34[BP+2].[SI-1]
POP   [BP+DI]
PUSH  [SI]
```

Assemble Command

- All opcode synonyms are supported. For example:

```
LOOPZ   100
LOOPE   100

JA      200
JNBE    200
```

- For 8087 opcodes the WAIT or FWAIT prefix must be explicitly specified. For example:

```
FWAIT FADD ST,ST(3) ;This line will assemble
                      a FWAIT prefix

FLD TBYTE PTR [BX]  ;This line will not
```

Example:

```
C>debug
-a200
08B4:0200 xor ax,ax
08B4:0202 mov [bx],ax
08B4:0204 ret
08B4:0205
```

Compare Command

Purpose: Compares the contents of two blocks of memory.

Format: C *range address*

Remarks: The contents of the two blocks of memory are compared; the length of the comparison is determined from the *range*. If unequal bytes are found, their addresses and contents are displayed, in the form:

addr1 byte1 byte2 addr2

where, the first half (addr1 byte1) refers to the location and contents of the mismatching locations in *range*, and the second half (byte2 addr2) refers to the byte found in *address*.

If you enter only an offset for the beginning address of *range*, the C command assumes the segment contained in the DS register. To specify an ending address for *range*, enter it with only an offset value.

Example: **C 100L20 200**

The 32 bytes of memory beginning at DS:100 are compared with the 32 bytes beginning at DS:200.

Dump Command

Purpose: Displays the contents of a portion of memory.

Format: D [*address*]

or

D [*range*]

Remarks: The dump is displayed in two parts:

1. A hexadecimal portion. Each byte is displayed in hexadecimal.

2. An ASCII portion. The bytes are displayed as ASCII characters. Unprintable characters are indicated by a period (.).

With a 40-column system display format, each line begins on an 8-byte boundary and shows 8 bytes.

With an 80-column system display format, each line begins on a 16-byte boundary and shows 16 bytes. There is a hyphen between the 8th and 9th bytes.

> **Note:** The first line may have fewer than 8 or 16 bytes if the starting address of the dump is not on a boundary. In this case, the second line of the dump begins on a boundary.

Dump Command

The Dump command has two format options:

Option 1

Use this option to display the contents of hex 40 bytes (40-column mode) or hex 80 bytes (80-column mode). For example:

D *address*

or

D

The contents are dumped starting with the specified address.

If you do not specify an address, the D command assumes the starting address is the location following the last location displayed by a previous D command. Thus, it is possible to dump consecutive 40-byte or 80-byte areas by entering consecutive D commands without parameters.

If no previous D command was entered, the location is offset hex 0100 into the segment originally initialized in the segment registers by DEBUG.

> **Note:** If you enter only an offset for the starting address, the D command assumes the segment contained in the DS register.

Option 2

Use this option to display the contents of the specified address range. For example:

D *range*

Note: If you enter only an offset for the starting address, the D command assumes the segment contained in the DS register. If you specify an ending address, enter it with only an offset value.

For example:

```
D cs:100 10C
```

A 40-column display format might look like this:

```
04BA:0100  42 45 52 54 41 20 54 00
                      BERTA T.

04BA:0108  20 42 4F 52 47
                      BORG
```

Enter Command

Purpose: The Enter command has two modes of operation:

- Replaces the contents of one or more bytes, starting at the specified address, with the values contained in the list. (See Option 1.)
- Displays and allows modification of bytes in a sequential manner. (See Option 2.)

Format: E *address* [*list*]

Remarks: If you enter only an offset for the address, the E command assumes the segment contained in the DS register.

The Enter command has two format options:

Option 1

Use this option to place the list in memory beginning at the specified address.

E *address list*

For example:

```
E ds:100 F3 "xyz" 8D
```

Memory locations ds:100 through ds:104 are filled with the five bytes specified in the list.

Option 2

Use this option to display the address and the byte of a location, then the system waits for your input.

For example:

E *address*

Now you can take one of the following actions:

1. Enter a one or two character *hexadecimal* value to replace the contents of the byte; then take any of the next three actions:

2. Press the space bar to advance to the next address. Its contents are displayed. If you want to change the contents take action 1, above.

   To advance to the next byte without changing the current byte, press the space bar again.

3. Enter a hyphen (-) to back up to the preceding address. A new line is displayed with the preceding address and its contents. If you want to change the contents take action 1, above.

   To back up one more byte without changing the current byte, enter another hyphen.

4. Press the Enter key to end the Enter command.

   Note: Display lines can have 4 or 8 bytes of data, depending on whether the system display format is 40- or 80-column. Spacing beyond an 8-byte boundary causes a new display line, with the beginning address, to be started.

   For example:

   ```
   E cs:100
   ```

   might cause this display:

   ```
   04BA:0100 EB._
   ```

   To change the contents of 04BA:0100 from hex EB to hex 41, enter 41.

   ```
   04BA:0100 EB.41 _
   ```

   To see the contents of the next three locations, press the space bar three times. The screen might look like this:

   ```
   04BA:0100  EB.41  10.  00.  BC._
   ```

   To change the contents of the current location (04BA:0103) from hex BC to hex 42, enter 42.

   ```
   04BA:0100  EB.41  10.  00.  BC.42_
   ```

Now, suppose you want to back up and change the hex 10 to hex 6F. This is what the screen would look like after entering two hyphens and the replacement byte:

```
04BA:0100  EB.41  10.  00.  BC.42-
04BA:0102  00.-
04BA:0101  10.6F_
```

Press the Enter key to end the Enter command. You will see the hyphen (-) prompt.

Fill Command

Purpose: Fills the memory locations in the range with the values in the list.

Format: F *range list*

Remarks: If the list contains fewer bytes than the address range, the list is used repeatedly until all the designated memory locations are filled.

If the list contains more bytes than the address range, the extra list items are ignored.

> **Note:** If you enter only an offset for the starting address of the range, the Fill command assumes the segment contained in the DS register.

Example: **F 4BA:100 L 5 F3 "XYZ" 8D**

Memory locations 04BA:100 through 04BA:104 are filled with the 5 bytes specified. Remember that the ASCII values of the list characters are stored. Thus, locations 100-104 will contain F3 58 59 5A 8D.

Go Command

Purpose: Executes the program you are debugging.

Stops the execution when the instruction at a specified address is reached (breakpoint), and displays the registers, flags, and the next instruction to be executed.

Format: G [=*address*] [*address* [*address*...]]

Remarks: Program execution begins with the current instruction, whose address is determined by the contents of the CS and IP registers, unless overridden by the =*address* parameter (the = must be entered). If =*address* is specified, program execution begins with CS:=*address*.

The Go command has two format options:

Option 1

Use this option to execute the program you are debugging without breakpoints. For example:

G [=*address*]

This option is useful when testing program execution with different parameters each time. (Refer to the Name command.) Be certain the CS:IP values are set properly before issuing the G command, if not using =*address*.

Option 2

This option performs the same function as Option 1 but, in addition, allows breakpoints to be set at the specified addresses. For example:

G [=*address*] *address*
[*address...*]

This method causes execution to stop at a specified location so the system/program environment can be examined.

You can specify up to ten breakpoints in any order. You may wish to take advantage of this if your program has many paths, and you want to stop the execution no matter which path the program takes.

The DEBUG program replaces the instruction codes at the breakpoint addresses with an interrupt code (hex CC). If *any one* breakpoint is reached during execution, the execution is stopped, the registers and flags are displayed, and all the breakpoint addresses are restored to their original instruction codes. If no breakpoint is reached, the instructions are *not* restored.

Go Command

Notes:

1. Once a program has reached completion (DEBUG has displayed the "Program terminated normally" message), it will be necessary to reload the program before it can be executed again.

2. Make sure that the address parameters refer to locations that contain valid 8088 instruction codes. If you specify an address that does not contain the first byte valid instruction, unpredictable results will occur.

3. The stack pointer must be valid and have 6 bytes available for the Go command; otherwise, unpredictable results will occur.

4. If only an offset is entered for a breakpoint, the G command assumes the segment contained in the CS register.

For example:

```
G 102 1EF 208
```

Execution begins with the current instruction, whose address is the current values of CS:IP. The *=address* parameter was not used.

Three breakpoints are specified; assume that the second is reached. Execution stops before the instruction at location CS:1EF is executed, the original instruction codes are restored, all three breakpoints are removed, the display occurs, and the Go command ends.

Refer to the Register command for a description of the display.

Hexarithmetic Command

Purpose: Adds the two hexadecimal values, then subtracts the second from the first.

Displays the sum and difference on one line.

Format: H *value value*

Example:

```
H 0F 8
17 07
```

The hexadecimal sum of 000F and 0008 is 0017, and their difference is 0007.

Input Command

Purpose: Inputs and displays (in hexadecimal) one byte from the specified port.

Format: I *portaddress*

Example:

```
I 2F8
6B
```

The single hexadecimal byte read from port 02F8 is displayed (6B).

Load Command

Purpose: Loads a file or absolute diskette sectors into memory.

Format: L [*address* [*drive sector sector*]]

Remarks: The maximum number of sectors that can be loaded with a single Load command is hex 80.

> **Note:** DEBUG displays a message if a diskette read error occurs. You can retry the read operation by pressing F3 to re-display the Load command. Then, press the Enter key.

The Load command has two format options:

Option 1

Use this option to load data from the diskette specified by *drive*, and place the data in memory beginning at the specified *address*. For example:

L *address drive sector sector*

The data is read from the specified starting relative sector (first sector) and continues until the requested number of sectors is read (second sector).

> Note: If you only enter an offset for the beginning address, the L command assumes the segment contained in the CS register.

For example, to load data, you might enter:

```
L 4BA:100 1 0F 6D
```

The data is loaded from the diskette in drive B and placed in memory beginning at 4BA:100. Hex 6D (109) consecutive sectors of data are transferred, starting with relative sector hex 0F (15) (the 16th sector on the diskette).

Load Command

Option 2

When issued without parameters, or with only the address parameter, use this option to load the file whose filespec is properly formatted in the file control block at CS:5C. For example:

L

or

L *address*

This condition is met by specifying the filespec when starting the DEBUG program, or by using the Name command.

> **Note:** If DEBUG was started with a filespec and subsequent Name commands were used, you may need to enter a new Name command for the proper filespec before issuing the Load command.

The file is loaded into memory beginning at CS:100 (or the location specified by *address*), and is read from the drive specified in the filespec (or from the default drive, if none was specified). Note that files with extensions of .COM or .EXE are always loaded at CS:100 – if you specified an address, it will be ignored.

The BX and CX registers are set to the number of bytes read; however, if the file being loaded has an extension of .EXE, BX and CX are set to the actual program size, and the file may be loaded at the high end of memory. Refer to the notes in "How to Start the DEBUG Program" at the beginning of this chapter for the conditions that are in effect when .EXE or .HEX files are loaded.

For example:

```
DEBUG
-N myprog
-L
-
```

The file named **myprog** is loaded from the default diskette and placed in memory beginning at location CS:0100.

Move Command

Purpose: Moves the contents of the memory locations specified by *range* to the locations beginning at the *address* specified.

Format: M *range address*

Remarks: Overlapping moves are always performed without loss of data during the transfer. (The source and destination areas share some of the same memory locations.)

The data in the source area remains unchanged unless overwritten by the move.

Notes:

1. If you enter only an offset for the beginning address of the range, the M command assumes the segment contained in the DS register. If you specify an ending address for the range, enter it with only an offset value.

2. If you enter only an offset for the address of the destination area, the M command assumes the segment contained in the DS register.

Example: **M CS:100 110 500**

The 17 bytes of data from CS:100 through CS:110 are moved to the area of memory beginning at DS:500.

Name Command

Purpose: The Name command has two functions:

- Formats file control blocks for the first two filespecs, at CS:5C and CS:6C. (Starting DEBUG with a filespec also formats a file control block at CS:5C.)

  The file control blocks are set up for the use of the Load and Write commands, and to supply required filenames for the program being debugged.

- All specified filespecs and other parameters are placed exactly as entered, including delimiters, in a parameter save area at CS:81, with CS:80 containing the number of characters entered. Register AX is set to indicate the validity of the drive specifiers entered with the first two filespecs.

Format: N *filespec* [*filespec...*]

Remarks: If you start the DEBUG program without a filespec, you must use the Name command before a file can be loaded with the L command.

Example:

```
DEBUG
-N myprog
-L
-
```

To define filespecs or other parameters required by the program being debugged, enter:

```
DEBUG myprog
-N file1 file2
-
```

In this example, DEBUG loads the file **myprog** at CS:100, and leaves the file control block at CS:5C formatted with the same filespec. Then, the Name command formats file control blocks for *file1* and *file2* at CS:5C and CS:6C, respectively. The file control block for **myprog** is overwritten. The parameter area at CS:81 contains all characters entered after the N, including all delimiters, and CS:80 contains the count of those characters (hex 0C).

Output Command

Purpose: Sends the *byte* to the specified output port.

Format: O *portaddress byte*

Example: To send the byte value 4F to output port 2F8, enter:

```
O 2F8 4F
```

Quit Command

Purpose: Ends the DEBUG program.

Format: Q

Remarks: The file that you are working on in memory is *not* saved by the Quit command. You must use the Write command to save the file.

DEBUG returns to the command processor which then issues the normal command prompt.

Example:

```
-Q
A>
```

Register Command

Purpose: The Register command has three functions:

- It displays the hexadecimal contents of a single register, with the option of changing those contents.
- It displays the hexadecimal contents of all the registers, plus the alphabetic flag settings, and the next instruction to be executed.
- It displays the eight 2-letter alphabetic flag settings, with the option of changing any or all of them.

Format: R [*registername*]

Remarks: When the DEBUG program starts, the registers and flags are set to certain values for the program being debugged. (Refer to "How to Start the DEBUG Program" at the beginning of this chapter.)

Display a Single Register

The valid *registernames* are:

AX	BP	SS
BX	SI	CS
CX	DI	IP
DX	DS	PC
SP	ES	F

Both IP and PC refer to the instruction pointer.

For example, to display the contents of a single register, you might enter:

```
R AX
```

The system might respond with:

```
AX F1E4
:_
```

Register Command

Now you may take one of two actions:

- Press Enter to leave the contents unchanged.

  or

- Change the contents of the AX register by entering a 1-4 character hexadecimal value, such as hex FFF.

```
AX F1E4
:FFF_
```

Now pressing Enter changes the contents of the AX register to hex 0FFF.

Display All Registers and Flags

To display the contents of all registers and flags (and the next instruction to be executed), enter:

```
R
```

The system might respond with:

```
AX=0E00 BX=00FF CX=0007 DX=01FF
SP=039D BP=0000 SI=005C DI=0000
DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A NV UP DI NG NZ AC PE NC
04BA:011A  CD21          INT     21
```

The first four lines display the hexadecimal contents of the registers and the eight alphabetic flag settings. The last line indicates the location of the next instruction to be executed, and its hexadecimal and unassembled formats. This is the instruction pointed to by CS:IP.

Note: A system with an 80-column display shows:

1st line - 8 registers
2nd line - 5 registers and 8 flag settings
3rd line - next instruction information

A system with a 40-column display shows:

1st line - 4 registers
2nd line - 4 registers
3rd line - 4 registers
4th line - 1 register and 8 flag settings
5th line - next instruction information

Display All Flags

There are eight flags, each with 2-letter codes to indicate either a *set* condition or a *clear* condition.

The flags appear in displays in the same order as presented in the following table:

Flag Name	Set	Clear
Overflow (yes/no)	OV	NV
Direction (decrement/increment)	DN	UP
Interrupt (enable/disable)	EI	DI
Sign (negative/positive)	NG	PL
Zero (yes/no)	ZR	NZ
Auxiliary carry (yes/no)	AC	NA
Parity (even/odd)	PE	PO
Carry (yes/no)	CY	NC

Figure 7. Alphabetic Flag Settings

To display all flags, enter:

```
R F
```

If all the flags are in a *set* condition, the response is:

```
OV DN EI NG ZR AC PE CY -_
```

Now you can take one of two actions:

1. Press Enter to leave the settings unchanged.

2. Change any or all of the settings.

To change a flag, just enter its opposite code. The opposite codes can be entered in any order - with or without intervening spaces. For example, to change the first, third, fifth, and seventh flags, enter:

```
OV DN EI NG ZR AC PE CY - PONZDINV
```

They are entered in reverse order in this example.

Press Enter and the flags are modified as specified, the prompt appears, and you can enter the next command.

If you want to see if the new codes are in effect, enter:

```
R F
```

The response will be:

```
NV DN DI NG NZ AC PO CY - _
```

The first, third, fifth, and seventh flags are changed as requested. The second, fourth, sixth, and eighth flags are unchanged.

> **Note:** A single flag can be changed only once per R F command.

Search Command

Purpose: Searches the *range* for the character(s) in the *list*.

Format: S *range list*

Remarks: All matches are indicated by displaying the addresses where matches are found.

A display of the prompt (-) without an address means that no match was found.

> **Note:** If you enter only an offset for the starting address of the range, the S command assumes the segment contained in the DS register.

Search Command

Example: If you want to search the range of addresses from CS:100 through CS:110 for hex 41, enter:

```
S CS:100 110 41
```

If two matches are found the response might be:

```
04BA:0104
04BA:010D
```

If you want to search the same range of addresses as in the previous example for a match with the 4-byte-long list, enter:

```
S CS:100 L 11 41 "AB" E
```

The starting addresses of all matches are listed. If no match is found, no address is displayed.

Trace Command

Purpose: Executes one or more instructions starting with the instruction at CS:IP, or at =*address* if it is specified. The = must be entered. One instruction is assumed, but you can specify more than one with *value*. Displays the contents of all registers and flags *after each* instruction executes. For a description of the display format, refer to the Register command.

Format: T [=*address*] [*value*]

Remarks: The display caused by the Trace command continues until *value* instructions are executed. Therefore, when tracing multiple instructions, remember you can suspend the scrolling at any time by pressing Ctrl-NumLock. Resume scrolling by entering any other character.

Trace Command

Example: **T**

If the IP register contains 011A, and that location contains B40E (MOV AH,0EH), this might be displayed:

```
AX=0E00  BX=00FF  CX=0007  DX=01FF
SP=039D  BP=0000  SI=005C  DI=0000
DS=04BA  ES=04BA  SS=04BA  CS=04BA
IP=011C NV UP DI NG NZ AC PE NC
04BA:011C CD21      INT   21
```

This displays the results *after* the instruction at 011A is executed, and indicates the next instruction to be executed is the INT 21 at location 04BA:011C.

```
T 10
```

Sixteen instructions are executed (starting at CS:IP). The contents of all registers and flags are displayed after each instruction. The display stops after the 16th instruction has been executed. Displays may scroll off the screen unless you suspend the display by pressing the Ctrl-NumLock keys.

Unassemble Command

Purpose: Unassembles instructions (translates the contents of memory into assembler-like statements) and displays their addresses and hexadecimal values, together with assembler-like statements. For example, a display might look like this:

```
04BA:0100 206472  AND [SI+72],AH
04BA:0103 FC      CLD
04BA:0104 7665    JBE 016B
```

Format: U [*address*]

or

U [*range*]

Unassemble Command

Remarks: The number of bytes unassembled depends on your system display format (whether 40 or 80 columns), and which option you use with the Unassemble command.

Notes:

1. In all cases, the number of bytes unassembled and displayed may be slightly more than either the amount requested or the default amount. This happens because the instructions are of variable lengths; therefore, to unassemble the last instruction may include more bytes than expected.

2. Make sure that the address parameters refer to locations containing valid 8088 instruction codes. If you specify an address that does not contain the first byte of a valid instruction, the display will be erroneous.

3. If you enter only an offset for the starting address, the U command assumes the segment contained in the CS register.

Unassemble Command

The Unassemble command has two format options:

Option 1

Use this option to either unassemble instructions without specifying an address, or to unassemble instructions beginning with a specified address. For example:

U

or

U *address*

Sixteen bytes are unassembled with a 40-column display. "Thirty-two" or "display; 32" bytes are unassembled with an 80-column display.

Instructions are unassembled beginning with the specified address.

If you do not specify an address, the U command assumes the starting address is the location following the last instruction unassembled by a previous U command. Thus, it is possible to unassemble consecutive locations, producing continuous unassembled displays, by entering consecutive U commands without parameters.

Unassemble Command

If no previous U command is entered, the location is offset hex 0100 into the segment originally initialized in the segment registers by DEBUG.

Option 2

Use this option to unassemble instructions in a specified address range. For example:

U *range*

All instructions in the specified address range are unassembled, regardless of the system display format.

> **Note:** If you specify an ending address, enter it with only an offset value.

For example:

```
U 04ba:0100 108
```

The display response might be:

```
04BA:0100  206472  AND [SI+72],AH
04BA:0103  FC      CLD
04BA:0104  7665    JBE 016B
04BA:0106  207370  AND [BP+DI+70],DH
```

The same display appears if you enter:

```
U 04BA:100 L 7
```

or

```
U 04BA:100 L 8
```

or

```
U 04BA:100 L 9
```

Write Command

Purpose: Writes the data being debugged to diskette.

Format: W [*address* [*drive sector sector*]]

Remarks: The maximum number of sectors that can be written with a single Write command is hex 80.

DEBUG displays a message if a diskette write error occurs. You can retry the write operation by pressing F3 to redisplay the Write command, then press the Enter key.

The Write command has two format options:

Option 1

Use this option to write data to diskette beginning at a specified address. For example:

W *address drive sector sector*

Write Command

The data beginning at the specified address is written to the diskette in the indicated drive. The data is written starting at the specified starting relative sector (first sector) and continues until the requested number of sectors are filled (second sector).

Notes:

1. Be extremely careful when you write data to absolute sectors because an erroneous sector specification will destroy whatever was on the diskette at that location.

2. If only an offset is entered for the beginning address, the W command assumes the segment contained in the CS register.

3. Remember, the starting sector and the sector count are both specified in *hexadecimal.*

For example:

```
W 1FD 1 100 A
```

The data beginning at CS:01FD is written to the diskette in drive B, starting at relative sector hex 100 (256) and continuing for hex 0A (10) sectors.

Option 2

This option allows you to use the Write command without specifying parameters or only specifying the address parameter. For example:

W

or

W *address*

When issued without parameters (or when issued with only the address parameter), the Write command writes the file (whose filespec is properly formatted in the file control block at CS:5C) to diskette.

This condition is met by specifying the filespec when starting the DEBUG program, or by using the Name command.

> **Note:** If DEBUG was started with a filespec and subsequent Name commands were used, you may need to enter a new Name command for the proper filespec before issuing the Write command.

In addition, the BX and CX registers must be set to the number of bytes to be written. They may have been set properly by the DEBUG or Load commands, but might have been changed by a Go or Trace command. You must be certain the BX and CX registers contain the correct values.

The file beginning at CS:100, or at the location specified by *address*, is written to the diskette in the drive specified in filespec or the default drive if none was specified.

The debugged file is written over the original file that was loaded into memory, or into a new file if the filename in the FCB didn't previously exist.

> **Note:** An error message is issued if you try to write a file with an extension of .EXE or .HEX. These files must be written in a specific format that DEBUG cannot support.

Write Command

If you find it necessary to modify a file with an extension of .EXE or .HEX, and the exact locations to be modified are known, use the following procedure:

1. RENAME the file to an extension other than .EXE or .HEX.

2. Load the file into memory using the DEBUG or Load command.

3. Modify the file as needed in memory, but do not try to execute it with the Go or Trace commands. Unpredictable results would occur.

4. Write the file back using the Write command.

5. RENAME the file back to its correct name.

Summary of DEBUG Commands

The following chart is provided for quick reference.

The section called "Format Notation" in Chapter 6 explains the notation used in the format of the following commands.

Command	Purpose	Format
Assemble	Assembles statements	A [*address*]
Compare	Compares memory	C *range address*
Dump	Displays memory	D [*address*] or D [*range*]
Enter	Changes memory	E *address* [*list*]
Fill	Changes memory blocks	F *range list*
Go	Executes with optional breakpoints	G [=*address*] [*address* [*address...*]]
Hexarithmetic	Hexadecimal add-subtract	H *value value*

Figure 8 (Part 1 of 2). DEBUG Commands

Command	Purpose	Format
Input	Reads/displays input byte	I *portaddress*
Load	Loads file or absolute diskette sectors	L [*address* [*drive sector sector*]]
Move	Moves memory block	M *range address*
Name	Defines files and parameters	N *filespec* [*filespec...*]
Output	Sends output byte	O *portaddress byte*
Quit	Ends DEBUG program	Q
Register	Displays registers/flags	R [*registername*]
Search	Searches for characters	S *range list*
Trace	Executes and displays	T [=*address*] [*value*]
Unassemble	Unassembles instructions	U [*address*] or U [*range*]
Write	Writes file or absolute diskette sectors	W [*address* [*drive sector sector*]]

Figure 8 (Part 2 of 2). DEBUG Commands

Chapter 13. Using Extended Screen and Keyboard Control

Contents

Notes:

Introduction

With DOS Version 2.00 you can issue special character sequences from within your program that can be used to control screen cursor positioning. You can also reassign the meaning of any key in the keyboard.

Notes:

1. The control sequences defined below are valid only when issued through DOS function calls 1, 2, 6, and 9, and require the presence of the extended screen and keyboard control device driver. This can be accomplished by placing the command:

   DEVICE=ANSI.SYS

   in your CONFIG.SYS (configuration) file. Note that the size of DOS in memory will be increased by the size of the ANSI.SYS program.

2. The default value is used when no explicit value, or a value of zero, is specified.

3. # - Numeric Parameter. A decimal number specified with ASCII characters.

4. In the control sequences described below, ESC is the 1 byte code for ESC (hex 1B), *not* the three characters "ESC." For example, ESC [2;10H could be created under DEBUG as follows:

   e200 1B "[2;10H"

Cursor Control

Cursor Position

CUP	Function
ESC [#;#H	Moves the cursor to the position specified by the parameters. The first parameter specifies the line number and the second parameter specifies the column number. The default value is one. If no parameter is given, the cursor is moved to the home position.

Cursor Up

CUU	Function
ESC [#A	Moves the cursor up one line without changing columns. The value of # determines the number of lines moved. The default value for # is one. This sequence is ignored if the cursor is already on the top line.

Cursor Down

CUD	Function
ESC [# B	Moves the cursor down one line without changing columns. The value of # determines the number of lines moved. The default value for # is one. The sequence is ignored if the cursor is already on the bottom line.

Cursor Forward

CUF	Function
ESC [# C	Moves the cursor forward one column without changing lines. The value of # determines the number of columns moved. The default value for # is one. This sequence is ignored if the cursor is already in the rightmost column.

Cursor Backward

CUB	Function
ESC [# D	Moves the cursor back one column without changing lines. The value of # determines the number of columns moved. The default value for # is one. This sequence is ignored if the cursor is already in the leftmost column.

Horizontal and Vertical Position

HVP	Function
ESC [#;#f	Moves the cursor to the position specified by the parameters. The first parameter specifies the line number and the second parameter specifies the column number. The default value is one. If no parameter is given, the cursor is moved to the home position (same as CUP).

Device Status Report

DSR	Function
ESC [6n	The console driver will output a CPR sequence on receipt of DSR (see below).

Cursor Position Report

CPR	Function
ESC [#;#R	The CPR sequence reports the current cursor position through the standard input device. The first parameter specifies the current line and the second parameter specifies the current column.

Save Cursor Position

SCP	Function
ESC [s	The current cursor position is saved. This cursor position can be restored with the RCP sequence.

Restore Cursor Position

RCP	Function
ESC [u	Restores the cursor to the value it had when the console driver received the SCP sequence.

Erasing

Erase in Display

ED	Function
ESC [2J	Erases all of the screen and the cursor goes to the home position.

Erase in Line

EL	Function
ESC [k	Erases from the cursor to the end of the line and includes the cursor position.

Set Graphics Rendition

SGR	Function
ESC [#;...;#m	Sets the character attribute specified by the parameter(s). All following characters will have the attribute according to the parameter(s) until the next occurrence of SGR.

Parameter	Meaning
0	All attributes Off (normal white on black)
1	Bold On (high intensity)
4	Underscore On (IBM Monochrome Display only)
5	Blink On
7	Reverse video On
8	Cancelled On (invisible)
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

Set Mode

SM	Function
ESC [=#h or **ESC [=h** or **ESC [=0h** or **ESC [?7h**	Invokes the screen width or type specified by the parameter. Parameter — Meaning 0 — 40x25 black and white 1 — 40x25 color 2 — 80x25 black and white 3 — 80x25 color 4 — 320x200 color 5 — 320x200 black and white 6 — 640x200 black and white 7 — wrap at end of line (typing past end-of-line results in new line)

Reset Mode

RM	Function
ESC [=#l or **ESC [=l** or **ESC [=0l** or **ESC [?7l**	Parameters are the same as SM (Set Mode) except that parameter 7 will reset wrap at end-of-line mode (characters past end-of-line are thrown away).

Note: The character **l** is a lowercase L.

Keyboard Key Reassignment

The control sequence is	Function
ESC [#;#;...#p or **ESC ["string";p** or **ESC [#;"string";#;** **#;"string";#p** or any other combination of strings and decimal numbers	The first ASCII code in the control sequence defines which code is being mapped. The remaining numbers define the sequence of ASCII codes generated when this key is intercepted. However, if the first code in the sequence is zero (NUL) then the first and second code make up an extended ASCII re-definition (see the Technical Reference manual for a list of all the ASCII and extended ASCII codes).

Here are some examples:

1. Reassign the Q and q key to the A and a key (and the other way as well):

   ESC [65;81p A becomes Q
   ESC [97;113p a becomes q
   ESC [81;65p Q becomes A
   ESC [113;97p q becomes a

2. Reassign the F10 key to a DIR command followed by a carriage return:

   ESC [0;68;"dir";13p

   The 0;68 is the extended ASCII code for the F10 key. 13 decimal is a carriage return.

Notes:

Chapter 14. Installable Device Drivers

Contents

Introduction

The DOS Version 2.00 device interface links the devices together in a chain. This allows new device drivers for optional devices to be added to DOS.

Device Driver Format

A device driver is a .COM file with all of the code in it to implement the device. In addition it has a special header at the front of it that identifies it as a device, defines the strategy and interrupt entry points, and defines various attributes of the device.

> **Note:** For device drivers, the .COM file must not use the ORG 100H. Because it does not use the program segment prefix, the device driver is simply loaded; therefore, the .COM file must have an orgin of zero (ORG 0 or no ORG statement).

Types of Devices

There are two basic types of devices:

- Character devices
- Block devices

Character Devices

These are devices that are designed to do character I/O in a serial manner like CON, AUX, and PRN. These devices have names like CON, AUX, CLOCK$, and you can open channels (handles or FCBs) to do input and output to them.

> Note: Because character devices have only one name, they can support only one device.

Block Devices

These devices are the "fixed disk or diskette drives" on the system, they can do random I/O in pieces called blocks (usually the physical sector size of the disk). These devices are not *named* as the character devices are, and cannot be *opened* directly. Instead they are *mapped* via the drive letters (A, B, C, etc.). Block devices can have units within them. In this way, a single block driver can be responsible for one or more disk or diskette drives. For example, block device driver ALPHA can be responsible for drives A, B, C and D. This means that ALPHA has four units defined and therefore takes up four drive letters. The way the drive units and drive letters correspond is determined by the position of the driver in the chain of all drivers. For example, if device driver ALPHA is the first block driver in the device chain, and it has defined four units, then those units will be A, B, C and D. If BETA is the second block driver, and it defines three units, then those units will be E, F and G. DOS Version 2.00 is not limited to 16 block device units as previous versions were. The new limit is 63, but drives are assigned alphabetically through the collating sequence, so after drive Z, the drive "characters" get a little strange (like <,\ , >).

Device Header

A device header is required at the beginning of a device driver. Here is what the Device Header looks like:

Description	Definition
Pointer to next device header	DWORD
Attribute	WORD
Pointer to device strategy	WORD
Pointer to device interrupt	WORD
Name/unit field	8 BYTES

Next Device Header Field

The pointer to the next device header field is a double word field (offset followed by segment) that is set by DOS at the time the device driver is loaded. However, it is important that this field be set to -1 prior to load time (when it is on the disk as a .COM file) unless there is more than one device driver in the .COM file. If there is more than one driver in the file, the first word of the double word pointer should be the offset of the next driver's Device Header.

> **Note:** If there is more than one device driver in the .COM file, the *last* driver in the file must have the pointer to next Device Header field set to -1.

Attribute Field

The next field in the header describes to the system the attributes of the device. They are as follows:

bit 15 = 1 if character device
0 if block device
bit 14 = 1 if IOCTL is supported
0 if it is not
bit 13 = 1 if non IBM format (block only)
0 if IBM format
bit 3 = 1 if current clock device
0 if it is not
bit 2 = 1 if current NUL device
0 if it is not
bit 1 = 1 if current standard output device
0 if it is not
bit 0 = 1 if current standard input device
0 if it is not

All other bits must be off.

The most important bit is bit 15, which tells the system that it is a block or a character device. With the exception of bits 13 and 14, the rest are for giving character devices special treatment and mean nothing on a block device. These *special treatment* bits allow you to tell DOS that your new device driver is the new standard input device and standard output device (the CON device). This can be done by setting bits 0 and 1 to 1. Similarly, a new CLOCK$ device could be installed by setting that attribute bit.

Although there is a NUL device attribute bit, the NUL device *cannot be reassigned.* This is an attribute that exists for DOS so it can tell if the NUL device is being used. The non IBM format bit applies only to block devices and affects the operation of the Get BPB (BIOS Parameter Block) device call (covered later in this chapter). The other bit of interest is the IOCTL bit. This is used for both block and character devices, and tells DOS whether the device is able to handle control strings (through the IOCTL system call).

If a driver cannot process control strings, it should initially set this bit to 0. This way DOS can return an error if an attempt is made through the IOCTL system call to send or receive control strings to the device. A device that is able to process such control strings should initialize this bit to 1. For devices of this type, DOS will make the calls to the IOCTL input and the IOCTL output device functions to send and receive IOCTL strings.

The IOCTL functions allow data to be sent to and from the device without actually doing a normal read or write. In this way, the device can use the data for its own use (like setting a baud rate, stop bits, changing form lengths, etc.). It is up to the device to interpret the information passed to it, but it must not be treated as a normal I/O request.

Strategy and Interrupt Routines

These two fields are the pointers to the entry points of the strategy and interrupt routines. They are word values, so they must be in the same segment as the Device Header.

Name Field

This is an 8-byte field that contains the name of a character device, or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional, because DOS will fill in this location with the value returned by the driver's INIT code. (Refer to "Installation of Device Drivers" in this chapter.)

Creating a Device Driver

In order to create a device driver that DOS can install, a .COM file must be created with the Device Header at the start of the file. Remember that for device drivers, the code should not be originated at 100H, but rather at 0. The link field (pointer to next Device Header) should be −1 unless there is more than one device driver in the .COM file. The attribute field and entry points must be set correctly.

If it is a character device, the name field should be filled in with the name of that character device. The name can be any legal 8-character filename.

DOS always processes installable device drivers before handling the default devices, so to install a new CON device, simply name the device CON (just be sure to set the standard input device and standard output device bits in the attribute word on a new CON device). The scan of the device list stops on the first match, so the installable device driver takes precedence.

> Note: Because DOS can install the driver anywhere in memory, care must be taken in any far memory references. You should not expect that your driver will always be loaded at the same place every time.

Installation of Device Drivers

DOS Version 2.00 allows new device drivers to be installed dynamically at boot time by reading and processing the device options in the CONFIG.SYS file.

DOS calls a device driver at it's strategy entry point first, passing in a Request Header the information describing what DOS wants the device driver to do.

The strategy routine does not perform the request, but rather it enqueues the request (saves a pointer to the Request Header). The second entry point is the interrupt routine, and is called by DOS immediately after the strategy routine returns. The "interrupt" routine is called with no parameters. Its function is to perform the operation based on the queued request and set up any return information.

DOS passes the pointer to the Request Header in ES:BX. This structure consists of a fixed length header (Request Header) followed by data pertinent to the operation to be performed.

> **Note:** It is the responsibility of the device driver to preserve the machine state (for example, save all registers on entry, and restore them on exit).
>
> The stack used by DOS will have enough room on it to save all of the registers. If more stack space is needed, it is the device drivers responsibility to allocate and maintain another stack.
>
> All calls to device drivers are FAR calls, and FAR returns should be executed to return to DOS. (See "Sample Device Driver" listing at the end of this chapter.)

Request Header

BYTE length in bytes of the Request Header plus any data at the end of the Request Header
BYTE unit code The subunit the operation is for (minor device). Has no meaning for character devices.
BYTE command code
WORD Status
8 BYTE area reserved for DOS
Data appropriate to the operation

Unit Code

The unit code field identifies which unit in your device driver the request is for. For example, if your device driver has 3 units defined, then the possible values of the unit code field would be 0, 1, and 2.

Command Code

The command code field in the Request Header can have the following values:

Code	Function
0	INIT
1	MEDIA Check (Block only, NOP for character)
2	BUILD BPB (Block only, NOP for character)
3	IOCTL input (only called if IOCTL bit is 1)
4	INPUT (read)
5	NON-DESTRUCTIVE INPUT NO WAIT (Character devices only)
6	INPUT STATUS (Character devices only)
7	INPUT FLUSH (Character devices only)
8	OUTPUT (write)
9	OUTPUT (write) with verify
10	OUTPUT STATUS (Character devices only)
11	OUTPUT FLUSH (Character devices only)
12	IOCTL output (only called if IOCTL bit is 1)

BUILD BPB and MEDIA CHECK

BUILD BPB and MEDIA CHECK, for block devices only, are explained here.

DOS calls MEDIA CHECK first for a drive unit. DOS passes it's current Media Descriptor byte (see "Media Descriptor Byte" later in this chapter). MEDIA CHECK returns one of the following four results:

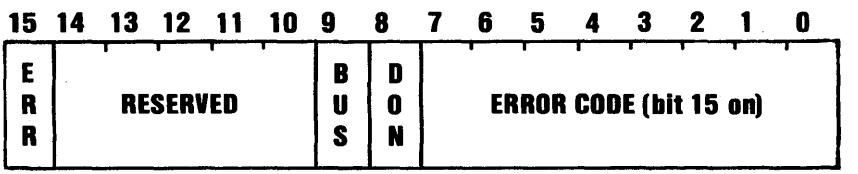
- Media Not Changed
- Media Changed
- Not Sure

DOS will call BUILD BPB under the following two conditions:

- If "Media Changed" is returned
- If "Not Sure" is returned and there are no dirty buffers (buffers with changed data, not yet written to disk).

Status Word

The status word in the Request Header.



The status word is zero on entry and is set by the driver interrupt routine on return.

Bit 8 is the done bit. When set it means the operation is complete. For DOS 2.00 the Driver just sets it to one when it exits.

Bit 15 is the error bit. If it is set, then the low 8 bits of the status word indicate the error. The errors are:

00 Write Protect Violation
01 Unknown Unit
02 Device Not Ready
03 Unknown command
04 CRC Error
05 Bad Drive Request Structure Length
06 Seek Error
07 Unknown Media
08 Sector Not Found
09 Printer Out of Paper
0A Write Fault
0B Read Fault
0C General Failure

Bit 9 is the busy bit that is set by status calls.

For output on character devices: If it is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request, and a write request (if made) would start immediately.

For input on character devices with a buffer: If it is 1 on return, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device buffer and a read would return quickly, it also indicates that the user has typed something. DOS assumes all character devices have an input *type ahead* buffer. Devices that do not have them should always return busy = 0 so that DOS will not continuously wait for something to get into a buffer that does not exist.

One of the functions defined for each device is INIT. This routine is called only once when the device is installed and never again. There are several things returned by the INIT routine. First, there is a location of the first free byte of memory after the device driver (like a terminate and stay resident) that is stored in the ending address field. In this manner, initialization code can be used once and thrown away in order to save space.

After sending the ending address field, a character device driver can set the status word and return. While block devices are installed in the same way as character devices, they must return additional information. The number of units for the device driver is returned, and this determines the logical names that the devices will have. For example, if the current maximum logical device letter is F at the time of the install call, and the block device driver INIT routine returns 3 units, then their logical names will be G, H, and I. This mapping is determined by the position of the driver in the device list, and the number of units on the device. The number of units returned by INIT will override the value in the name/unit field of the Device Header.

In addition, a pointer to a BPB (BIOS Parameter Block) pointer array is also returned. This is a pointer to an array of *n* word pointers, where *n* is the number of units defined. These word pointers point to BPBs. In this way, if all of the units are the same, the entire array can point to the same BPB in order to save space.

> **Note:** This array must be protected (below the free pointer set by the return).

The BPB (BIOS Parameter Block) contains information pertinent to the devices like sector size, sectors per allocation unit, etc. The sector size in the BPB cannot be greater than the maximum allowed (set at DOS initialization time).

The last thing that INIT of a block device must pass back is the "media descriptor byte". This byte means nothing to DOS, but is passed to devices so that they know what parameters DOS is currently using for a particular Drive-Unit.

Block devices may take several approaches; they may be *dumb* or *smart*. A dumb device would define a unit (and therefore a BPB) for each possible media drive combination. Unit 0 = drive 0 single side, unit 1 = drive 0 double side, etc. For this approach, media descriptor bytes would mean nothing. A smart device would allow multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media supported (sector size in BPB must be as large as maximum sector size that DOS is currently using). Smart drivers will use the "media byte" to pass information about what media is currently in a unit.

Function Call Parameters

All strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue the strategy routines store them in. The command code in the Request Header tells the driver which function to perform.

> Note: All DWORD pointers are stored offset first, then segment.

INIT

Command code=0

ES:BX

13-BYTE Request Header
BYTE number of units (not set by character devices)
DWORD Ending Address
DWORD Pointer to BPB array (not set by character devices)

The driver must do the following:

- Set the number of units (block devices only).
- Set up the pointer to the BPB array (block devices only).
- Perform any initialization code (to modems, printers, etc.).
- Set up the ending address for resident code.
- Set the status word in the Request Header.

  Note: If there are multiple device drivers in a single .COM file, the ending address returned by the last INIT called will be the one DOS uses. For the sake of simplicity, it is recommended that all of the device drivers in a single .COM file return the same ending address.

MEDIA CHECK

Command code=1

ES:BX

13-BYTE Request Header
BYTE Media Descriptor from DOS
BYTE return information

The driver must perform the following:

- Set the return byte:
  - -1 Media has been changed
  - 0 Don't know if media has been changed
  - 1 Media has not been changed

- Set the status word in the Request Header.

BUILD BPB (BIOS Parameter Block)

Command code=2

ES:BX

13-BYTE Request Header
BYTE Media Descriptor from DOS
DWORD Transfer Address (buffer address)
DWORD Pointer to BPB table

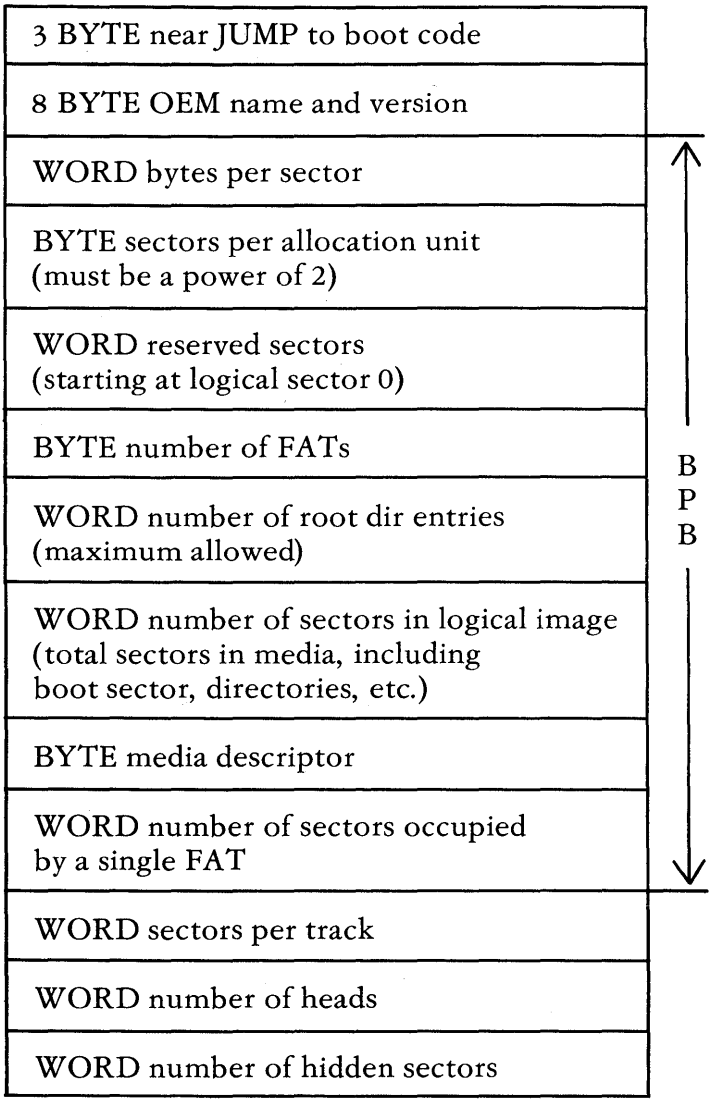
The driver must perform the following:

- Set the pointer to the BPB.
- Set the status word in the Request Header.

The driver must determine the correct media that is currently in the unit to return the pointer to the BPB table. The way the buffer is used (pointer passed by DOS) is determined by the non-IBM format bit in the attribute field of the device header. If the bit is zero (device is IBM format compatible) then the buffer contains the first sector of the FAT (most importantly the FAT id byte). The driver must not alter this buffer in this case. If the bit is a one, then the buffer is a one sector scratch area that can be used for anything.

If the device is IBM format compatible, then it must be true that the first sector of the first FAT is located at the same sector for all possible media. This is because the FAT sector is read *before* the media is actually determined.

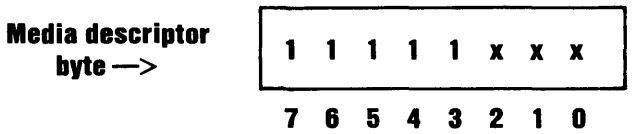
The information relating to the BPB for a particular media is kept in the boot sector for the media. In particular, the format of the boot sector is:



The three words at the end are optional. DOS does not care about them because they are not part of the BPB. They are intended to help the device driver understand the media. Sectors per track may be redundant because it can be calculated from the total size of the disk. The number of heads is useful for supporting different multi-head drives that have the same storage capacity but a different number of surfaces. The number of hidden sectors is useful for supporting drive partitioning schemes.

MEDIA Descriptor Byte

Currently the media descriptor byte has been defined for a few media types:



Bit	Meaning	
0	1=2 sided	0=not 2 sided
1	1=8 sector	0=not 8 sector
2	1=removable	0=not removable
3-7	must be set to 1	

Examples of current DOS media descriptor bytes:

- 5 1/4" Diskettes:

  hex FC 1 sided 9 sector
  hex FD 2 sided 9 sector
  hex FE 1 sided 8 sector
  hex FF 2 sided 8 sector

- Fixed Disks:

  hex F8 (Fixed disk)

- 8" Diskettes:

  Hex FE (IBM 3740 Format). Single sided, single density, 128 bytes per sector, soft sectored, 4 sectors per allocation unit, 1 reserved sector, 2 FATs, 68 directory entries, 77*26 sectors.

  Hex FD (IBM 3740 Format). Dual sided, single density, 128 bytes per sector, soft sectored, 4 sectors per allocation unit, 4 reserved sectors, 2 FATs, 68 directory entries, 77*26 sectors.

  Hex FE. Single sided, double density, 1024 bytes per sector, soft sectored, 1 sector per allocation unit, 1 reserved sector, 2 FATs, 192 directory entries, 77*8*2 sectors.

  Note: The two MEDIA descriptor bytes that are the same for 8" diskettes (hex FE) is not a misprint. To establish whether a diskette is single density or double density, a read of a single density address mark should be made. If an error occurs, the media is double density.

INPUT or OUTPUT

Command codes=3,4,8,9, and 12

ES:BX

13-BYTE Request Header
BYTE Media descriptor byte
DWORD transfer address (buffer address)
WORD byte/sector Count
WORD starting sector number (no meaning on character devices)

The driver must perform the following:

- Do the requested function.
- Set the actual number of sectors (bytes) transferred.
- Set the status word in the Request Header.

Note: No error checking is performed on an IOCTL call. However, the driver must set the return sector (byte) count to the correct number transferred.

The following applies to block device drivers:

Under certain circumstances the device driver may be asked to do a write operation of 64K bytes that seems to be a *wrap around* of the transfer address in the device driver request packet. This arises due to an optimization added to the write code in DOS. It will only happen on WRITEs that are within a sector size of 64K bytes on files that are being extended past the current end of file. It is allowable for the device driver to ignore the balance of the WRITE that wraps around, if it so choses. For example, a WRITE of 10000H bytes worth of sectors with a transfer address of xxxx:1 could ignore the last two bytes.

Remember: A program that uses DOS function calls can never request an input or output operation of more than FFFFH bytes; therefore, a wrap around in the transfer (buffer) segment cannot occur. It is for this reason that you can ignore bytes that would have wrapped around in the transfer segment.

Non Destructive Input No Wait

Command code=5

ES:BX

13-BYTE Request Header
BYTE read from device

The driver must perform the following:

- Return a byte from the device.
- Set the status word in the Request Header.

This call is analagous to the console input status call on previous versions of DOS. If the character device returns busy bit = 0 (characters in buffer), then the next character that would be read is returned. This character is not removed from the input buffer (hence the term Non Destructive Input). This call allows DOS to look ahead one input character.

STATUS

Command codes=6 and 10

ES:BX

13-BYTE Request Header

All the driver must do is perform the operation and set the status word in the Request Header accordingly.

FLUSH

Command codes=7 and 11

ES:BX

13-BYTE Request Header

This call tells the driver to flush (terminate) all pending requests that it has knowledge of. Its primary use is to flush the input queue on character devices. The driver must set status word in the Request Header upon return.

The CLOCK$ Device

A popular add on feature is a "Real Time Clock" board. To allow these boards to be integrated into the system for TIME and DATE, there is a special device (determined by the attribute word) which is the CLOCK$ device. In all respects, this device defines and performs functions like any other character device (most functions will be set done bit, reset error bit, return). When a read or write to this device occurs, exactly 6 bytes are transferred. The first two bytes are a word which is the count of days since 1-1-80. The third byte is minutes, the fourth hours, the fifth 1/100 seconds, and the sixth seconds. Reading the CLOCK$ device gets the date and time, writing to it sets the date and time.

Sample Device Driver

```
                        ; ******************************************************
                        ; *                    PROLOG                          *
                        ; *  THIS IS AN INSTALLABLE DEVICE DRIVER FOR AN       *
                        ; *  IN STORAGE DISKETTE (VIRTUAL) WITH 180K CAPACITY. *
                        ; ******************************************************
0000                    CSEG    SEGMENT PARA PUBLIC 'CODE'
                        ;
                        ;         M A C R O ( S )
                        ;
                        STATUS  MACRO   STATE,ERR,RC
                                IFIDN   <STATE>,<DONE>
                                 OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
                                ENDIF
                                IFIDN   <STATE>,<BUSY>
                                 OR      ES:WORD PTR SRH_STA_FLD[BX],0200H
                                ENDIF
                                IFIDN   <ERR>,<ERROR>
                                 OR      ES:WORD PTR SRH_STA_FLD[BX],1000H
                                ENDIF
                                IFNB    <RC>
                                 OR      ES:WORD PTR SRH_STA_FLD[BX],RC
                                ENDIF
                                ENDM
                        ;
                        ;         E Q U A T E S
                        ;
                        ; READ/WRITE
                        ;
= 0000                  SRH         EQU     0               ;STATIC REQUEST HEADER START
= 000D                  SRH_LEN     EQU     13              ;  "      "       "    LENGTH
=                       SRH_LEN_FLD EQU     SRH             ;  "      "       "      "    FIELD
= 0001                  SRH_UCD_FLD EQU     SRH+1           ;  "      "       "    UNIT CODE FIELD
= 0002                  SRH_CCD_FLD EQU     SRH+2           ;  "      "       "    COMMAND CODE FIELD
= 0003                  SRH_STA_FLD EQU     SRH+3           ;  "      "       "    STATUS FIELD
= 0005                  SRH_RES_FLD EQU     SRH+5           ;  "      "       "    RESERVED AREA FIELD
                        ;
= 000D                  MD          EQU     SRH+SRH_LEN     ;MEDIA DESCRIPTOR BYTE
= 0001                  MD_LEN      EQU     1               ;  "      "       "   LENGTH
= 000E                  DTA         EQU     MD+MD_LEN       ;DISK TRANSFER ADDRESS
= 0004                  DTA_LEN     EQU     4               ; DTA LENGTH
= 0012                  COUNT       EQU     DTA+DTA_LEN     ;BYTE/SECTOR COUNT
= 0002                  COUNT_LEN   EQU     2               ;    "         "   LENGTH
= 0014                  SSN         EQU     COUNT+COUNT_LEN ;STARTING SECTOR NUMBER
= 0002                  SSN_LEN     EQU     2               ;  "      "       "   LENGTH
                        ;
                        ; MEDIA CHECK
                        ;
= 000E                  RET_BYTE    EQU     MD+MD_LEN       ;BYTE RETURNED FROM DRIVER
                        ;
                        ; BUILD BPB
                        ;
= 0012                  BPBA_PTR    EQU     DTA+DTA_LEN     ;POINTER TO BPB
= 0004                  BPBA_PTR_LEN EQU    4               ;  "    "  "  LENGTH
                        ;
                        ; INIT
```

```
                                ;
= 000D                          UNITS       EQU     SRH+SRH_LEN
= 0001                          UNITS_LEN   EQU     1
= 000E                          BR_ADDR_0   EQU     UNITS+UNITS_LEN
= 0010                          BR_ADDR_1   EQU     BR_ADDR_0+2
= 0004                          BR_ADDR_LEN EQU     4
= 0012                          BPB_PTR_OFF EQU     BR_ADDR_0+BR_ADDR_LEN
= 0014                          BPB_PTR_SEG EQU     BPB_PTR_OFF+2
                                ;
                                ;
0000                            VDSK    PROC    FAR
                                        ASSUME  CS:CSEG,ES:CSEG,DS:CSEG
0000                            BEGIN:
= 0000                          START       EQU     $
                                ;       S P E C I A L   D E V I C E   H E A D E R
0000  FF FF FF FF               NEXT_DEV    DD      -1              ;POINTER TO NEXT DEVICE
0004  2000                      ATTRIBUTE   DW      2000H           ;BLOCK DEVICE (NON-IBM FORMAT)
0006  00E1 R                    STRATEGY    DW      DEV_STRATEGY    ;POINTER TO DEVICE STRATEGY
0008  00EC R                    INTERRUPT   DW      DEV_INT         ;POINTER TO DEVICE INTERRUPT HANDLER
000A  01                        DEV_NAME    DB      1               ;NUMBER OF BLOCK DEVICES
000B     07 [                               DB      7 DUP(?)        ;7 BYTES OF FILLER
               ??
                  ]
                                ;
0012  ????                      RH_OFF      DW      ?               ;REQUEST HEADER OFFSET
0014  ????                      RH_SEG      DW      ?               ;REQUEST HEADER SEGMENT
                                ; BIOS PARAMETER BLOCK
= 0016                          BPB     EQU     $
0016  0200                              DW      512                 ;SECTOR SIZE
0018  01                                DB      1                   ;SECTORS/ALLOCATION UNIT
0019  0001                              DW      1                   ;NUMBER OF RESERVED SECTORS
001B  02                                DB      2                   ;NUMBER OF FATS
001C  0040                              DW      64                  ;NUMBER OF DIRECTORY ENTRIES
001E  0168                              DW      360                 ;TOTAL NUMBER OF SECTORS
0020  FC                                DB      0FCH                ;MEDIA DESCRIPTOR
0021  0002                              DW      2                   ;NUMBER OF SECTORS OCCUPIED BY FAT
                                ;
0023  0016 R                    BPB_PTR DW      BPB                 ;BIOS PARAMETER BLOCK POINTER ARRAY (1 ENTR)
                                ; CURRENT VIRTUAL DISK INFORMATION
0025  ????                      TOTAL       DW      ?               ;TOTAL SECTORS TO TRANSFER
0027  00                        VERIFY      DB      0               ;VERIFY  1=YES, 0=NO
0028  0000                      START_SEC   DW      0               ;STARTING SECTOR NUMBER
002A  0000                      VDISK_PTR   DW      0               ;STARTING SEGMENT OF VIRTRUAL DISK
002C  ????????                  USER_DTA    DD      ?               ;POINTER TO CALLERS DISK TRANSFER ADDRESS
= 0030                          BOOT_REC    EQU     $               ;DUMMY DOS BOOT RECORD
0030     03 [                               DB      3 DUP(0)        ;3 BYTE JUMP TO BOOT CODE (NOT BOOTABLE)
                  00
                     ]
                                ;
0033  49 42 4D 20 20 32                    DB      'IBM  2.0'      ;VENDOR IDENTIFICATION
      2E 30
003B  0200                              DW      512                 ;NUMBER OF BYTES IN A SECTOR
003D  01                                DB      1                   ;1 SECTOR PER ALLOCATION UNIT
003E  0001                              DW      1                   ;1 RESERVED SECTOR
```

```
0040 02                         DB      2               ;2 FATS
0041 0040                       DW      64              ;NUMBER OF DIRECTORY ENTRIES
0043 0168                       DW      360             ;360 TOTAL SECTORS IN IMAGE
0045 FC                         DB      0FCH            ;TELLS DOS THIS IS A SINGLE SIDED 9 SECTOR I
0046 0002                       DW      2               ;NUMBER OF SECTORS IN FAT
                        ;
                        ;       FUNCTION TABLE
                        ;
0048                    FUNTAB  LABEL   BYTE
0048 0105 R                     DW      INIT            ;INITIALIZATION
004A 01B8 R                     DW      MEDIA_CHECK     ;MEDIA CHECK (BLOCK ONLY)
004C 01CC R                     DW      BUILD_BPB       ;BUILD BPB      "     "
004E 0212 R                     DW      IOCTL_IN        ;IOCTL INPUT
0050 0212 R                     DW      INPUT           ;INPUT (READ)
0052 0212 R                     DW      ND_INPUT        ;NON_DESTRUCTIVE INPUT NO WAIT (CHAR ONLY)
0054 0212 R                     DW      IN_STAT         ;INPUT STATUS                   "     "
0056 0212 R                     DW      IN_FLUSH        ;INPUT FLUSH                    "     "
0058 0241 R                     DW      OUTPUT          ;OUTPUT (WRITE)
005A 0280 R                     DW      OUT_VERIFY      ;OUTPUT (WRITE) WITH VERIFY
005C 0212 R                     DW      OUT_STAT        ;OUTPUT STATUS                  "     "
005E 0212 R                     DW      OUT_FLUSH       ;OUTPUT FLUSH                   "     "
0060 0212 R                     DW      IOCTL_OUT       ;IOCTL OUTPUT
                        ;
                        ; L O C A L   P R O C E D U R E S
                        ;
0062                    IN_SAVE PROC    NEAR
0062 26: 8B 47 0E               MOV     AX,ES:WORD PTR DTA[BX]  ;SAVE CALLERS DTA
0066 2E: A3 002C R              MOV     CS:USER_DTA,AX
006A 26: 8B 47 10               MOV     AX,ES:WORD PTR DTA+2[BX]
006E 2E: A3 002E R              MOV     CS:USER_DTA+2,AX
0072 26: 8B 47 12               MOV     AX,ES:WORD PTR COUNT[BX] ;GET NUMBER OF SECTORS TO READ
0076 32 E4                      XOR     AH,AH
0078 2E: A3 0025 R              MOV     CS:TOTAL,AX             ;MOVE NUMBER OF SECTORS TO TOTAL
007C C3                         RET
007D                    IN_SAVE ENDP
                        ;
007D                    CALC_ADDR PROC NEAR
007D 2E: A1 0028 R              MOV     AX,CS:START_SEC         ;GET STARTING SECTOR NUMBER
0081 B9 0020                    MOV     CX,20H                  ;MOV 512 TO CX SEGMENT STYLE
0084 F7 E1                      MUL     CX                      ;MULTIPLY TO GET ACTUAL SECTOR
0086 2E: 8B 16 002A R           MOV     DX,CS:VDISK_PTR         ;GET SEGMENT OF VIRTUAL DISK
008B 03 D0                      ADD     DX,AX                   ;ADD THAT SEGMENT TO INITIAL SEGMENT
008D 8E DA                      MOV     DS,DX                   ;SAVE THAT AS THE ACTUAL SEGMENT
008F 33 F6                      XOR     SI,SI                   ;IT'S ON A PARAGRAPH BOUNDARY
0091 2E: A1 0025 R              MOV     AX,CS:TOTAL             ;TOTAL NUMBER OF SECTORS TO READ
0095 B9 0200                    MOV     CX,512                  ;BYTES PER SECTOR
0098 F7 E1                      MUL     CX                      ;MULTIPLY TO GET COPY LENGTH
009A 0B C0                      OR      AX,AX                   ;CHECK FOR GREATER THAN 64K
009C 75 03                      JNZ     MOVE_IT
009E B8 FFFF                    MOV     AX,0FFFFH               ;MOVE IN FOR 64K
00A1                    MOVE_IT:
00A1 91                         XCHG    CX,AX                   ;MOVE LENGTH TO CX
00A2 C3                         RET
00A3                    CALC_ADDR ENDP
                        ;
```

```
00A3                            SECTOR_READ PROC NEAR
00A3  E8 007D R                         CALL    CALC_ADDR               ;CALCULATE THE STARTING "SECTOR"
00A6  2E: 8E 06 002E R                  MOV     ES,CS:USER_DTA+2        ;SET DESTINATION (ES:DI) TO POINT
00AB  2E: 8B 3E 002C R                  MOV     DI,CS:USER_DTA          ;TO CALLERS DTA
                                        ;
                                        ; CHECK FOR DTA WRAP IN CASE WE CAME THROUGH VIA VERIFY
                                        ;
00B0  8B C7                             MOV     AX,DI                   ;GET OFFSET OF DTA
00B2  03 C1                             ADD     AX,CX                   ;ADD COPY LENGTH TO IT
00B4  73 07                             JNC     READ_COPY               ;CARRY FLAG = 0, NO WRAP
00B6  B8 FFFF                           MOV     AX,0FFFFH               ;MAX LENGTH
00B9  2B C7                             SUB     AX,DI                   ;SUBTRACT DTA OFFSET FROM MAX
00BB  8B C8                             MOV     CX,AX                   ;USE THAT AS COPY LENGTH TO AVOID WRAP
00BD                            READ_COPY:
00BD  F3/ A4                    REP     MOVSB                           ;DO THE "READ"
00BF  C3                                RET
00C0                            SECTOR_READ ENDP
                                ;
00C0                            SECTOR_WRITE PROC NEAR
00C0  E8 007D R                         CALL    CALC_ADDR               ;CALCULATE STARTING "SECTOR"
00C3  1E                                PUSH    DS
00C4  07                                POP     ES                      ;ESATABLISH ADDRESSABILITY
00C5  8B FE                             MOV     DI,SI                   ; ES:DI POINT TO "DISK"
00C7  2E: 8E 1E 002E R                  MOV     DS,CS:USER_DTA+2        ; DS:SI POINT TO CALLERS DTA
00CC  2E: 8B 36 002C R                  MOV     SI,CS:USER_DTA
                                        ;
                                        ; CHECK FOR DTA WRAP
                                        ;
00D1  8B C6                             MOV     AX,SI                   ;MOVE DTA OFFSET TO AX
00D3  03 C1                             ADD     AX,CX                   ;ADD COPY LENGTH TO OFFSET
00D5  73 07                             JNC     WRITE_COPY              ;CARRY FLAG = 0, NO SEGMENT WRAP
00D7  B8 FFFF                           MOV     AX,0FFFFH               ;MOVE IN MAX COPY LENGTH
00DA  2B C6                             SUB     AX,SI                   ;SUBTRACT DTA OFFSET FROM MAX
00DC  8B C8                             MOV     CX,AX                   ;USE AS NEW COPY LENGTH TO AVOID WRAP
00DE                            WRITE_COPY:
00DE  F3/ A4                    REP     MOVSB                           ;DO THE "WRITE"
00E0  C3                                RET
00E1                            SECTOR_WRITE ENDP
```

```
                                PAGE
                        ;
                        ; D E V I C E   S T R A T E G Y
                        ;
00E1                    DEV_STRATEGY:
00E1  2E: 8C 06 0014 R          MOV     CS:RH_SEG,ES            ;SAVE SEGMENT OF REQUEST HEADER POINTER
00E6  2E: 89 1E 0012 R          MOV     CS:RH_OFF,BX            ;SAVE OFFSET  OF    "       "       "
00EB  CB                        RET
                        ;
                        ; D E V I C E   I N T E R R U P T   H A N D L E R
                        ;
00EC                    DEV_INT:
                        ; PRESERVE MACHINE STATE ON ENTRY
                                CLD
00ED  1E                        PUSH    DS
00EE  06                        PUSH    ES
00EF  50                        PUSH    AX
00F0  53                        PUSH    BX
00F1  51                        PUSH    CX
00F2  52                        PUSH    DX
00F3  57                        PUSH    DI
00F4  56                        PUSH    SI
                        ;
                        ; DO THE BRANCH ACCORDING TO THE FUNCTION PASSED
                        ;
00F5  26: 8A 47 02              MOV     AL,ES:[BX]+2            ;GET FUNCTION BYTE
00F9  D0 C0                     ROL     AL,1                    ;GET OFFSET INTO TABLE
00FB  8D 3E 0048 R              LEA     DI,FUNTAB               ;GET ADDRESS OF FUNCTION TABLE
00FF  32 E4                     XOR     AH,AH
0101  03 F8                     ADD     DI,AX
0103  FF 25                     JMP     WORD PTR[DI]
                        ;
                        ;   INIT
                        ;
0105                    INIT:
0105  0E                        PUSH    CS
0106  5A                        POP     DX                      ;CURRENT CS TO DX
0107  2E: 8D 06 02A0 R          LEA     AX,CS:VDISK             ;GET ADDRESS OF VIRTUAL DISK
010C  B1 04                     MOV     CL,4
010E  D3 C8                     ROR     AX,CL                   ;DIVIDE BY 16 (PARAGRAPH FORM)
0110  03 D0                     ADD     DX,AX                   ;ADD TO CURRENT CS VALUE
0112  2E: 89 16 002A R          MOV     CS:VDISK_PTR,DX         ;SAVE AS STARTING SEGMENT OF VIRTUAL DISK
0117  B8 2D00                   MOV     AX,2D00H                ; ADD 2D00H PARAGRAPHS TO STARTING
011A  03 D0                     ADD     DX,AX                   ;     SEGMENT OF VIRTUAL DISK
011C  26: C7 47 0E 0000         MOV     ES:WORD PTR BR_ADDR_0[BX],0
0122  26: 89 57 10              MOV     ES:BR_ADDR_1[BX],DX     ;MAKE THAT THE BREAK ADDRESS
0126  26: C6 47 0D 01           MOV     ES:BYTE PTR UNITS[BX],1 ;NUMBER OF DISKETTE UNITS
012B  8B 16 0023 R              LEA     DX,BPB_PTR              ;GET ADDRESS OF BPB POINTER ARRAY
012F  26: 89 57 12              MOV     ES:BPB_PTR_OFF[BX],DX   ;SAVE OFFSET IN DATA PACKET
0133  26: 8C 4F 14              MOV     ES:BPB_PTR_SEG[BX],CS   ;SAVE SEGMENT IN DATA PACKET
0137  2E: 8E 06 002A R          MOV     ES,CS:VDISK_PTR         ;GET STARTING SECTOR OF VIRTUAL DISK
013C  33 FF                     XOR     DI,DI                   ;ZERO OUT DI (BOOT RECORD)
013E  8D 36 0030 R              LEA     SI,BOOT_REC             ;ADDRESS OF BOOT RECORD
0142  B9 0018                   MOV     CX,24                   ;
0145  F3/ A4            REP     MOVSB                           ;COPY 24 BYTES OF BOOT RECORD
```

```
0147  2E: C7 06 0028 R 0001          MOV     CS:WORD PTR START_SEC,1
014E  2E: C7 06 0025 R 0002          MOV     CS:WORD PTR TOTAL,2
0155  EB 007D R                      CALL    CALC_ADDR               ;CALCULATE ADDRESS OF LOGICAL SECTOR 1
0158  1E                             PUSH    DS
0159  07                             POP     ES
015A  8B FE                          MOV     DI,SI                   ;MOVE THAT ADDRESS TO ES:DI
015C  32 C0                          XOR     AL,AL
015E  F3/ AA                   REP   STOSB                           ;ZERO OUT FAT AREA
0160  C6 04 FC                      MOV     DS:BYTE PTR [SI],0FCH   ;SET THE FIRST FAT ENTRY
0163  C6 44 01 FF                   MOV     DS:BYTE PTR 1[SI],0FFH
0167  C6 44 02 FF                   MOV     DS:BYTE PTR 2[SI],0FFH
016B  1E                             PUSH    DS                      ;SAVE POINTER TO FAT
016C  56                             PUSH    SI                      ;           ON THE STACK
016D  2E: C7 06 0028 R 0003          MOV     CS:WORD PTR START_SEC,3
0174  2E: C7 06 0025 R 0002          MOV     CS:WORD PTR TOTAL,2
017B  EB 007D R                      CALL    CALC_ADDR               ;CALCULATE ADDRESS OF LOGICAL SECTOR 3
017E  1E                             PUSH    DS
017F  07                             POP     ES
0180  8B FE                          MOV     DI,SI                   ;MOVE THAT ADDRESS TO ES:DI
0182  5E                             POP     SI
0183  1F                             POP     DS                      ;RESTORE ADDRESS TO FIRST FAT
0184  F3/ A4                   REP   MOVSB                           ;COPY FIRST FAT TO SECOND FAT
0186  2E: C7 06 0028 R 0005          MOV     CS:WORD PTR START_SEC,5
018D  2E: C7 06 0025 R 0004          MOV     CS:WORD PTR TOTAL,4
0194  EB 007D R                      CALL    CALC_ADDR               ;CALCULATE ADDR OF L.S. 5 (START OF DIR)
0197  32 C0                          XOR     AL,AL
0199  1E                             PUSH    DS
019A  07                             POP     ES                      ;SET UP ES:DI TO POINT TO IT
019B  33 FF                          XOR     DI,DI
019D  F3/ AA                   REP   STOSB                           ;ZERO OUT DIRECTORY
019F  2E: 8E 06 0014 R               MOV     ES,CS:RH_SEG            ;RESTORE ES:BX TO REQUEST HEADER
01A4  2E: 8B 1E 0012 R               MOV     BX,CS:RH_OFF
                                     STATUS  DONE,NOERROR,0          ;SET STATUS WORD (DONE, NOERROR)
01A9  26: 81 4F 03 0100        +      OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
01AF  26: 81 4F 03 0000        +      OR      ES:WORD PTR SRH_STA_FLD[BX],0
01B5  E9 0288 R                      JMP     EXIT
                               ;
                               ; MEDIA CHECK
                               ;
01B8                           MEDIA_CHECK:                          ;MEDIA CHECK (BLOCK ONLY)
                               ;
                               ;  SET MEDIA NOT CHANGED
                               ;
01B8  26: C6 47 0E 01                MOV     ES:BYTE PTR RET_BYTE[BX],1 ;STORE IN RETURN BYTE
                                     STATUS  DONE,NOERROR,0          ;TURN ON THE DONE BIT
01BD  26: 81 4F 03 0100        +      OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
01C3  26: 81 4F 03 0000        +      OR      ES:WORD PTR SRH_STA_FLD[BX],0
01C9  E9 0288 R                      JMP     EXIT
                               ;
                               ; BUILD BIOS PARAMETER BLOCK
                               ;
01CC                           BUILD_BPB:
01CC  06                             PUSH    ES                      ;SAVE SRH SEGMENT
01CD  53                             PUSH    BX                      ;SAVE SRH OFFSET
01CE  2E: C7 06 0028 R 0000          MOV     CS:WORD PTR START_SEC,0
```

```
01D5  2E: C7 06 0025 R 0001           MOV     CS:WORD PTR TOTAL,1
01DC  E8 007D R                       CALL    CALC_ADDR               ;CALCULATE ADDRESS OF FIRST SECTOR
01DF  0E                              PUSH    CS
01E0  07                              POP     ES
01E1  8D 3E 0016 R                    LEA     DI,BPB                  ;ADDRESS OF BIOS PARAMETER BLOCK
01E5  83 C6 0B                        ADD     SI,11                   ;ADD 11 TO SI
01E8  B9 000D                         MOV     CX,13                   ;LENGTH OF BPB
01EB  F3/ A4                    REP   MOVSB
01ED  5B                              POP     BX                      ;RESTORE OFFSET OF SRH
01EE  07                              POP     ES                      ;RESTORE SEGMENT OF SRH
01EF  8D 16 0016 R                    LEA     DX,BPB                  ;GET BPB ARRAY POINTER
01F3  26: 89 57 12                    MOV     ES:BPBA_PTR[BX],DX      ;SAVE PTR TO BPB TABLE
01F7  26: 8C 4F 14                    MOV     ES:BPBA_PTR+2[BX],CS
01FB  26: 89 57 0E                    MOV     ES:DTA[BX],DX           ;OFFSET OF SECTOR BUFFER
01FF  26: 8C 4F 10                    MOV     ES:DTA+2[BX],CS
                                      STATUS  DONE,NOERROR,0
0203  26: 81 4F 03 0100         +      OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
0209  26: 81 4F 03 0000         +      OR      ES:WORD PTR SRH_STA_FLD[BX],0
020F  EB 77 90                        JMP     EXIT
                                ;
                                ; THE FOLLOWING ENTRIES ARE FOR NOT SUPPORTED BY THIS DEVICE
                                ;
0212                            IOCTL_IN:
0212                            IOCTL_OUT:
0212                            ND_INPUT:                               ;NON_DESTRUCTIVE INPUT NO WAIT (CHAR ONLY)
0212                            IN_STAT:                                ;INPUT STATUS                   "   "
0212                            IN_FLUSH:                               ;INPUT FLUSH                    "   "
0212                            OUT_STAT:                               ;OUTPUT STATUS                  "   "
0212                            OUT_FLUSH:                              ;OUTPUT FLUSH                   "   "
                                ;
                                ; DISK READ
                                ;
0212                            INPUT:
0212  E8 0062 R                         CALL    IN_SAVE                 ;CALL THE INITIAL SAVE ROUTINE
0215  26: 8B 47 14                      MOV     AX,ES:WORD PTR SSN[BX]  ;SET STARTING SECTOR NUMBER
0219  2E: A3 0028 R                     MOV     CS:START_SEC,AX         ;SAVE STARTING SECTOR NUMBER
021D  26: 8B 47 12                      MOV     AX,ES:WORD PTR COUNT[BX]
0221  2E: A3 0025 R                     MOV     CS:TOTAL,AX             ;SAVE TOTAL SECTORS TO TRANSFER
0225  E8 00A3 R                         CALL    SECTOR_READ             ;READ IN THAT MANY SECTORS
0228  2E: 8B 1E 0012 R                  MOV     BX,CS:RH_OFF            ;RESTORE ES:BX AS REQUEST HEADER POINTER
022D  2E: 8E 06 0014 R                  MOV     ES,CS:RH_SEG
                                        STATUS  DONE,NOERROR,0          ;SET STATUS WORD (DONE, NOERROR)
0232  26: 81 4F 03 0100         +        OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
0238  26: 81 4F 03 0000         +        OR      ES:WORD PTR SRH_STA_FLD[BX],0
023E  EB 48 90                          JMP     EXIT
                                ;
                                ; DISK WRITE
                                ;
0241                            OUTPUT:                                 ;OUTPUT (WRITE)
0241  E8 0062 R                         CALL    IN_SAVE
0244  26: 8B 47 14                      MOV     AX,ES:WORD PTR SSN[BX]  ;GET STARTING SECTOR NUMBER
0248  2E: A3 0028 R                     MOV     CS:START_SEC,AX         ;SET    "       "       "
024C  26: 8B 47 12                      MOV     AX,ES:WORD PTR COUNT[BX]
0250  2E: A3 0025 R                     MOV     CS:TOTAL,AX             ;SAVE TOTAL SECTORS TO WRITE
0254  E8 00C0 R                         CALL    SECTOR_WRITE            ;WRITE OUT THOSE SECTORS
```

```
0257  2E: 8B 1E 0012 R                MOV     BX,CS:RH_OFF            ;RESTORE ES:BX AS REQUEST HEADER POINTER
025C  2E: 8E 06 0014 R                MOV     ES,CS:RH_SEG
0261  2E: 80 3E 0027 R 00             CMP     CS:BYTE PTR VERIFY,0    ;WRITE VERIFY SET
0267  74 08                           JZ      NO_VERIFY               ;NO, NO WRITE VERIFY
0269  2E: C6 06 0027 R 00             MOV     CS:BYTE PTR VERIFY,0    ;RESET VERIFY INDICATOR
026F  EB A1                           JMP     INPUT                   ;READ THOSE SECTORS BACK IN
0271                                  NO_VERIFY:
                                      STATUS  DONE,NOERROR,0          ;SET DONE, NO ERROR IN STATUS WORD
0271  26: 81 4F 03 0100      +         OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
0277  26: 81 4F 03 0000      +         OR      ES:WORD PTR SRH_STA_FLD[BX],0
027D  EB 09 90                        JMP     EXIT
0280                                  OUT_VERIFY:                     ;OUTPUT (WRITE) WITH VERIFY
0280  2E: C6 06 0027 R 01             MOV     CS:BYTE PTR VERIFY,1    ;SET THE VERIFY FLAG
0286  EB B9                           JMP     OUTPUT                  ;BRANCH TO OUTPUT ROUTINE
                                      ;
                                      ; COMMON EXIT
                                      ;
0288                                  EXIT:
0288  5E                              POP     SI                      ;RESTORE ALL OF THE REGISTERS
0289  5F                              POP     DI
028A  5A                              POP     DX
028B  59                              POP     CX
028C  5B                              POP     BX
028D  58                              POP     AX
028E  07                              POP     ES
028F  1F                              POP     DS
0290  CB                              RET
0291                                  E_O_P:
                                      ; MACRO TO ALIGN THE VIRTUAL DISK ON A PARAGRAPH BOUNDARY
                                      if ($-START) MOD 16
02A0                                  ORG     ($-START)+16-(($-START) MOD 16)
                                      endif
= 02A0                                VDISK   EQU     $
02A0                                  VDSK    ENDP
02A0                                  CSEG    ENDS
                                              END     BEGIN
```

Appendixes

Contents

Appendix A. DOS Version 2.00 Enhancements

The information in this appendix is divided into two categories—those topics that apply to all users, and those topics that apply to system programmers or application developers. In each case, a brief description of the feature or change is offered, and you are referred to another section of the book for further details.

For All Users

DOS Version 2.00 incorporates the following new and changed features:

- *Special characters.* The characters <, >, , and now have special meanings to DOS, and can no longer be used in filenames. If you have files whose names contain any of these characters, they should be renamed (using your *old* version of DOS) before attempting to use them with DOS Version 2.00.

- *Configuration file.* You can create a file of special commands that DOS will read each time it starts up. The commands allow you to specify the number of disk buffers DOS should use, the names of device drivers, and additional information concerning DOS operation. Please refer to "Configuring Your System" in Chapter 5 (Section 1) for additional information.

- *Support for one or more fixed disk devices.* The disk can be divided into multiple partitions, each usable by a different operating system. You can start (boot) your operating system from the fixed disk, and utility programs included to perform disk initialization, backup, and restore functions. If you have a fixed disk, please read the "Preparing Your Fixed Disk" information in Chapter 4 for setup instructions and the BACKUP and RESTORE commands in Chapter 6 for their respective functions.

- *Support for increased diskette capacity.* Beginning with DOS Version 2.00, DOS formats diskettes at 9 sectors per track, which increases capacity from 163840 to 184320 characters of information for single-sided diskettes, and from 327680 to 368640 characters for dual-sided diskettes. The smaller capacity diskettes created by DOS Version 1.00 or DOS Version 1.10 (8 sectors per track) are also usable with DOS Version 2.00. You do not need to reformat them. Please see the FORMAT and DISKCOPY commands in Chapter 6 for more information.

- *Multiple disk buffers.* A *disk buffer* is an area of user memory that DOS reserves at startup and is used for performing disk and diskette operations. DOS normally allocates two disk buffers at start-up time. Some users, however, will find that certain applications, such as data base applications, may run faster if DOS has more buffers available to it. DOS Version 2.00 allows you to specify the number of buffers that DOS should reserve at start-up time. Please refer to "Configuring Your System" in Chapter 9 for instructions on specifying additional buffers.

- *Tree-Structured Directories.* This new feature allows you to place related groups of files in their own directories—all on the same disk. The individual directories are isolated from each other, giving the appearance of separate disks. Therefore, a search for a file in a given directory will not "see" files in other directories on the same disk.

  Each directory, beginning with the normal system directory (called the *root directory*) may contain special entries naming other directories on the same disk. These other directories, in turn, may contain entries for even more directories, and so on. When viewed in a logical order beginning with the root directory, the directory structure appears much like a diagram of a family tree—thus the term *tree-structured directories*.

  You may add or remove directories, copy files from one directory to another, instruct DOS to look in a specific directory to locate a file, etc. For complete details, please refer to Chapter 5 "Using Tree-Structured Directories".

- *Disk Volume Labels.* This feature allows you to specify a unique volume label (up to 11 characters) at the time you format a disk. The volume label is placed in the root directory, and is included in the displays produced by the DIR, CHKDSK, and TREE commands. Please refer to the FORMAT command for further information.

- *Extended DOS screen and keyboard control.* This feature allows you to issue special character sequences from within your program that DOS will use for screen cursor positioning and color, and further allows you to assign the meaning of any key on the keyboard. For example, you may assign the character string "DIR A:" to the F10 key so that simply pressing the F10 key has the same result as entering the **DIR A:** command. Please refer to "Using Extended Screen and Keyboard Control" in Chapter 13, and "Configuring Your System" in Chapter 9 for more detailed information.

- *Redirection of Standard Input and Output.* This feature applies to all DOS programs that read from the keyboard or write to the screen (the standard input and output devices). By using the special characters < (for input) and > (for output), you can cause a program to receive its input from a source other than the keyboard, or to direct its output to a destination other than the screen. For example, the command:

  DIR A:>DIRLIST

  causes the directory listing from drive A to be placed in a file named DIRLIST on the default drive. Device names can also be used. For example, the command:

  DIR A:>PRN

  causes the directory listing to appear on the printer instead of the screen. Please refer to "Redirection of Standard Input and Output" in Chapter 10 for further information.

- *Piping of standard input and output*. This feature allows the standard output of one program to be used as the standard input to another. DOS acts as a "pipeline" to direct the output of the first program to the input of the second—thus, the *term* "piping." For further information and an example of its use, please refer to "Piping of Standard Input and Output" in Chapter 10.

New Commands

The following new commands have been added to DOS Version 2.00. Please consult the command descriptions in Chapter 6 and Chapter 10 for further details and examples of their use.

ASSIGN

Allows you to reassign drive letters so that a request for a given drive can be routed to a different drive.

BACKUP

Backs up one or more files from a fixed disk to diskettes.

BREAK

Allows you to specify when DOS should check for a Ctrl-Break being entered at the keyboard. Normally, DOS only performs this check during screen, keyboard, printer, or auxiliary device operation. With this command, you can instruct DOS to check for Ctrl-Break whenever a program requests DOS to perform *any* function (such as disk operations). In this way, it is possible to "break out" of a program that performs few or no screen or keyboard operations (such as a compiler).

CLS

Clears the screen when used from a batch file or the keyboard.

CTTY

Allows you to define a different primary console, so that a remote terminal device can be used in place of the standard screen and keyboard. This command also reverses this assignment, to restore the keyboard and screen as the standard input and output devices.

ECHO, IF, FOR, SHIFT, GOTO

New subcommands provided to extend the flexibility of batch processing.

FDISK

Initializes and configures a fixed disk.

> **Note:** This command must be used before you use your fixed disk for the first time. Please refer to Chapter 4 "Preparing Your Fixed Disk".

GRAPHICS

Allows the Shift-PrtSc keys (display screen contents on printer) to print the image of a graphics display screen.

MKDIR, RMDIR and CHDIR

Create, remove, and inform DOS to use a directory other than the system directory.

PATH

Allows you to specify one or more paths of directory names that DOS will search if the command you have issued was not found in the current directory. They allow conditional execution of commands within a batch file by causing DOS to check for specified conditions.

PRINT

Prints a queue (list) of files on the system printer while you are using the system for other work.

PROMPT

Allows you to change the system prompt to a desired string.

RECOVER

Recovers a specific file that cannot be copied or otherwise used because of a defective spot on the disk that prevents the file from being read. This command also recovers multiple files when the directory has been damaged.

RESTORE

Restores one or more files from a diskette to a fixed disk.

SET

Allows you to enter keywords and parameters into a DOS "environment" that is accessible by commands and applications.

TREE

Displays the entire directory structure of the specified disk.

VER

Displays the DOS version number on the screen.

VERIFY

Instructs DOS to perform a verify operation (or to stop performing the verify) each time data is written to disk, until a new verify command is issued to turn the verify feature off. The verify operation increases assurance that the data was properly recorded on disk (that is, it can be read without error).

VOL

Displays the volume label of the disk in the specified drive.

Enhanced Commands

The following commands, that existed in DOS Version 1.10, have been enhanced for DOS Version 2.00. For more detailed information, please consult the individual command descriptions in Chapter 6 and Chapter 10.

CHKDSK

Supports the fixed disk, the new and old diskette formats, and analyzes all directories on the volume. It also enables you to create files containing all sectors that were found to be allocated but were not associated with a file, so that you can recover "lost" data. An important feature is that, unlike Version 1.10 CHKDSK, it will take *no* corrective action on the disk being analyzed unless instructed to do so.

COMP

The file compare utility now allows multiple files to be compared. For example, you can compare all of the files on one disk with their counterparts on another disk. Also, COMP no longer prompts you to insert diskettes before comparing.

DEBUG

Now contains a command allowing you to enter assembly language statements that are assembled directly into memory.

DIR

Now displays the volume identification of the specified disk and clearly identifies entries that contain the names of other directories. It also displays the amount of available space left on the disk.

DISKCOPY and DISKCOMP

Support the new 9-sector-per-track diskette format.

EDLIN

The line editor contains several new subcommands for more flexible management of source data. They include commands to copy and move lines, and to merge the contents of another file. The Replace and Search commands have been changed to begin their search at the current line plus one.

ERASE

Now requires you to press the Enter key after entering the Y/N response to the

Are you sure (Y/N)?

message that appears when you instruct DOS to erase all of the files on a volume. This is intended to prevent accidental erasure of all files from the larger capacity devices supported by DOS Version 2.00.

FORMAT

Now formats diskettes at 9 sectors per track (in DOS 1.00 and 1.10 diskettes were formatted at 8 sectors per track), allowing each new diskette to hold more data. It also allows you to specify a volume identification that is recorded in the disk's directory. Support for initializing a fixed disk is also included.

LPT2:, LPT3:, and COM2:

Now recognized as valid device names by DOS, and can be used in place of filenames.

For Programmers

- DOS Version 2.00 includes the ability to install your own device drivers for character or block-oriented devices. Please refer to Chapter 14 "Installable Device Drivers" for more details.

- Three changes were made to internal functions that cause different results from those obtained on DOS Version 1.10:

  1. The function call (hex 1B) that previously returned a pointer to the file allocation table now returns a pointer to only the table's identification byte, for purposes of determing the disk type. All applications that use call hex 1B to obtain the file allocation table should be changed to use interrupt hex 25 to read the file allocation table directly from the disk. The file allocation table always begins at logical sector 1, and its size can be determined from the information returned by call hex 36. We recommend that you avoid using calls hex 1B and hex 1C.

  2. The mapping of logical sectors on dual sided diskettes has been rearranged to facilitate program loading and to improve system performance.

     This change allows DEBUG to load an entire file with a single L command.

     Applications that use interrupts hex 25 and hex 26 on multi-sided disks or diskettes may require modification to operate properly on DOS Version 2.00. Please see the description of Int 25 in Appendix D.

3. Additional bits have been defined in the file attribute byte of the DOS disk directory. Programs that depended upon the file attribute byte being equal to zero if a file was not a hidden or system file may not work correctly. (See Appendix C for the definition of the new attribute bits.)

- A new set of function calls has been made available to provide a wide variety of services. We suggest that systems programmers and application developers review all of Appendix D for details on these new functions.

Notes:

Appendix B. DOS Technical Information

Appendixes B-K are intended to supply technically oriented users with information about the structure, facilities, and program interfaces of DOS. It is assumed that the reader is familiar with the 8088 architecture, interrupt mechanism, and instruction set.

DOS Structure

DOS consists of the following four components:

1. The boot record resides on track 0, sector 1, side 0 of every disk formatted by the FORMAT command. It is put on all disks in order to produce an error message if you try to start up the system with a non-DOS diskette in drive A. For fixed disks, it resides on the first sector (sector 1, head 0) of the first cylinder of the DOS partition.

2. The Read-Only Memory (ROM) BIOS interface module (file IBMBIO.COM) provides a low-level interface to the ROM BIOS device routines.

3. The DOS program itself (file IBMDOS.COM) provides a high-level interface for user programs. It consists of file management routines, data blocking/deblocking for the disk routines, and a variety of built-in functions easily accessible by user programs. (Refer to Appendix D.)

   When these function routines are invoked by a user program, they accept high-level information via register and control block contents, then (for device operations) translate the requirement into one or more calls to IBMBIO to complete the request.

4. The command processor, COMMAND.COM.

DOS Initialization

When the system is started (either System Reset or power ON with the DOS diskette in drive A), the boot record is read into memory and given control. It checks the directory to assure that the first two files listed are IBMBIO.COM and IBMDOS.COM, in that order. (An error message is issued if not.) These two files are then read into memory. (IBMBIO.COM must be the first file in the directory, and its sectors must be contiguous.)

The initialization code in IBMBIO.COM determines equipment status, resets the disk system, initializes the attached devices, causes device drivers to be loaded, and sets the low-numbered interrupt vectors. It then relocates IBMDOS.COM downward and calls the first byte of DOS.

As in IBMBIO.COM, offset 0 in DOS contains a jump to its initialization code, which will later be overlaid by a data area and the command processor. DOS initializes its internal working tables, initializes interrupt vectors for interrupts hex 20 through hex 27 and builds a Program Segment Prefix (see Appendix E) for COMMAND.COM at the lowest available segment, then returns to IBMBIO.COM.

The last remaining task of initialization is for IBMBIO.COM to load COMMAND.COM at the location set up by DOS initialization. IBMBIO.COM then passes control to the first byte of COMMAND.

The Command Processor

The command processor supplied with DOS (file COMMAND.COM) consists of four distinctly separate parts:

- A resident portion resides in memory immediately following IBMDOS.COM and its data area. This portion contains routines to process interrupt types hex 22 (terminate address), hex 23 (CTRL-BREAK handler), and hex 24 (critical error handling), as well as a routine to reload the transient portion if needed. (When a program terminates, a checksum methodology determines if the program had caused the transient portion to be overlaid. If so, it is reloaded.) Note that all standard DOS error handling is done within this portion of COMMAND. This includes displaying error messages and interpreting the reply of Abort, Retry, or Ignore. (See message **Disk error reading drive** *x* under "Device Error Message" at the beginning of Chapter 8.)

- An initialization portion follows the resident portion and is given control during startup. This section contains the AUTOEXEC file processor setup routine. The initialization portion determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND loads because it's no longer needed.

- A transient portion is loaded at the high end of memory. This is (portion 3) the command processor itself, containing all of the internal command processors, the batch file processor, and (portion 4) a routine to load and execute external commands (files with filename extensions of .COM or .EXE). This "loader" is at the highest end of memory, and is invoked by the EXEC function call to load programs.

  Portion 3 of COMMAND produces the system prompt (such as A>), reads the command from the keyboard (or batch file) and causes it to be executed. For external commands, it builds a command line and issues an EXEC function call to load and transfer control to the program.

Appendix E contains detailed information describing the conditions in effect when a program is given control by EXEC.

Available DOS Functions

DOS provides a significant number of functions to user programs, all available through issuance of a set of interrupt codes. There are routines for keyboard input (with and without echo and Ctrl-Break detection), console and printer output, constructing file control blocks, memory management, date and time functions, and a variety of disk, directory, and file handling functions. See "DOS Interrupts and Function Calls" in Appendix D for detailed information.

File Management Notes

Through the INT 21 (function call) mechanism, DOS provides methods to create, read, write, rename, and erase files. Files are not necessarily written sequentially on disk—space is allocated as it is needed, and the first location available on the disk is allocated as the next location for a file being written. Therefore, if considerable file creation and erasure activity has taken place, newly created files will probably *not* be written in sequential sectors.

However, due to the mapping (chaining) of file space via the File Allocation Table, and the function calls available, any file can be used in either a sequential or random manner.

There are two sets of function calls that support file management. The new, extended set of calls is the preferred method (functions 39 through 57). Through these calls, sequential and random file accesses are simpler than using the traditional (FCB oriented) set of calls. The FCB calls continue to function as in the past: By using the current block and current record fields of the FCB, and the sequential disk read or write functions, you can make the file appear sequential—DOS will do the calculations necessary to locate the proper sectors on the disk. On the other hand, by using the random record field, and random disk functions, you can cause any record in the file to be accessed *directly*—again, DOS will locate the correct sectors on the disk for you.

Space is allocated in increments called *clusters*. For single sided diskettes, this unit of allocation is one sector; for dual sided diskettes, each cluster is two consecutive sectors in length. The cluster size of a fixed disk is determined at FORMAT time, and is based on the size of the DOS partition.

The Disk Transfer Area (DTA)

The Disk Transfer Area (also commonly called *buffer*) is the memory area DOS will use to contain the data for all file reads and writes that are performed with the traditional (FCB) set of function calls. This area can be at any location within memory, and should be set by your program. (See function call hex 1A.)

Only one DTA can be in effect at a time, so it is the program's responsibility to inform DOS what memory location to use *before* using any disk read or write functions. Once set, DOS continues to use that area for all disk operations until another function call hex 1A is issued to define a new DTA. When a program is given control by COMMAND, a default DTA has already been established at hex 80 into the program's Program Segment Prefix, large enough to hold 128 bytes.

When using the extended file management function calls, you specify a buffer address when you issue the read or write call. There is no need to set a DTA address.

Error Trapping

DOS provides a method by which a program can receive control whenever a disk or device read/write error occurs, or when a bad memory image of the file allocation table is detected. When these events occur, DOS executes an INT hex 24 to pass control to the error handler. The default error handler resides in COMMAND.COM, but any program can establish its own by setting the INT hex 24 vector to point to the new error handler. DOS provides error information via the registers and provides Abort, Retry, or Ignore support via return codes. (Refer to Appendix D "DOS Interrupts and Function Calls".)

Notes:

Appendix C. DOS Disk Allocation

All disk and diskettes formatted by DOS are created with a sector size of 512 bytes. The DOS area (entire diskette for diskettes, DOS partition for fixed disks) is formatted as follows:

Boot record - variable size
First copy of file allocation table - variable size
Second copy of file allocation table - variable size
Root directory - variable size
Data area

Allocation of space for a file (in the data area) is done only when needed (it is not pre-allocated). The space is allocated one cluster (unit of allocation) at a time. A cluster is always one or more consecutive sectors, and all of the clusters for a file are "chained" together in the File Allocation Table.

The clusters are arranged on disk to minimize head movement for multi-sided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is accomplished by using the sequential sectors on the lowest-numbered head, then all the sectors on the next head, and so on until all sectors on all heads of the track are used. Then, the next sector to be used will be sector 1 on head 0 of the next track.

For fixed disk, the size of the file allocation table and directory are determined when FORMAT initializes it, and are based on the size of the DOS partition.

For diskettes, the following table can be used:

# Sides	Sectors/ Track	FAT size Sectors	Dir Sectors	Dir Entries	Sectors/ Cluster
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2

Files in the data area are not necessarily written sequentially on the disk. The data area space is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files. (Refer to the description of the "DOS File Allocation Table".)

DOS Disk Directory

FORMAT initially builds the root directory for all disks. Its location (logical sector number) and the maximum number of entries are available through the device driver interfaces.

Since directories other than the root directory are actually files, there is no limit to the number of entries they may contain. Sub-directories can be read as data files, using an extended FCB with the appropriate attribute byte.

All directory entries are 32 bytes in length, and are in the following format (byte offsets are in decimal):

0-7 Filename. The first byte of this field indicates its status.

- hex 00 Never been used. This is used to limit the length of directory searches, for performance reasons.
- hex E5 Was used, but the file has been erased.
- hex 2E The entry is for a directory. If the second byte is also hex 2E, then the cluster field contains the cluster number of this directory's parent directory (hex 0000 if the parent directory is the root directory).

Any other character is the first character of a filename.

8-10 Filename extension.

11 File attribute. The attribute byte is mapped as follows (values are in hexadecimal):

01 File is marked read-only. An attempt to open the file for output using function call hex 3D results in an error code being returned. This value can be used along with other values below.

02 Hidden file. The file is excluded from normal directory searches.

04 System file. The file is excluded from normal directory searches.

08 The entry contains the volume label in the first 11 bytes. The entry contains no other usable information, and may exist only in the root directory.

10 The entry defines a sub-directory, and is excluded from normal directory searches.

20 Archive bit. The bit is set on whenever the file has been written to and closed. It is used by backup/restore functions of the FDISK utility for determining whether or not the file was changed since it was last backed up. This bit can be used along with other attribute bits.

Note: The system files (IBMBIO.COM and IBMDOS.COM) are marked as read only, hidden, and system files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the CHMOD function call.

12-21 Reserved.

22-23 Time the file was created or last updated. The time is mapped in the bits as follows:

```
<    hh    >< mm  >< xx  >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

where:

hh is the binary number of hours (0-23)
mm is the binary number of minutes (0-59)
xx is the binary number of two-second increments

24-25 Date the file was created or last updated. The mm/dd/yy are mapped in the bits as follows:

```
<        25          >< 24       >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
y  y  y  y  y  y  y  m m m m d d d d d
```

where:

mm is 1-12
dd is 1-31
yy is 0-119 (1980-2099)

26-27 Starting cluster; the relative cluster number of the first cluster in the file.

Note that the first cluster for data space on all fixed disks and diskettes is always cluster 002.

The cluster number is stored with the least significant byte first.

> **Note:** System programmers, see "DOS File Allocation Table" for details about converting cluster numbers to logical sector numbers.

28-31 File size in bytes. The first word contains the low-order part of the size. Both words are stored with the least significant byte first.

DOS File Allocation Table

This information is presented for the benefit of system programmers who wish to develop device drivers. It explains how DOS uses the File Allocation Table to convert the clusters of a file to logical sector numbers. The driver is then responsible for locating the logical sector on disk. We wish to emphasize that this information should not be used for any other purpose. We recommend that system utilities use the DOS file management function calls rather than interpreting the FAT.

The File Allocation Table (FAT) is used by DOS to allocate disk space for a file, one cluster at a time.

The FAT consists of a 12-bit entry (1.5 bytes) for each cluster on the disk.

Note that the first two FAT entries map a portion of the directory; these FAT entries contain indicators of the size and format of the disk.

The second and third bytes always contain hex FFFF. The first byte is used as follows:

Hex Value	Meaning
FF	Dual sided, 8 sector-per-track diskette.
FE	Single sided, 8 sector-per-track diskette.
FD	Dual sided, 9 sector-per-track diskette.
FC	Single sided, 9 sector-per-track diskette.
F8	Fixed disk

The third FAT entry begins the mapping of the data area (cluster 002).

Each entry contains three hexadecimal characters, either:

000	if the cluster is unused and available, or
FF8-FFF	to indicate the last cluster of a file, or
XXX	any other hexadecimal characters that are the cluster number of the *next cluster* in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

> **Note:** The values FF0-FF7 are used to indicate reserved clusters (FF7 indicates a bad cluster if it is not part of an allocation chain), and FF8-FFF are used as end-of-file marks.

The File Allocation Table always begins on logical sector 1 (second actual sector on a diskette or in a fixed disk partition), following the boot record. If larger than 1 sector, the sectors are contiguous. Two copies of the FAT are written, one following the other, for integrity. The FAT is read into one of the DOS buffers whenever needed (open, allocate more space, etc.), and that buffer is given a high priority to keep it in memory as long as possible, for performance reasons.

How to Use the File Allocation Table

Obtain the *starting cluster* of the file from the directory entry.

Now, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).

2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.

3. Use a MOV instruction to move the word at the calculated FAT offset into a register.

4. If the last cluster used was an even number, keep the low-order 12 bits of the register; otherwise, keep the high-order 12 bits.

5. If the resultant 12 bits are hex FF8-FFF, there are no more clusters in the file. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by INT 25 and 26 and by DEBUG):

1. Subtract 2 from the cluster number.

2. Multiply the result by the number of sectors per cluster.

3. Add the logical sector number of the beginning of the data area.

Notes:

Appendix D. DOS Interrupts and Function Calls

Interrupts

> **Note:** We recommend that a program wishing to examine or set the contents of any interrupt vector use the DOS function calls (hex 35 and hex 25) provided for those purposes, and avoid referencing the interrupt vector locations directly.

DOS reserves interrupt types hex 20 to hex 3F for its use. This means absolute memory locations hex 80 to hex FF are reserved by DOS. The defined interrupts are as follows with all values in hexadecimal.

20 Program terminate. Issuing Interrupt hex 20 is the traditional way to exit from a program. This vector transfers to the logic in DOS for restoration of the terminate, Ctrl-Break, and critical error exit addreses to the values they had on entry to the program. All file buffers are flushed. All files changed in length should be closed (see function call hex 10 and hex 3E) prior to issuing this interrupt. If the changed file is not closed, its length, date, and time are not recorded correctly in the directory.

In order for a program to pass a completion (or error) code when terminating, it must use either function call hex 4C (exit) or hex 31 (terminate and stay resident). These two new methods are preferred over using interrupt hex 20, and the codes returned by them can be interrogated in batch processing (see ERRORLEVEL subcommand of batch processing).

Important: Every program must ensure that the CS register contains the segment address of its Program Segment Prefix control block prior to issuing interrupt hex 20.

21 Function request. Refer to "Function Calls" in this appendix.

22 Terminate address. The address found at this interrupt location is the address to which control transfers when the program terminates. This address is copied into the program's Program Segment Prefix at the time the segment is created. If a program wishes to execute a second program it must set the terminate address prior to issuing the EXEC function call to execute the new program. Otherwise, when the second program executes, its termination would cause transfer to its host's termination address. This address, as well as the Ctrl-Break address below, may be set via DOS function call hex 25. Do not issue this interrupt directly.

23 Ctrl-Break exit address. If the user enters Ctrl-Break during screen, printer, or asynchronous communications adapter operations, an interrupt type hex 23 is executed. (If BREAK is on, the interrupt hex 23 is issued on *any* function call.) If the Ctrl-Break routine saves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution. If the program returns with a long return, the carry flag is used to determine whether the program will be aborted or not; if the carry flag is set, it will be aborted, otherwise execution will continue (as with a return by IRET). If the Ctrl-Break interrupts functions 9 or 10, buffered I/O, then ^C, carriage-return, and linefeed are output. If execution is then continued with an IRET, I/O continues from the start of the line. When the interrupt occurs, all registers are set to the value they had when the original function call to DOS was made. There are no restrictions on what the Ctrl-Break handler is allowed to do, including DOS function calls, as long as the registers are unchanged if IRET is used.

If the program creates a new segment and loads in a second program which itself changes the Ctrl-Break address, the termination of the second program and return to the first causes the Ctrl-Break address to be restored to the value it had before execution of the second program. (It is restored from the second program's Program Segment Prefix.)

24 Critical error handler vector. When a critical error occurs within DOS, control is transferred with an interrupt 24H. On entry to the error handler, AH will have its bit 7=0 (high-order bit) if the error was a disk error (probably the most common occurrence), bit 7=1 if not.

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved (see below).

The registers will be set up for a retry operation, and an error code will be in the lower half of the DI register with the upper half undefined. These are the error codes:

Error Code	Description
0	Attempt to write on write-protected diskette
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data error (CRC)
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack will be in effect (the first item described below is at the top of the stack), and will contain the following from top to bottom:

IP CS FLAGS	DOS registers from issuing INT hex 24
AX BX CX DX SI DI BP DS ES	User registers at time of original INT hex 21 request
IP CS FLAGS	From the original interrupt hex 21 from the user to DOS

The registers are set such that if an IRET is executed, DOS will respond according to (AL) as follows:

(AL)=0 ignore the error.

=1 retry the operation.

=2 terminate the program through interrupt hex 23.

Disk Errors

If it is a hard error on disk (AH bit 7=0), register AL contains the failing drive number (0 = drive A, etc.); AH bits 0-2 indicate the affected disk area and whether it was a read or write operation, as follows:

Bit 0=0 if read operation,
1 if write operation

Bits 2-1 (affected disk area)

0 0 DOS area (system files)
0 1 file allocation table
1 0 directory
1 1 data area

Other Errors

If AH bit 7=1, then the error occurred on a character device, or was the result of a bad memory image of the FAT. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

If a character device, the contents of AL are unpredictable, the error code is in DI as above.

Notes:

1. Before giving this routine control for disk errors, DOS performs five retries.

2. For disk errors, this exit is taken only for errors occurring during an interrupt hex 21 function call. It is not used for errors during an interrupt hex 25 or hex 26.

3. This routine is entered in a disabled state.

4. The SS, SP, DS, ES, BX, CX, and DX registers must be preserved.

5. This interrupt handler should refrain from using DOS function calls. If necessary, it may use calls 1 through 12. Use of any other call will destroy the DOS stack and will leave DOS in an unpredictable state.

6. The interrupt handler must not change the contents of the device header.

7. If the interrupt handler will handle errors itself rather than returning to DOS, it should restore the application program's registers from the stack, remove all but the last 3 words on the stack, then issue an IRET. This will return to the program immediately after the INT 21 that experienced the error. Note that if this is done, DOS will be in an unstable state until a function call higher than 12 is issued.

The device header pointed to by BP:SI is formatted as follows:

DWORD Pointer to next device (FFFF if last device)
WORD Attributes Bit 15 = 1 if character device, 0 if block if bit 15 is 1 Bit 0 = 1 if Current standard input Bit 1 = 1 if Current standard output Bit 2 = 1 if Current NUL device Bit 3 = 1 if Current CLOCK device Bit 14 is the IOCTL bit
WORD Pointer to Device driver strategy entry point
WORD Pointer to Device driver interrupt entry point
8-BYTE character device named field for block devices the first byte is the number of units

To tell if the error occurred on a block or character device you must look at bit 15 in the attribute field (WORD at BP:SI+4).

If the name of the character device is desired, look at the eight bytes starting at BP:SI+10.

25 Absolute disk read. This transfers control directly to the DOS BIOS. Upon return, the original flags are still on the stack (put there by the INT instruction). This is necessary because return information is passed back in the current flags. Be sure to pop the stack to prevent uncontrolled growth. The request is as follows:

(AL)	Drive number (for example, 0=A or 1=B)
(CX)	Number of sectors to read
(DX)	Beginning logical sector number
(DS:BX)	Transfer address

The number of sectors specified are transferred between the given drive and the transfer address. *Logical sector numbers* are obtained by numbering each sector sequentially starting from track 0, head 0, sector 1 (logical sector 0) and continuing along the same head, then to the next head until the last sector on the last head of the track is counted. Thus, logical sector 1 is track 0, head 0, sector 2; logical sector 2 is track 0, head 0, sector 3; and so on. Numbering then continues with sector 1 on head 0 of the next track. Note that although the sectors are sequentially numbered (for example, sectors 2 and 3 on track 0 in the example above), they may not be physically adjacent on disk, due to interleaving. Note that the mapping is different from that used by DOS Version 1.10 for dual-sided diskettes.

All registers except the segment registers are destroyed by this call. If the transfer was successful the carry flag (CF) will be zero. If the transfer was not successful CF=1 and (AX) will indicate the error as follows. (AL) is the DOS error code that is the same as the error code returned in the low byte of DI when an INT hex 24 is issued, and (AH) will contain:

hex 80	Attachment failed to respond
hex 40	SEEK operation failed
hex 20	Controller failure
hex 10	Bad CRC on diskette read
hex 08	DMA overrun on operation
hex 04	Requested sector not found
hex 03	Write attempt on write-protected diskette
hex 02	Address mark not found
hex 00	Error other than types listed above

26 Absolute disk write. This vector is the counterpart of interrupt 25 above. Except for the fact that this is a write, the description above applies.

27 Terminate but stay resident. This vector is used by programs that are to remain resident when COMMAND regains control. This is the traditional method for DOS programs to remain resident upon termination.

A new DOS function call has been established that allows the terminating program to pass a completion (or error) code to DOS, that can be interpreted within batch processing (see function call hex 31). After initializing itself, the program must set DX to its last address plus one in the segment in which it is executing (the offset at which other programs can be loaded), then execute an INT 27H. DOS then considers the program as an extension of DOS, so the program is not overlaid when other programs are executed. This concept is very useful for loading programs such as user-written interrupt handlers that must remain resident.

Notes:

1. This interrupt must *not* be used by .EXE programs which are loaded into the high end of memory.

2. This interrupt restores the interrupt 22, 23, and 24 vectors in the same manner as INT 20. Therefore, it can not be used to install permanently resident Ctrl-Break or Critical Error Handler routines.

3. The maximum size of memory that can be made resident by this method is 64K. You can use call hex 31 to make a larger program resident.

28	Used internally by DOS.
29-2E	Reserved for DOS.
2F	Used internally by DOS.
30-3F	Reserved for DOS.

Function Calls

DOS provides a wide variety of function calls for character device I/O, file management, memory management, date and time functions, execution of other programs, and others. They are grouped as follows (call numbers are in hexadecimal):

0-12	Traditional character device I/O
12-24	Traditional file management
25-26	Traditional non-device functions
27-29	Traditional file management
2A-2E	Traditional non-device functions
2F-38	Extended function group
39-3B	Directory group
3C-46	Extended file management group

47	Directory group
48-4B	Extended memory management group
4C-4F	Extended function group
54-57	Extended function group

Functions 2F through 57 are new for DOS Version 2.00. Where similar functions exist in both this group and the group of traditional calls, we recommend using the new calls. They have been defined with simpler interfaces and provide more powerful functions than their traditional counterparts. However, if you use these new function calls your program cannot be run on DOS Versions 1.00 or 1.10.

When DOS takes control, it switches to an internal stack. User registers are preserved unless information is passed back to the requester as indicated in the specific requests. The user stack needs to be sufficient to accommodate the interrupt system. It is recommended that it be hex 80 in addition to the user needs.

Error Return Table

Many of the new function calls return the carry flag clear if the operation was successful. If an error condition was encountered, the carry flag is set, and AX contains one of the following binary error return codes:

Code	Condition
1	Invalid function number
2	File not found
3	Path not found
4	Too many open files (no handles left)
5	Access denied
6	Invalid handle
7	Memory control blocks destroyed
8	Insufficient memory
9	Invalid memory block address
10	Invalid environment
11	Invalid format
12	Invalid access code
13	Invalid data
15	Invalid drive was specified
16	Attempted to remove the current directory
17	Not same device
18	No more files

Several of the calls accept an ASCIZ string as input. This consists of an ASCII string containing an optional drive specifier, followed by a directory path, and in some cases a filename. The string is terminated by a byte of binary zeros. For example:

```
B:\LEVEL1\LEVEL2\FILE1
```

followed by a byte of zeros.

> **Note:** All calls that accept path names will accept a forward slash or a backslash as a path separator character.

The new calls supporting files or devices use an identifier known as a "handle." When you create or open a file or device with the new calls, a 16-bit binary value is returned in AX. This is the "handle" (sometimes known as a token) that you will use in referring to the file after it's been opened.

The following handles are pre-defined by DOS and can be used by your program. You do not need to open them before using them:

0000	Standard input device. Input can be redirected.
0001	Standard output device. Output can be redirected.
0002	Standard error output device. Output cannot be redirected.
0003	Standard auxiliary device.
0004	Standard printer device.

Invoking DOS Functions

Most of the function calls require input to be passed to them in registers. After setting the proper register values, the function may be invoked in one of these ways:

1. Place the function number in AH and execute a long call to offset hex 50 in your Program Segment Prefix. Note that programs using this method will not operate correctly on DOS Verisons 1.00 and 1.10.

2. Place the function number in AH and issue interrupt type hex 21.

3. There is an additional mechanism provided for pre-existing programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the DOS function dispatcher. Register AX is always destroyed if this mechanism is used; otherwise, it is the same as normal function calls. This method is valid only for function calls 0-24 (hexadecimal).

The functions are as follows with all values in hexadecimal.

0 Program terminate. The terminate, Ctrl-Break, and critical error exit addresses are restored to the values they had on entry to the terminating program, from the values saved in the Program Segment Prefix. All file buffers are flushed, but any files which have been changed in length but not closed will not be recorded properly in the directory. Control transfers to the terminate address. This call performs exactly the same function as INT 20H. It is the program's responsibility to ensure that the CS register contains the segment address of its Program Segment Prefix control block prior to calling this function.

> **Note:** Calls hex 1 through hex C use the standard devices listed at the end of the "Error Return Table" in this chapter.

1 Keyboard input. Waits for a character to be read at the standard input device (unless one is ready), then echoes the character to the standard output device and returns it in AL. The character is checked for a Ctrl-Break. If Ctrl-Break is detected, an interrupt hex 23 is executed.

> **Note:** For functions 1, 6, 7, and 8, extended ASCII codes will require two function calls. (See the IBM Personal Computer BASIC manual for a description of the extended ASCII codes.) The first call returns 00 as an indicator that the next call will return an extended code.

2 Display output. The character in DL is output to the standard output device. The backspace character results in moving the cursor left one position, writing a space at this position and remaining there. If a Ctrl-Break is detected after the output, an interrupt hex 23 is executed.

3 Auxiliary (Asynchronous Communications Adapter) input. Waits for a character from the standard auxiliary device, then returns that character in AL.

Notes:

1. Auxiliary (AUX, COM1, COM2) support is unbuffered and non-interrupt driven.

2. At startup, DOS initializes the first auxiliary port to 2400 baud, no parity, one stop bit, and 8-bit word.

3. The auxiliary function calls (3 and 4) do not return status or error codes. For greater control, it is recommended that the ROM BIOS routine (INT hex 14) be used.

4 Auxiliary (Asynchronous Communications Adapter) output. The character in DL is output to the standard auxiliary device.

5 Printer output. The character in DL is output to the standard printer device.

6 Direct console I/O. If DL is hex FF, AL returns with the zero flag clear and an input character from the standard input device if one is ready. If a character is not ready, the

zero flag will be set. If DL is not hex FF, then DL is assumed to have a valid character that is output to the standard output device. This function does not check for Ctrl-Break, or Ctrl-PrtSc.

7 Direct console input without echo. Waits for a character to be read at the standard input device (unless one is ready), then returns the character in AL. As with function 6, no checks are made on the character.

8 Console input without echo. This function is identical to function 1, except the key is not echoed.

9 Print string. On entry, DS:DX must point to a character string in memory terminated by a $ (hex 24). Each character in the string will be output to the standard output device in the same form as function 2.

A Buffered keyboard input. On entry, DS:DX point to an input buffer. The first byte must not be zero and specifies the number of characters the buffer can hold. Characters are read from the standard input device and placed in the buffer beginning at the third byte. Reading the standard input device and filling the buffer continues until Enter is read. If the buffer fills to one less than the maximum number of characters it can hold, then each additional character read is ignored and causes the bell to ring, until Enter is read. The second byte of the buffer is set to the number of characters received, excluding the carriage return (hex 0D), which is always the last character. Editing of this buffer is described in Chapter 3.

B Check standard input status. If a character is available from the standard input device, AL will be hex FF. Otherwise, AL will be 00. If a Ctrl-Break is detected, an interrupt type hex 23 is executed.

C Clear standard input buffer and invoke a standard input function. Clear the standard input buffer of any pre-typed characters, then execute the function number in AL (only 1, 6, 7, 8, and A are allowed). This forces the system to wait until a character is typed.

D Disk reset. Flushes all file buffers. Files changed in size but not closed are not properly recorded in the disk directory. This function need not be called before a diskette change if all files written have been closed.

E Select disk. The drive specified in DL (0=A, 1=B, etc.) is selected (if valid) as the default drive. The number of drives (total of diskette and fixed disk drives) is returned in AL. If the system has only one diskette drive, it will be counted as two to be consistent with the philosophy of thinking of the system as having logical drives A and B. BIOS equipment determination (INT 11H) can be used as an alternative method, returning the actual number of physical diskette drives.

F Open file. On entry, DS:DX point to a current unopened file control block (FCB). The directory is searched for the named file and AL returns hex FF if it is not found. If it is found, AL returns 00 and the FCB is filled as follows:

If the drive code was 0 (default drive), it is changed to the actual drive used (1=A, 2=B, etc.). This allows changing the default drive without interfering with subsequent operations on this file. The current block field (FCB bytes C-D) is set to zero. The size of the record to be worked with (FCB bytes E-F) is set to the system default of hex 80. The size of the file and the date are set in the FCB from information obtained from the directory.

It is your responsibility to set the record size (FCB bytes E-F) to the size you wish to think of the file in terms of, if the default hex 80 is insufficient. It is also your responsibility to set the random record field and/or current record field. These actions should be done after open but before any disk operations are requested.

10 Close file. This function must be called after file writes to ensure all directory information is updated. On entry, DS:DX point to an opened FCB. The current disk directory is searched and if the file is found, its position is compared with that kept in the FCB. If the file is not found in its correct position in the current directory, it is assumed the diskette was changed and AL returns hex FF. Otherwise, the directory is updated to reflect the status in the FCB and AL returns 00.

11 Search for the first entry. On entry, DS:DX point to an unopened FCB. The current disk directory is searched for the first matching filename (name could have "?"s indicating any letter matches) and if none are found, AL returns hex FF. Otherwise, AL returns 00 and the locations at the disk transfer address are set as follows:

If the FCB provided for searching was an extended FCB, then the first byte at the disk transfer address is set to hex FF, followed by five bytes of zeros, then the attribute byte from the search FCB, then the drive number used (1=A, 2=B, etc.), then the 32 bytes of the directory entry. Thus, the disk transfer address contains a valid unopened extended FCB with the same search attributes as the search FCB.

If the FCB provided for searching was a normal FCB, then the first byte is set to the drive number used (1=A, 2=B), and the next 32 bytes contain the matching directory entry. Thus, the disk transfer address contains a valid unopened normal FCB.

Notes:

If an extended FCB is used, the following search pattern is used:

1. If the FCB attribute byte is zero, only normal file entries are found. Entries for volume label, sub-directories, hidden and system files, will not be returned.

2. If the attribute field is set for hidden or system files, or directory entries, it is to be considered as an inclusive search. All normal file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).

3. If the attribute field is set for the volume label, it is considered an exclusive search, and *only* the volume label entry is returned.

The attribute bits are defined in the "DOS Disk Directory" section of Appendix C.

12 Search for the next entry. After function 11 has been called and found a match, function 12 may be called to find the next match to an ambiguous request (?s in the search filename). Both inputs and outputs are the same as function 11. The reserved area of the FCB keeps information necessary for continuing the search, so no disk operations may be performed with this FCB between a previous function 11 or 12 call and this one.

13 Delete file. On entry, DS:DX point to an unopened FCB. All matching current directory entries are deleted. If no directory entries match, AL returns hex FF, otherwise AL returns 00.

14 Sequential read. On entry, DS:DX point to an opened FCB. The record addressed by the current block (FCB bytes C-D) and the current record (FCB byte 1F) is loaded at the disk transfer address, then the record address is incremented. (The length of the record is determined by the FCB record size field.) If end-of-file is encountered, AL returns either 01 or 03. A return of 01 indicates no data in the record; 03 indicates a partial record is read and filled out with zeros. A return of 02 means there was not enough space in the disk transfer segment to read one record, so the transfer was ended. AL returns 00 if the transfer was completed successfully.

15 Sequential write. On entry, DS:DX point to an opened FCB. The record addressed by the current block and current record fields (size determined by the FCB record size field) is written from the disk transfer address (or, in the case of records less than sector sizes, is buffered up for an eventual write when a sector's worth of data is accumulated). The record address is then incremented. If the diskette is full, AL returns 01. A return of 02 means there was not enough space in the disk transfer segment to write one record, so the transfer was ended. AL returns 00 if the transfer was completed successfully.

16 Create file. On entry, DS:DX point to an unopened FCB. The current disk directory is searched for a matching entry, and if found, it is re-used. If no match was found, the directory is searched for an empty entry, and AL returns FF if none is found. Otherwise, the entry is initialized to a zero-length file, the file is opened (see function F), and AL returns 00.

The file may be marked *hidden* during its creation by using an extended FCB containing the appropriate attribute byte.

17 Rename file. On entry, DS:DX point to a modified FCB which has a drive code and a file name in the usual position, and a second file name starting 6 bytes after the first (DS:DX+hex 11) in what is normally a reserved area. Every matching occurrence of the first name in the current directory is changed to the second (with the restriction that two files cannot have the same name and extension). If "?"s appear in the second name, then the corresponding positions in the original name will be unchanged. AL returns FF if no match was found or if an attempt was made to rename to a filename that already existed, otherwise 00.

18 Used internally by DOS.

19 Current disk. AL returns with the code of the current default drive (0=A, 1=B, etc.).

1A Set disk transfer address. The disk transfer address is set to DS:DX. DOS does not allow disk transfers to wrap around within the segment, or overflow into the next segment.

1B Allocation table information. On return, DS:BX point to a byte containing the FAT identification byte for the default drive, DX has the number of allocation units, AL has the number of sectors per allocation unit, and CX has the size of the physical sector.

> Note: Beginning with DOS Version 2.00, this call no longer returns the address of a complete File Allocation Table, because the FAT's are no longer kept resident in memory.

1C Allocation table information for specific drive. This call is identical to call hex 1B except that on entry, DL contains the number of the drive from which the information should be gotten (0 = default, 1= A, etc.).

1D Used internally by DOS.

1E Used internally by DOS.

1F Used internally by DOS.

20 Used internally by DOS.

21 Random read. On entry, DS:DX point to an opened FCB. The current block and current record fields are set to agree with the random record field, then the record addressed by these fields is read into memory at the current disk transfer address. If end-of-file is encountered, AL returns either 01 or 03. If 01 is returned, no more data is available. If 03 is returned, a partial record is available filled out with zeros. A return of 02 means there was not enough space in the disk transfer segment to read one record, so the transfer was ended. AL returns 00 if the transfer was completed successfully.

22 Random write. On entry, DS:DX point to an opened FCB. The current block and current record fields are set to agree with the random record field, then the record addressed by these fields is written (or in the case of records not the same as sector sizes - buffered) from the disk transfer address. If the diskette is full AL returns 01. A return of 02 means there was not enough space in the disk transfer segment to write one record; so, the transfer was ended. AL returns 00 if the transfer was completed successfully.

23 File size. On entry, DS:DX point to an unopened FCB. The diskette directory is searched for the first matching entry and if none is found, AL returns FF. Otherwise, the random record field is set to the number of records in the file (in terms of the record size field rounded up) and AL returns 00.

> **Note:** Be sure to set the FCB record size field before using this function call; otherwise, erroneous information will be returned.

24 Set random record field. On entry, DS:DX point to an opened FCB. This function sets the random record field to the same file address as the current block and record fields.

25 Set interrupt vector. The interrupt vector table for the interrupt type specified in AL is set to the 4-byte address contained in DS:DX. Note that the original contents of the interrupt vector can be obtained through call hex 35.

26 Create a new program segment. On entry, DX has a segment number at which to set up a new program segment. The entire hex 100 area at location zero in the current program segment is copied into location zero in the new program segment. The memory size information at location 6 in the new segment is updated and the current termination, Ctrl-Break exit and critical error addresses from interrupt vector table entries for interrupt types 22, 23, and 24 are saved in the new program segment starting at hex 0A. They are restored from this area when the program terminates.

> **Note:** Use of this call should be avoided, now that DOS contains the EXEC function call (hex 4B).

27 Random block read. On entry, DS:DX point to an opened FCB, and CX contains a record count that must not be zero. The specified number of records (in terms of the record size field) are read from the file address specified by the random record field into the disk transfer address. If end-of-file is reached before all records have been read, AL returns either 01 or 03. A return of 01 indicates end-of-file and the last record is complete. A return of 03 indicates the last record is a partial record. If wrap-around above address hex FFFF in the disk transfer segment would have occurred, as many records as possible are read and AL returns 02. If all records are read successfully, AL returns 00. In any case, CX returns with the actual number of records read, and the random record field and the current block/record fields are set to address the next record (the first record not read).

28 Random block write. Essentially the same as function 27 above, except for writing and a write-protect check. If there is insufficient space on the disk, AL returns 01 and no records are written. If CX is zero upon entry, no records are written, but the file is set to the length specified by the random record field, whether longer or shorter than the current file size. (Allocation units are released or allocated as appropriate.)

29 Parse filename. On entry, DS:SI point to a command line to parse, and ES:DI point to a portion of memory to be filled with an unopened FCB. The contents of AL are used to determine the action to take, as shown below:

```
      <ignored>
bit:  7  6  5  4  3  2  1  0
```

If bit 0 = 1, then leading separators are scanned off the command line at DS:SI. Otherwise, no scan-off of leading separators takes place.

If bit 1 = 1, then the drive ID byte in the result FCB will be set (changed) *only* if a drive was specified in the command line being parsed.

If bit 2 = 1, then the filename in the FCB will be changed only if the command line contains a filename.

If bit 3 = 1, then the filename extension in the FCB will be changed only if the command line contains a filename extension.

Filename separators include the following characters: . ; , = + plus TAB and SPACE. Filename terminators include all of these characters plus \, <, >, | , /, ", [,], and any control characters.

The command line is parsed for a filename of the form d:filename.ext, and if found, a corresponding unopened FCB is created at ES:DI. If no drive specifier is present, the default drive is assumed. If no extension is present, it is assumed to be all blanks. If the character * appears in the filename or extension, then it and all remaining characters in the name or extension are set to ?.

If either ? or * appears in the filename or extension, AL returns 01; if the drive specifier is invalid AL returns FF; otherwise 00.

DS:SI will return pointing to the first character after the filename and ES:DI will point to the first byte of the formatted FCB. If no valid filename is present, ES:DI+1 will contain a blank.

> **Note:** This call is not useful for command lines containing path names.

2A Get date. Returns date in CX:DX. CX has the year (1980-2099 in binary), DH has the month (1-Jan, 2-Feb, etc.) and DL has the day. If the time-of-day clock rolls over to the next day, the date is adjusted accordingly, taking into account the number of days in each month and leap years.

2B Set date. On entry, CX:DX must have a valid date in the same format as returned by function 2A, above. If the date is indeed valid and the set operation is successful, AL returns 00. If the date is not valid, AL returns FF.

2C Get time. Returns with time-of-day in CX:DX. Time is actually represented as four 8-bit binary quantities as follows. CH has the hours (0-23), CL has minutes (0-59), DH has seconds (0-59), DL has 1/100 seconds (0-99). This format is readily converted to a printable form yet can also be used for calculations, such as subtracting one time value from another.

2D Set time. On entry, CX:DX has time in the same format as returned by function 2C, above. If any component of the time is not valid, the set operation is aborted and AL returns FF. If the time is valid, AL returns 00.

2E Set/reset verify switch. On entry, DL must contain 0, and AL must contain 1 to turn verify on, or 0 to turn verify off. When on, DOS will perform a verify operation each time it performs a diskette write to assure proper data recording. Although disk recording errors are very rare, this function has been provided for those user applications in which you may wish to verify the proper recording of critical data. Note that the current setting of the verify switch can be obtained through call hex 54.

2F Get DTA. On return, ES:BX contains the current DTA transfer address.

30 Get DOS version number. On return, AL contains the major version number. AH contains the minor version number.

> Note: If AL returns zero, it can be assumed that it is a pre-DOS Version 2.00 system.

31 Terminates process and remain resident (KEEP process). On entry, AL contains a binary exit code. DX contains the memory size value in paragraphs. This function call terminates the current process and attempts to set the initial allocation block to the number of paragraphs in DX. It will not free up any other allocation blocks belonging to that process. The exit code passed in AL is retrievable by the parent through Wait (function call hex 4D0) and can be tested through the ERRORLEVEL batch subcommands.

32 Used internally by DOS.

33 Ctrl-Break check. On entry, AL contains 00 to request the current state of control-break checking, 01 to set the state. If setting the state, DL must contain 00 for OFF or 01 for ON. DL returns the current state (00 = OFF, 01 = ON).

34 Used internally by DOS.

35 Get vector. On entry, AL contains a hexadecimal interrupt number. The CS:IP interrupt vector for the specified interrupt is returned in ES:BX. Note that interrupt vectors can be set through call hex 25.

36 Get disk free space. On entry, DL contains a drive: 0 = default, 1 = A, etc. On return, AX returns FFFF if the drive number was invalid. Otherwise, BX contains the number of available allocation units (clusters), DX contains the total number of clusters on the drive, CX contains the number of bytes per sector, and AX contains the number of sectors per cluster.

> **Note:** This call returns the same information in the same registers (except for the FAT pointer) as the get FAT pointer call (hex 1B) did in previous versions of DOS.

37 Used internally by DOS.

38 Return country dependent information (international). On entry, DS:DX points to a 32-byte block of memory in which returned information is passed and AL contains a function code. In DOS 2.00, this function code must be zero. The following information is pertinent to international applications:

WORD Date/time format
BYTE ASCIIZ string currency symbol followed by byte of zeros
BYTE ASCIIZ string thousands separator followed by byte of zeros
BYTE ASCIIZ string decimal separator followed by byte of zeros
24 bytes Reserved

The date and time format has the following values and meaning:

0 = USA standard *h:m:s m/d/y*

1 = Europe standard *h:m:s d/m/y*

2 = Japan standard *h:m:s d:m:y*

39 Create a sub-directory (MKDIR). On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. If any member of the directory path does not exist, then the directory path is not changed. On return, a new directory is created at the end of the specified path. Error returns are 3 and 5 (refer to error return table).

3A Remove a directory entry (RMDIR). On entry, DS:DX contains the address of an ASCIIZ string with the drive and directory path names. The specified directory is removed from the structure. The current directory cannot be removed. Error returns are 3 and 5 (refer to error return table). Note that code 5 is returned if the specified directory is not empty.

3B Change the current directory (CHDIR). On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. If any member of the directory path does not exist, then the directory path is not changed. Otherwise, the current directory is set to the ASCIIZ string. Error return is 3 (refer to the error return table).

3C Create a file (CREAT). On entry, DS:DX contains the address of an ASCIIZ string with the drive, path, and filename. CX contains the attribute of the file. This function call creates a new file or truncates an old file to zero length in preparation for writing. If the file did not exist, then the file is created in the appropriate directory and the file is given the read/write access code. The file is opened for read/write, and the handle is returned in AX. Error returns are 3, 4, and 5 (refer to the error return table). If an error code of 5 is returned, either the directory was full or a file by the same name exists and is marked read-only. Note that the change mode function call (hex 43) can later be used to change the file's attribute.

3D Open a file. On entry, DS:DX contains the address of an ASCIIZ string with the drive, path, and filenames. AL contains the access code. On return, AX contains an error code or a 16-bit file handle associated with the file. The following values are allowed for the access code:

0 = file is opened for reading.

1 = file is opened for writing.

2 = file is opened for both reading and writing.

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte (the read/write pointer can be changed through function call hex 42). The returned file handle must be used for subsequent input and output to the file. The file's date and time can be obtained or set through call hex 57, and its attribute can be obtained through call hex 43. Error returns are 2, 4, 5, and 12 (refer to the error return table).

Note: This call will open any normal or hidden file whose name matches the name specified.

3E Close a file handle. On entry, BX contains the file handle that was returned by "open." On return, the file will be closed and all internal buffers are flushed. Error return is 6 (refer to the error return table).

3F Read from a file or device. On entry, BX contains the file handle. CX contains the number of bytes to read. DS:DX contains the buffer address. On return, AX contains the number of bytes read. If the value is zero, then the program has tried to read from the end of file. This function call transfers (CX) bytes from a file into a buffer location. It is not guaranteed that all bytes will be read. For example, reading from the keyboard will read at most one line of text. If this read is performed from the standard input device, the input can be redirected (see "Redirection of Standard Input and Output" in Chapter 10). Error returns are 5 and 6 (refer to the error return table).

40 Write to a file or device. On entry, BX contains the file handle. CX contains the number of bytes to write. DS:DX contains the address of the data to write. Write transfers (CX) bytes from a buffer into a file. AX returns the number of bytes actually written. If this value is not the same as the number requested, it should be considered an error (no error code is returned, but your program can compare these values). The usual reason for this is a full disk. If this write is performed to the standard output device, the output can be redirected (see "Redirection of Standard Input and Output" in Chapter 10). Error returns are 5 and 6 (refer to the error return table).

41 Delete a file from a specified directory (Unlink). On entry, DS:DX contains the address of an ASCIIZ string with a drive, path, and filename. Global filename characters are not allowed in any part of the string. This function call removes a directory entry associated with a filename. Read-only files cannot be deleted by this call. To delete one of these files, you can first use call hex 43 to change the file's attribute to 0, then delete the file. Error returns are 2 and 5 (refer to the error return table).

42 Move file read/write pointer (Lseek). On entry, AL contains a method value. BX contains the file handle. CX:DX contains the desired offset in bytes (CX contains the most significant part). On return, DX:AX contains the new location of the pointer (DX contains the most significant part).

It moves the read/write pointer according to the following methods:

AL 0 = The pointer is moved to offset (CX:DX) bytes from the beginning of the file.

AL 1 = The pointer is moved to the current location plus offset.

AL 2 = The pointer is moved to the end-of-file plus offset. This method can be used to determine file's size.

Error returns are 1 and 6 (refer to the error return table).

43 Change file mode (CHMOD). On entry, AL contains a function code, and DS:DX contains the address of an ASCIIZ string with the drive, path, and filename. If AL contains 01 then the file will be set to the attribute in CX. (See "DOS Disk Directory" in Appendix C for the attribute byte description.) If AL is 0 then the file's current attribute will be returned in CX. Error returns are 2, 3, and 5 (refer to the error return table).

44 I/O control for devices (IOCTL). On entry, AL contains the function value. BX contains the file handle. On return, AX contains the number of bytes transferred for functions 2, 3, 4, and 5 or status (00 = not ready, FF = ready) for functions 6 and 7, or an error code. Use IOCTL to Set or Get device information associated with open device handle, or send/receive control strings to the device handle. The following function values are allowed in AL:

0 = Get device information (returned in DX).

1 = Set device information (determined by DX). Currently, DH must be zero for this call.

2 = Read CX number of bytes into DS:DX from device control channel.

3 = Write CX number of bytes from DS:DX to device control channel.

4 = Same as 2, but use drive number in BL (0 = default, 1 = A, etc.).

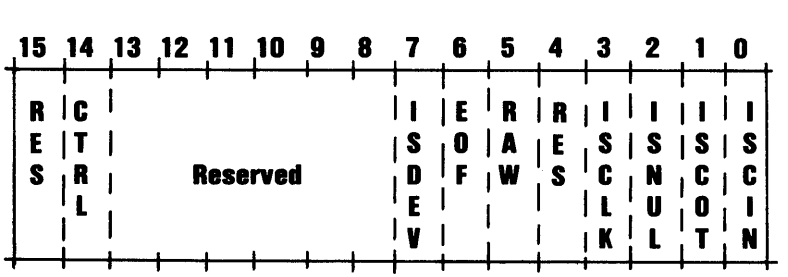
5 = Same as 3, but use drive number in BL (0 = default, 1 = A, etc.).

6 = Get input status.

7 = Get output status.

IOCTL can be used to get information about device channels. You can make calls on regular files, but only function values 0, 6, and 7 are defined in that case. All other calls return an "invalid function" error.

Calls AL=0 and AL=1. The bits of DX are defined as follows:



ISDEV = 1 if this channel is a device.
0 if this channel is a disk file (bits 8-15 = 0 in this case).

If ISDEV = 1

EOF = 0 if end-of-file on input.

BIN = 1 if operating in binary mode (no checks for Ctrl-Z).
= 0 if operating in ASCII mode (checking for Ctrl-Z as end-of-file).

ISCLK = 1 if this device is the clock device.

ISNUL = 1 if this device is the null device.

ISCOT = 1 if this device is the console output.

ISCIN = 1 if this device is the console input.

CTRL = 0 if this device cannot process control strings via calls AL=2 and AL=3.

CTRL = 1 if this device can process control strings via calls AL=2 and AL=3. Note that this bit can not be set by function call hex 44.

If ISDEV = 0

EOF = 0 if channel has been written. Bits 0-5 are the block device number for the channel (0 = A, 1 = B, ...).

Bits 15, 8-13, 4 are reserved and should not be altered.

Note: DH must be zero for call AL=1.

Calls AL=2, AL=3, AL=4, AL=5. These four calls allow arbitrary control strings to be sent or received from a character device. The Call syntax is the same as the Read and Write calls, except for calls 4 and 5 which accept a drive number in BL instead of a handle in BX. An "invalid function" error is returned if the CTRL bit is zero. An "access-denied" code is returned by calls 4 and 5 if the drive is invalid. Error returns are 1, 6, and 13 (refer to the error return table).

Calls 6 and 7. These calls allow you to check if a file handle is ready for input or output. If used for a file, AL always returns FF until end-of-file is reached, then always returns 00 unless the current file position is changed through call hex 42. When used for a device, AL returns FF for ready or zero for not ready.

45 Duplicate a file handle (DUP). On entry, BX contains the file handle. On return, AX contains the returned file handle. This function call takes an opened file handle and returns a new file handle that refers to the same file at the same position. Error returns are 4 and 6 (refer to the error return table).

> **Note:** If you move the read/write pointer of either handle, the pointer for the other handle will also be changed.

46 Force a duplicate of a handle (DUP). On entry, BX contains the file handle. CX contains a second file handle. On return, the CX file handle will refer to the same stream as the BX file handle. If the CX file handle was an open file, then it is closed first. Error return is 6 (refer to the error return table).

> **Note:** If you move the read/write pointer of either handle, the pointer for the other handle will also be changed.

47 Get Current directory. On entry, DL contains a drive number (0 = default, 1 = A, etc.) and DS:SI point to a 64-byte area of user memory. The full path name (starting from the root directory) of the current directory for the specified drive is placed in the area pointed to by DS:SI. Note that the drive letter will not be part of the returned string. The string will not begin with a backslash and will be terminated by a byte containing hex 00. The error returned is 15.

48 Allocate memory. On entry, BX contains the number of paragraphs requested. On return, AX:0 points to the allocated memory block. If the allocation fails, BX will return the size of the largest block of memory available in paragraphs. Error returns are 7 and 8 (refer to the error return table).

49 Free allocated memory. On entry, ES contains the segment of the block being returned. On return, a block of memory is returned to the system pool that was allocated by call hex 48. Error returns are 7 and 9 (refer to the error return table).

4A SETBLOCK-Modify allocated memory blocks. On entry, ES contains the segment of the block. BX contains the new requested block size in paragraphs. DOS will attempt to "grow" or "shrink" the specified block. If the call fails on a grow request, then on return, BX contains the maximum block size possible. Error returns are 7, 8, and 9 (refer to the error return table).

4B Load or execute a program (EXEC). This function call allows a program to load another program into memory and (default) begin execution of it. DS:DX points to the ASCIIZ string with drive, path, and filename of the file to be loaded. ES:BX points to a parameter block for the load and AL contains a function value. The following function values are allowed:

0 = Load and execute the program. A program segment prefix is established for the program and the terminate and control-break addresses are set to the instruction after the EXEC system call.

Note: When control is returned, all registers are changed including the stack. You must restore SS, SP and any other required registers before proceeding.

3 = Load, do not create the program segment prefix, and do not begin execution. This is useful in loading program overlays.

For each of these values, the block pointed to by ES:BX has the following format:

AL = 0 Load/execute program

WORD segment address of environment string to be passed
DWORD pointer to command line to be placed at PSP+ 80h
DWORD points to default FCB to be passed at PSP+ 5Ch
DWORD pointer to default FCB to be passed at PSP+ 6Ch

AL = 3 Load overlay

WORD segment address where file will be loaded
WORD relocation factor to be applied to the image

Note that all open files of a process are duplicated in the newly created process after an EXEC. This is extremely powerful; the parent process has control over the meanings of standard input, output, auxiliary, and printer devices. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output, and then execute a sort program that takes its input from standard input and writes to standard output.

Also inherited (or copied from the parent) is an "environment." This is a block of text strings (less than 32K bytes total) that convey various configuration parameters. The following is the format of the environment (always on a paragraph boundary):

Byte ASCIIZ string 1
Byte ASCIIZ string 2
...
Byte ASCIIZ string n
Byte of zero

Typically the environment strings have the form:

parameter=value

For example, the string VERIFY=ON could be passed. A zero value of the environment address will cause the newly created process to inherit the parent's environment unchanged. The segment address of the environment is placed at offset hex 2C of the Program Segment Prefix for the program being invoked. Error returns are 1, 2, 5, 8, 10, and 11 (refer to the error return table).

Notes:

1. When your program received control, all of available memory was allocated to it. You must free some memory (see call hex 4A) before EXEC can load the program you are invoking. Normally, you would shrink down to the minimum amount of memory you need, and free the rest.

2. The EXEC call uses the loader portion of COMMAND.COM (at the high end of memory) to perform the loading. If your program has overlaid the loader, this call will attempt to re-load the loader, thus destroying the contents of the last 1536 bytes of memory. If you have used the "Allocate Memory" call to allocate all of memory and the loader has been overlaid, the EXEC call will return an error due to insufficient memory to load the loader.

4C Terminate a process (Exit). On entry, AL contains a binary return code. This function call will terminate the current process, transferring control to the invoking process. In addition, a return code can be sent. The return code can be interrogated by the batch subcommands IF and ERRORLEVEL and by the wait function call (4D). All files open at the time are closed.

4D Retrieve the return code of a sub-process (Wait). This function call returns the Exit code specified by another process (via call hex 4C or call hex 31) in AX. It returns the Exit code only once. The low byte of this code is that sent by the exiting routine. The high byte is zero for normal termination, 01 if terminated by Crtl-Break, 02 if terminated as the result of a critical device error, or 03 if terminated by function call hex 31.

4E Find first matching file (FIND FIRST). On input, DS:DX points to an ASCIIZ string containing the drive, path, and filename of the file to be found. The filename portion can contain global filename characters. CX contains the attribute to be used in searching for the file. See function call hex 11 for a description of how the attribute bits are used for searches. If a file is found that matches the specified drive, path, and filename and attribute, the current DTA will be filled in as follows:

21 bytes - reserved for DOS use on subsequent find next calls

1 byte - attribute found

2 bytes - file's time

2 bytes - file's date

2 bytes - low word of file size

2 bytes - high word of file size

13 bytes - name and extension of file found, followed by a byte of zeros. All blanks are removed from the name and extension, and if an extension is present, it is preceded by a period. Thus, the name returned appears just as you would enter it as a command parameter. Such as, TREE.COM followed by a byte of zeros. Error returns are 2 and 18 (refer to the error return table).

4F Find next matching file. On input, the current DTA must contain the information that was filled in by a previous Find First call (hex 4E). No other input is required. This call will find the next directory entry matching the name that was specified on the previous Find First call. If a matching file is found, the current DTA will be set as described in call hex 4E. If no more matching files are found, error code 18 is returned (refer to the error return table).

50 Used internally by DOS.

51 Used internally by DOS.

52 Used internally by DOS.

53 Used internally by DOS.

54 Get verify state. On return, AL returns 00 if verify is OFF, 01 if verify is ON. Note that the verify switch can be set through call hex 2E.

55 Used internally by DOS.

56 Rename a file. On input, DS:DX points to an ASCIIZ string containing the drive, path, and filename of the file to be renamed. ES:DI points to an ASCIIZ string containing the path and filename to which the file is to be renamed. If a drive is used in this string, it must be the same as the drive specified or implied in the first string. The directory paths need not be the same, allowing a file to be moved to another directory and renamed in the process. Error returns are 3, 5, and 17 (refer to the error return table).

57 Get/Set a file's date and time. On input, AL contains 00 or 01. BX contains a file handle. If AL=00 on entry, DX and CX will return the date and time from the handle's internal table, respectively. If AL=01 on entry, the handle's date and time will be set to the date and time in DX and CX, respectively. The date and time formats are the same as those for the directory entry described in Appendix C, except that when passed in registers, the bytes are reversed (that is, DH contains the *low* order portion of the date, etc.). Error returns are 1 and 6 (refer to the error return table).

Notes:

Appendix E. DOS Control Blocks and Work Areas

DOS Memory Map

0000:0000	Interrupt vector table
0040:0000	ROM communication area
0050:0000	DOS communication area
XXXX:0000	IBMBIO.COM — DOS interface to ROM I/O routines
XXXX:0000	IBMDOS.COM — DOS interrupt handlers, service routines (INT 21 functions)
	DOS buffers, control areas, and installed device drivers
XXXX:0000	Resident portion of COMMAND.COM — Interrupt handlers for interrupts hex 22 (terminate), hex 23 (Ctrl-Break), hex 24 (critical error), and code to reload the transient portion.
XXXX:0000	External command or utility — (.COM or .EXE file)
XXXX:0000	User stack for .COM files (256 bytes)
XXXX:0000	Transient portion of COMMAND.COM — Command interpreter, internal commands, batch processor, external command loader.

Notes:

1. Memory map addresses are in segment:offset format. For example, 0070:0000 is absolute address hex 0700.

2. The DOS Communication Area is used as follows:

0050:0000 Print screen status flag store

- **0** Print screen not active or successful print screen operation
- **1** Print screen in progress
- **255** Error encountered during print screen operation

0050:0001 Used by BASIC

0050:0004 Single-drive mode status byte

- **0** Diskette for drive A was last used
- **1** Diskette for drive B was last used

0050:0010 – 0021 Used by BASIC

0050:0022 – 002F Used by DOS for diskette initialization

0050:0030 – 0033 Used by MODE command

All other locations within the 256 bytes beginning at 0050:0000 are reserved for DOS use.

3. User memory is allocated from the lowest end of available memory that will satisfy the request for memory.

DOS Program Segment

When you enter an external command, or invoke a program through the EXEC function call, DOS determines the lowest available address to use as the start of available memory for the program being invoked. This area is called the Program Segment (it must not be moved).

At offset 0 within the Program Segment, DOS builds the Program Segment Prefix control block. (See below.) EXEC loads the program at offset hex 100 and gives it control.

The program returns from EXEC by a jump to offset 0 in the Program Segment Prefix, by issuing an INT 20, by issuing an INT 21 with register AH=0 or hex 4C, or by calling location hex 50 in the Program Segment Prefix with AH=0 or hex 4C.

> **Note:** It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating via any of these methods except call hex 4C.

All four methods result in transferring control to the resident portion of COMMAND.COM (function call hex 4C allows the terminating process to pass a return code). All of these methods result in turning to the program that issued the EXEC. During this returning process, interrupt vectors hex 22, hex 23, and hex 24 (terminate, Ctrl-Break, and critical error exit addresses) are restored from the values saved in the Program Segment Prefix of the terminating program. Control is then given to the terminate address. If this is a program returning to COMMAND, control transfers to its transient portion. If a batch file was in process, it is continued; otherwise, COMMAND issues the system prompt and waits for the next command to be entered from the keyboard.

When a program receives control, the following conditions are in effect:

For all programs:

- The segment address of the passed environment is contained at offset hex 2C in the Program Segment Prefix.

  The environment is a series of ASCII strings (totaling less than 32K) in the form:

  NAME=parameter

Each string is terminated by a byte of zeros, and the entire set of strings is terminated by another byte of zeros. The environment built by the command processor (and passed to all programs it invokes) will contain a COMSPEC= string at a minimum (the parameter on COMSPEC is the path used by DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings entered through the SET command (see Chapter 10).

The environment that you are passed is actually a copy of the invoking process environment. If your application uses a "terminate and stay resident" concept, you should be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

- Offset hex 50 in the Program Segment Prefix contains code to invoke the DOS function dispatcher. Thus, by placing the desired function number in AH, a program can issue a long call to PSP+ 50 to invoke a DOS function, rather than issuing an interrupt type hex 21.

- Disk transfer address (DTA) is set to hex 80 (default DTA in the Program Segment Prefix).

- File control blocks at hex 5C and hex 6C are formatted from the first two parameters entered when the command was invoked. Note that if either parameter contained a path name, then the corresponding FCB will contain only a valid drive number. The filename field will not be valid.

- An Unformatted parameter area at hex 81 contains all the characters entered after the command name (including leading and imbedded delimiters), with hex 80 set to the number of characters. If the <, >, or ¦ parameters were entered on the command line, they (and the filenames associated with them) will not appear in this area, because redirection of standard input and output is transparent to applications.

- Offset 6 (one word) contains the number of bytes available in the segment.

- Register AX reflects the validity of drive specifiers entered with the first two parameters as follows:

  — AL=FF if the first parameter contained an invalid drive specifier (otherwise AL=00)

  — AH=FF if the second parameter contained an invalid drive specifier (otherwise AH=00)

For .EXE programs:

- DS and ES registers are set to point to the Program Segment Prefix.

- CS, IP, SS, and SP registers are set to the values passed by the linker.

For .COM programs:

- All four segment registers contain the segment address of the initial allocation block, that starts with the Program Segment Prefix control block.

- All of user memory is allocated to the program. If the program wishes to invoke another program through the EXEC function call, it must first free some memory through the Setblock (hex 4A) function call, to provide space for the program being invoked.

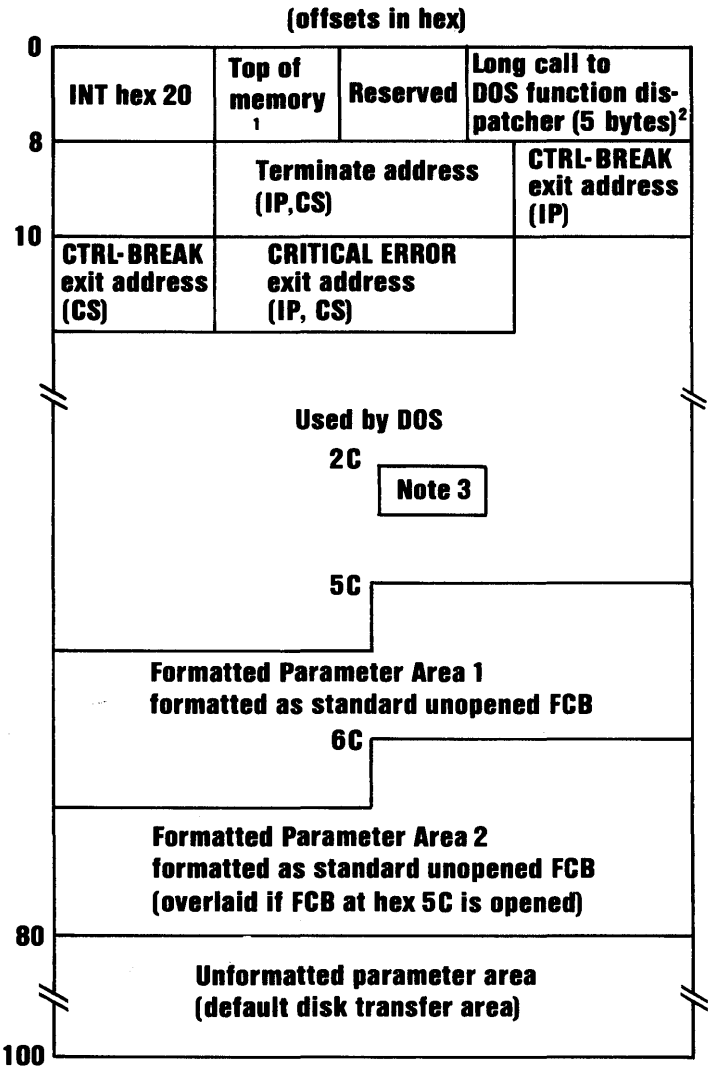
- The Instruction Pointer (IP) is set to hex 100.

- SP register is set to the end of the program's segment. The segment size at offset 6 is reduced by hex 100 to allow for a stack of that size.

- A word of zeros is placed on the top of the stack.

The Program Segment Prefix (with offsets in hexadecimal) is formatted as follows.

Program Segment Prefix

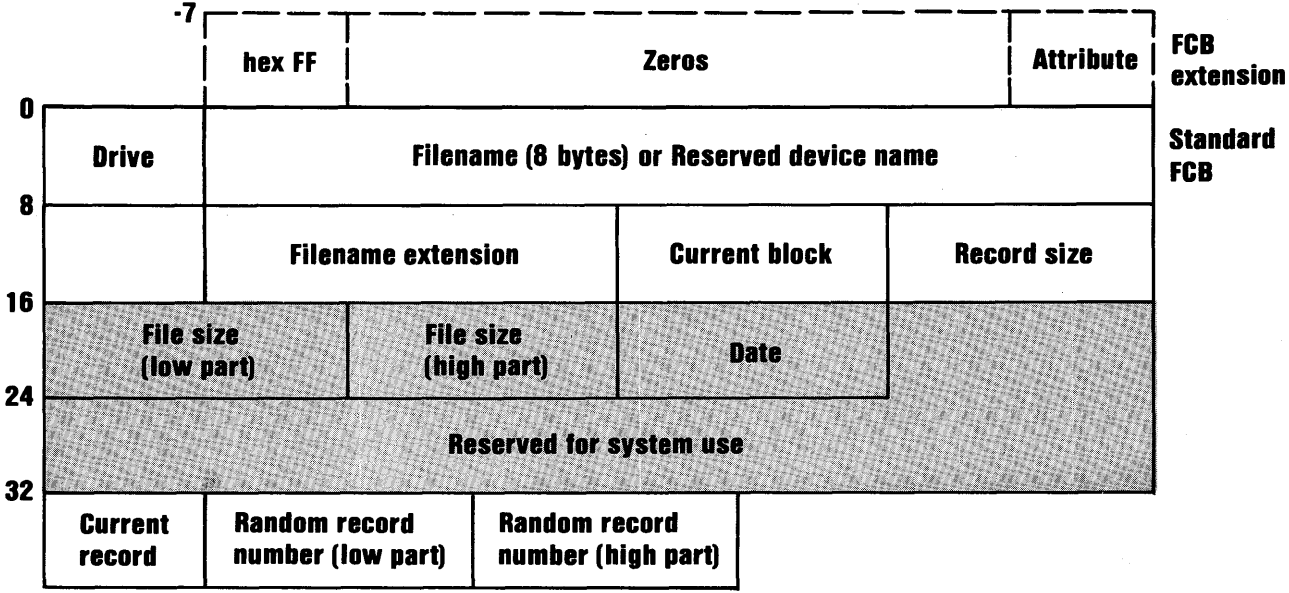


1. First segment of available memory is in segment (paragraph) form (for example, hex 1000 would represent 64K).

2. The word at offset 6 contains the number of bytes available in the segment.

3. Offset hex 2C contains the segment address of the environment.

4. Programs must not alter any part of the PSP below offset hex 5C.



Unshaded areas must be filled in by the using program.

Shaded areas are filled in by DOS and must not be modified.

Standard File Control Block

The standard file control block (FCB) is defined as follows, with the offsets in decimal:

Byte	Function
0	Drive number. For example, **Before open:** 0 - default drive 1 - drive A 2 - drive B etc. **After open:** 1 - drive A 2 - drive B etc. A 0 is replaced by the actual drive number during open.
1-8	Filename, left-justified with trailing blanks. If a reserved device name is placed here (such as LPT1), do not include the optional colon.
9-11	Filename extension, left-justified with trailing blanks (can be all blanks).
12-13	Current block number relative to the beginning of the file, starting with zero (set to zero by the open function call). A block consists of 128 records, each of the size specified in the logical record size field. The current block number is used with the current record field (below) for sequential reads and writes.

14-15 Logical record size in bytes. Set to hex 80 by the open function call. If this is not correct, you must set the value because DOS uses it to determine the proper locations in the file for all disk reads and writes.

16-19 File size in bytes. In this 2-word field, the first word is the low-order part of the size.

20-21 Date the file was created or last updated. The *mm/dd/yy* are mapped in the bits as follows:

<							21								>	<			20				>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
y	*y*	*y*	*y*	*y*	*y*	*y*	*m*	*m*	*m*	*m*	*d*	*d*	*d*	*d*	*d*								

where:

mm is 1-12
dd is 1-31
yy is 0-119 (1980-2099)

22-31 Reserved for system use.

32 Current relative record number (0-127) within the current block. (See above.) You must set this field before doing *sequential* read/write operations to the diskette. (This field is not initialized by the open function call.)

33-36 Relative record number relative to the beginning of the file, starting with zero. You must set this field before doing *random* read/write operations to the diskette. (This field is not initialized by the open function call.)

If the record size is less than 64 bytes, both words are used. Otherwise, only the first three bytes are used. Note that if you use the File Control Block at hex 5C in the program segment, the last byte of the FCB overlaps the first byte of the unformatted parameter area.

Notes:

1. An unopened FCB consists of the FCB prefix (if used), drive number, and filename/extensions properly filled in. An open FCB is one in which the remaining fields have been filled in by the Create or Open function calls.

2. Bytes 0-15 and 32-36 must be set by the user program. Bytes 16-31 are set by DOS and must not be changed by user programs.

3. All word fields are stored with the least significant byte first. For example, a record length of 128 is stored as hex 80 at offset 14, and hex 00 at offset 15.

Extended File Control Block

The extended File Control Block is used to create or search for files in the disk directory that have special attributes.

It adds a 7-byte prefix to the FCB, formatted as follows:

Byte	Function
FCB-7	Flag byte containing hex FF to indicate an extended FCB.
FCB-6 to FCB-2	Reserved.
FCB-1	Attribute byte. See "DOS Disk Directory" in Appendix C for attribute bit definitions. Also refer to function call hex 11 (search first) for details on using the attribute bits during directory searches. This function is present to allow applications to define their own files as *hidden* (and thereby exclude them from directory searches), and to allow selective directory searches.

Any references in the DOS Function Calls (refer to Appendix D) to an FCB, whether opened or unopened, may use either a normal or extended FCB. If using an extended FCB, the appropriate register should be set to the first byte of the prefix, rather than the drive-number field.

Appendix F. Executing Commands from Within an Application

Beginning with DOS Version 2.00, application programs may invoke a secondary copy of the command processor. Your program may pass a DOS command as a parameter, that the secondary command processor will execute as though it had been entered from the standard input device. The procedure is:

1. Assure that adequate free memory (17K) exists to contain the second copy of the command processor and the command it is to execute.

2. Build a parameter string for the secondary command processor in the form:

   1 byte = length of parameter string
   xx byte = parameter string
   1 byte = hex 0D (carriage return)

   For example, the assembly statement below would build the string to cause execution of a DISKCOPY command:

```
DB 17,"\C DISKCOPY A: B:", 13
```

3. Use the EXEC function call (hex 4B, function value 0) to cause execution of the secondary copy of the command processor (the drive, directory and name of the command processor can be gotten from the COMSPEC=parameter in the environment passed to you at PSP+ hex 2C). Remember to set offset 2 of the EXEC control block to point to the parameter string built above.

Refer to Chapter 10, "Advanced Commands" for additional information about invoking a second copy of the command processor.

Appendix G. Fixed Disk Information

The IBM Personal Computer Fixed Disk Support Architecture has been designed to meet the following objectives:

- Allow multiple operating systems to utilize the fixed disk without the need to dump/restore when changing operating systems.
- Allow a user-selected operating system to be started from the fixed disk.

Fixed Disk Architecture

The architecture is defined as follows:

- In order to *share* the fixed disk among operating systems, the disk may be logically divided into 1 to 4 "partitions." The space within a given partition is contiguous, and can be dedicated to a specific operating system. Each operating system may "own" only one partition. The number and sizes of the partitions is user-selectable through a fixed disk utility program. (The DOS utility is FDISK.COM.) The partition information is kept in a partition table that is imbedded in the master fixed disk boot record on the first sector of the disk.

- Any operating system must consider its partition to be an entire disk, and must ensure that its functions and utilities do not access other partitions on the disk.

- Each partition can contain a boot record on its first sector, and any other programs or data that you choose—including a copy of an operating system. For example, the DOS FORMAT command may be used to format (and place a copy of DOS in) the DOS partition, in the same manner that a diskette is formatted. With the FDISK utility, you may designate a partition as "bootable" (active)—the master fixed disk boot record will cause that partition's boot record to receive control when the system is started or restarted.

System Initialization

The System initialization (or system boot) sequence is as follows:

1. System initialization first attempts to load an operating system from diskette drive A. If the drive is not ready or a read error occurs, it then attempts to read a master fixed disk boot record from the first sector of the first fixed disk on the system. If unsuccessful, or if no fixed disk is present, it invokes ROM BASIC.

2. If successful, the master fixed disk boot record is given control and it examines the partition table imbedded within it. If one of the entries indicates a "bootable" (active) partition, its boot record is read (from the partition's first sector) and given control.

3. If none of the partitions is bootable, ROM BASIC is invoked.

4. If any of the boot indicators is invalid, or if more than one indicator is marked as bootable, the message **Invalid partition table** is displayed and the system enters an enabled loop. You may then insert a system diskette in drive A and use system reset to restart from diskette.

5. If the partition's boot record can't be successfully read within 5 retries due to read errors, the message **Error loading operating system** appears and the system enters an enabled loop.

6. If the partition's boot record does not contain a valid "signature" (see "Boot Record/Partition Table"), the message **Missing operating system** appears, and the system enters an enabled loop.

   Note: When changing the size or location of any partition, you must ensure that all existing data on the disk has been backed up (the partitioning process will "lose track" of the previous partition boundaries).

Boot Record/Partition Table

A fixed disk boot record must be written on the first sector of all fixed disks, and contains:

1. Code to load (and give control to) the boot record for 1 of 4 possible operating systems.

2. A partition table at the end of the boot record. Each table entry is 16 bytes long, and contains the starting and ending cylinder, sector and head for each of 4 possible partitions, as well as the number of sectors preceding the partition and the number of sectors occupied by the partition. The "boot indicator" byte is used by the boot record to determine if one of the partitions contains a loadable operating system. FDISK initialization utilities mark a user-selected partition as "bootable" by placing a value of hex 80 in the corresponding partition's boot indicator (setting all other partitions' indicators to zero at the same time.) The presence of the hex 80 tells the standard boot routine to load the sector whose location is contained in the following 3 bytes. That sector will be the actual boot record for the selected operating system, and it will be responsible for the remainder of the system's loading process (as it is from diskette). All boot records are loaded at absolute address 0:7C00.

The partition table (with its offsets into the boot record) is as follows:

Offs	Purpose	Head		Sector	Cylinder
1BE	Partition 1 begin	boot ind	H	S	CYL
1C2	Partition 1 end	syst ind	H	S	CYL
1C6	Partition 1 rel sect	Low word		High word	
1CA	Partition 1 # sects	Low word		High word	
1CE	Partition 2 begin	boot ind	H	S	CYL
1D2	Partition 2 end	syst ind	H	S	CYL
1D6	Partition 2 rel sect	Low word		High word	
1DA	Partition 2 # sects	Low word		High word	
1DE	Partition 3 begin	boot ind	H	S	CYL
1E2	Partition 3 end	syst ind	H	S	CYL
1E6	Partition 3 rel sect	Low word		High word	
1EA	Partition 3 # sects	Low word		High word	
1EE	Partition 4 begin	boot ind	H	S	CYL
1F2	Partition 4 end	syst ind	H	S	CYL
1F6	Partition 4 rel sect	Low word		High word	
1FA	Partition 4 # sects	Low word		High word	
1FE	Signature	hex 55	hex AA		

Technical Information

When shipped by IBM, the 10-megabyte fixed disks are formatted with 512-byte sectors at an interleave factor of 6 (17 sectors per track, 4 heads per cylinder). They contain no data or boot records.

The boot indicator byte must contain 0 for a non-bootable partition, or hex 80 for a bootable partition. Only one partition can be marked bootable.

The "syst ind" field contains an indicator of the operating system that "owns" the partition. Each operating system can "own" only one partition.

The system indicators are:

hex 00 - unknown (unspecified)
hex 01 - DOS

The 1-byte fields labelled "CYL" contain the low-order 8 bits of the cylinder number—the high order 2 bits are in the high order 2 bits of the "S" (sector) field. This corresponds with ROM BIOS Interrupt hex 13 (disk I/O) requirements, to allow for a 10-bit cylinder number.

The fields are ordered in such a manner that only 2 MOV instructions are required to properly set up the DX and CX registers for a ROM BIOS call to load the appropriate boot record (fixed disk booting is only possible from the first fixed disk on a system, whose BIOS drive number (hex 80) corresponds to the boot indicator byte.)

All partitions are allocated in cylinder multiples and begin on sector 1, head 0. EXCEPTION: the partition which is allocated at the beginning of the disk starts at sector 2, to account for the disk's master boot record.

The number of sectors preceding each partition on the disk is kept in the 4-byte field labelled "rel sect." This value is obtained by counting the sectors beginning with cylinder 0, sector 1, head 0 of the disk, and incrementing the sector, head and then track values up to the beginning of the partition. Thus, if the disk has 17 sectors per track and 4 heads, and the second partition begins at cylinder 1, sector 1, head 0, the partition's starting relative sector is 68 (decimal)—there were 17 sectors on each of 4 heads on 1 track allocated ahead of it. The field is stored with the least significant word first.

The number of sectors allocated to the partition is kept in the"# of sects" field. This is a 4-byte field stored least significant word first.

The last 2 bytes of the boot record (hex 55AA) are used as a signature to identify a valid boot record. Both this record and the partition boot records are required to contain the signature at offset hex 1FE.

The Master disk boot record will invoke ROM BASIC if no indicator byte reflects a "bootable" system.

When a partition's boot record is given control, it is passed its partition table entry address in the DS:SI registers.

System programmers designing a utility to initialize/manage a fixed disk must provide the following functions at a minimum:

1. Write the master disk boot record/partition table to the disk's first sector to initialize it.

2. Perform partitioning of the disk—that is, create or update partition table information (all fields for the partition) when the user wishes to create a partition. This may be limited to creating a partition for only one type of operating system, but must allow repartitioning the entire disk, or adding a partition without interfering with existing partitions (user's choice).

3. Provide a means for marking a user-specified partition as bootable, and resetting the bootable indicator bytes for all other partitions at the same time.

Appendix H. EXE File Structure and Loading

The .EXE files produced by the Linker program consist of two parts:

- Control and relocation information
- The load module itself

The control and relocation information, which is described below, is at the beginning of the file in an area known as the *header*. The load module immediately follows the header. The load module begins on a sector boundary and is the memory image of the module constructed by the linker.

The header is formatted as follows:

Hex Offset	Contents
00-01	Hex 4D, hex 5A—This is the LINK program's *signature* to mark the file as a valid .EXE file.
02-03	Length of image mod 512 (remainder after dividing the load module image size by 512).
04-05	Size of the file in 512-byte increments (*pages*), including the header.
06-07	Number of relocation table items that follow the formatted portion of the header.
08-09	Size of the header in 16-byte increments (*paragraphs*). This is used to locate the beginning of the load module in the file.

Hex Offset	Contents
0A-0B	Minimum number of 16-byte paragraphs required above the end of the loaded program.
0C-0D	Maximum number of 16-byte paragraphs required above the end of the loaded program.
0E-0F	Offset of stack segment in load module (in segment form).
10-11	Value to be in the SP register when the module is given control.
12-13	Word checksum - negative sum of all the words in the file, ignoring overflow.
14-15	Value to be in the IP register when the module is given control.
16-17	Offset of code segment within load module (in segment form).
18-19	Offset of the first relocation item within the file.
1A-1B	Overlay number (0 for resident part of the program).

The relocation table follows the formatted area just described. The relocation table is made up of a variable number of relocation items. The number of items is contained at offset 06-07. The relocation item contains two fields—a 2-byte offset value, followed by a 2-byte segment value. These two fields contain the offset into the load module of a word which requires modification before the module is given control. This process is called *relocation* and is accomplished as follows:

1. A Program Segment Prefix is built following the resident portion of the program that is performing the load operation.

2. The formatted part of the header is read into memory (it's size is at offset 08-09).

3. The load module size is determined by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. The actual size is downward adjusted based on the contents of offsets 02-03. Note that all files created by pre-release 1.10 LINK programs *always* placed a value of 4 at that location, regardless of actual program size. Therefore, we recommend that this field be ignored if it contains a value of 4. Based on the setting of the high/low loader switch, an appropriate segment is determined at which to load the load module. This segment is called the *start segment*.

4. The load module is read into memory beginning at the start segment.

5. The relocation table items are read into a work area (one or more at a time).

6. Each relocation table item segment value is added to the start segment value. This calculated segment, in conjunction with the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.

7. Once all relocation items have been processed, the SS and SP registers are set from the values in the header and the start segment value is added to SS. The ES and DS registers are set to the segment address of the Program Segment Prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is used to give the module control.

Appendix I. Running Compilers and Assemblers

Using Compilers and Assemblers With Fixed Disk

The following is a summary of how to run the IBM Personal Computer compilers and the IBM Personal Computer Macro Assembler with the Disk Operating System (DOS) Version 2.00. For the purposes of this appendix, the word *compile* will also refer to assemble.

1. Make sure to back up your original language diskettes using the techniques described within this book and/or the respective language books.

2. Make sure all source code you will compile is in the current directory of your disk.

3. Most of the language processors may be located in any directory at compile time (see exception list at the end of this appendix).

4. When Compiling you should use the command lines as described in your language book. You can modify the drive specifier in the command line to indicate the location of the particular language processor.

5. When Linking, use the IBM Personal Computer Linker, Version 2.00, provided with DOS Version 2.00 diskette. The instructions within this book and your language book will provide the necessary information for linking.

6. When running a program (.BAT, .COM, or a .EXE file) it is not always necessary for that program to be in the current directory (see the PATH command in Chapter 6 of this book). However, if the program requires another file at runtime; such as a data file or a Common Runtime Module, then those files must be in the current directory at runtime.

7. It is possible for COMMAND.COM to be overwritten in memory. Therefore, it is usually helpful to have a copy of COMMAND.COM in the root directory of the drive from which DOS was started.

Exceptions

- All IBM Personal Computer Language Products:
  - All files accessed by your program must be in the current directory at runtime.
  - Any Common Runtime Module used by your program must be in the current directory at runtime.
- IBM Personal Computer BASIC Compiler:
  - BASRUN.EXE, if used at runtime, must be in the current directory.
- IBM Personal Computer COBOL Compiler:
  - COBOL.COM and its overlays must be in the current directory at compile time.
  - To compile, you must use the command line and indicate the drive location of the overlays with the /C parameter.
  - COBRUN.EXE must be in the current directory at runtime.
- IBM Personal Computer Pascal Compiler:
  - Pascal requires a hex update (see "Pascal Hex Patch" in Appendix J) in order to run this compiler from a drive other than drive A.
  - PASKEY must be in the current directory at compile time.

Notes:

Appendix J. Running the Pascal Compiler

Using Pascal Hex Patch With Fixed Disk

The IBM Personal Computer Pascal Compiler 1.00 requires that a special file called PASKEY be in the current directory on diskette drive A. The following hex update can be used so that the Pascal compiler will look on the default drive for this PASKEY file.

The following describes how to update your PAS1 diskette. Follow the instructions carefully. If you make a mistake don't panic. Just start again.

To update your PAS1 diskette, you need a blank formatted diskette. Put your DOS diskette in diskette drive A and the blank formatted diskette in diskette drive B. You must use the DOS DISKCOPY command to make an exact copy of your PAS1 diskette.

After the DISKCOPY program is started and you see the message **Strike any key when ready**, insert the ORIGINAL copy of the Pascal PAS1 diskette in diskette drive A. It must be the diskette from your IBM Personal Computer Pascal Compiler package or an exact copy of it. Anything else will not work. After you insert the diskettes, press the Spacebar to start the disk copy:

```
A>DISKCOPY A: B:

Insert source diskette in drive A:

Insert target diskette in drive B:

Strike any key when ready


Copy another (Y/N)?N
```

Place your DOS diskette back in diskette drive B. You now have an exact copy of your Pascal PAS1 diskette in diskette drive B. You can now apply the update to the PAS1 diskette with the DOS DEBUG program on the DOS diskette.

The DEBUG program prompt is "-." All the things that you type are after the "-."

When you are responding to the "-" prompt, you must press the Enter key. Note that the *xxxx* in the data displayed line will be filled with the appropriate memory addresses:

```
A>DEBUG
-LDS: 100 1 A5 1
-DDS: 177 LA
xxxx:0177  41-3A 50 41 53 4B 45 59 20      A:PASKEY
xxxx:0180 20
```

If you do not see the above line of data after the "DDS:177 LA" command then you did something wrong and you should start again. You can start again by typing Q followed by Enter. This returns you to DOS:

```
-EDS: 177 'PASKEY '
-WDS: 100 1 A5 1
-Q
```

(Remember to enter the two blanks after PASKEY.)

You are now back in DOS; you should now recreate your working PAS1 diskette from the updated diskette and mark both diskettes so you know which are the updated ones. The PAS1.EXE file on the updated PAS1 diskette will now look for the PASKEY file in the current directory on the default drive.

Notes:

Appendix K. Considerations for Using Applications

If you have any of the following applications, please refer to the appropriate section in this appendix for additional information about using these applications with DOS 2.00:

- Accounting Packages by BPI Systems, Inc.
- Accounting Packages Version 1.00 by Peachtree Software, Inc.
- Accounting Packages Version 1.10 by Peachtree Software, Inc.
- Arithmetic Games 1 and 2
- Asynchronous Communications Support Version 1.00
- Asynchronous Communications Support Version 2.00
- EasyWriter Version 1.10
- Fact Track
- PFS:File
- PFS:Report

- SNA 3270 Emulation and RJE Support Version 1.00
- The Dow Jones Reporter Version 1.00
- Typing Tutor
- VisiCalc Version 1.10
- 3101 Emulator Version 1.00

Accounting Packages by BPI Systems, Inc.

These packages are designed for, and should be used only with DOS 1.00 or DOS 1.10. You should not use these applications with DOS 2.00.

Accounting Packages Version 1.00 by Peachtree Software, Inc.

These packages are designed for, and should be used only with DOS 1.00 or DOS 1.10. You should not use these applications with DOS 2.00.

Accounting Packages Version 1.10 by Peachtree Software, Inc.

These packages are designed to be used with either DOS 1.10 or DOS 2.00. If you use these packages with DOS 2.00 on a diskette-only system, you must have 128K bytes memory and 320K byte diskette drives in order to accommodate the larger size of DOS 2.00 and BASIC 2.00.

The Accounting Packages Version 1.10 by Peachtree Software, Inc. include functions allowing both programs and data files to reside on the IBM Personal Computer fixed disk. When using the fixed disk, these packages require DOS2.00 and 128K bytes of memory.

Data files created using Accounting Packages Version 1.00 by Peachtree Software, Inc. may be used with the Version 1.10 packages.

Arithmetic Games 1 and 2

When using the arithmetic games with DOS 2.00, you must have a minimum of 96K bytes of memory. In addition, if you have a single-drive IBM Personal Computer, you should use the single-drive setup procedure.

Asynchronous Communications Support Version 1.00

To use this application with DOS 2.00 you must have a minimum of 96K bytes of memory.

To install this application in a subdirectory on a fixed disk, use the following procedure. This procedure assumes the following:

- The name of the subdirectory is ASYNC.
- The fixed disk is the default drive.
- The fixed disk drive is drive C.
- The root directory of the fixed drive contains DOS 2.00.

The Procedure

1. Insert the Asynchronous Communications Support diskette in drive A.

2. Enter the following commands in order to create the subdirectory ASYNC and to copy the files from drive A to drive C:

   MD \ASYNC
   Create the subdirectory

   CD \ASYNC
   Operate from it

   COPY A:*.* C:\ASYNC
   Copy all files on the application diskette

   COPY C:\BASIC.COM C:\ASYNC
   Copy BASIC.COM to the subdirectory

   ERASE UPDATE.BAT
   Not needed

   ERASE MESSAGE
   Not needed

3. You must now patch TERMINAL.BAS by doing the following:

   BASIC
   Run BASIC

   LOAD "TERMINAL.BAS
   Load the program to be patched

   115 DEF SEG=&H1300
   Patch 1

   117 D$(1)="C:":D$(2)="A:"
   Patch 2

   210 IDSEG=&H1300
   Patch 3

   SAVE "TERMINAL.BAS"
   Save the patched program

   SYSTEM
   Leave BASIC and return to DOS 2.00

4. You may now run the Asynchronous Communications Support program by doing the following:

   CD \ASYNC
   Operate from the subdirectory

   AUTOEXEC
   Run the program

   CD \
   Return to the root directory

   Notes:

   1. The use of the subdirectory name of ASYNC is an example only. You may use any name you wish.

2. All file transfers will be to and from the files in the subdirectory unless the printer is the destination.

3. This program tests the length of a filespec, and does not accept subdirectory names as part of a filespec.

4. You may rename AUTOEXEC.BAT if you choose.

Asynchronous Communications Support Version 2.00

To use this application with DOS 2.00 you must have a minimum of 96K bytes of memory.

To install this application in a subdirectory on a fixed disk, use the following procedure. This procedure assumes the following:

- The name of the subdirectory is ASYNC.
- The fixed disk is the default drive.
- The fixed disk drive is drive C.
- The root directory of the fixed drive contains DOS 2.00.

The Procedure

1. Insert the Asynchronous Communications Support diskette into drive A.

2. Enter the following commands in order to create the subdirectory ASYNC and to copy the files from drive A to drive C:

   MD \ASYNC
   Create the subdirectory

   CD \ASYNC
   Operate from it

   COPY A:*.* C:\ASYNC
   Copy all files on the application diskette

   COPY C:\BASIC.COM C:\ASYNC
   Copy BASIC.COM to the subdirectory

   ERASE UPDATE.BAT
   Not needed

   ERASE MESSAGE
   Not needed

3. You may now run the Asynchronous Communications Support program by doing the following:

   CD \ASYNC
   Operate from the subdirectory

   AUTOEXEC
   Run the program

   CD \
   Return to the root directory

4. You may run the file conversion program, FILECONV, by doing the following:

   CD \ASYNC
   Operate from the subdirectory

   FILECONV
   Run the FILECONV program

   CD \
   Return to the root directory

   Notes:

   1. The use of the subdirectory name of ASYNC is an example only. You may use any name you wish.

   2. All file transfers will be to and from the files in the subdirectory unless the printer is the destination.

   3. This program tests the length of a filespec, and does not accept subdirectory names as part of a filespec.

   4. You may rename AUTOEXEC.BAT if you choose.

EasyWriter Version 1.10

To use EasyWriter Version 1.10 with DOS 2.00 it is recommended your IBM Personal Computer have a minimum of 128K bytes of memory.

To copy DOS 2.00 to your EasyWriter program diskette, use the following procedure:

1. Follow the instructions in the "Startup" section of your EasyWriter manual for copying DOS to your EasyWriter program diskette. While copying DOS 2.00 to the program diskette, you may see the message:

   INSUFFICIENT DISK SPACE (or)

   SECTOR NOT FOUND ERROR WRITING DRIVE B
   ABORT, RETRY, IGNORE?

   If you see the second message above, enter I (for ignore).

   > Note: For a system with a fixed disk, follow the instructions for copying DOS 2.00 to your EasyWriter program diskette for a one-drive system.

Due to insufficient diskette space, the DOS 2.00 FORMAT.COM utility was not transferred to your EasyWriter diskette. To use the FORMAT.COM utility, insert the DOS 2.00 diskette in drive A and a blank diskette in drive B. Then type: FORMAT B: and press the enter key. This will format the diskette in drive B.

2. EasyWriter 1.10 recognizes drive A and drive B as the drives to read or write from for EasyWriter files. Because the fixed drive is initially defined as drive C, you must reassign its name to be drive B. Enter:

   ASSIGN B=C

3. You can have your fixed disk automatically assigned to be drive B everytime you load your EasyWriter program. Use the following procedure to do this:

   a. Copy DOS 2.00 onto your EasyWriter program diskette following the instructions given above.

   b. Start DOS 2.00 from the fixed disk. The DOS prompt C> should appear.

   c. Put your EasyWriter program diskette into drive A.

   d. Enter COPY ASSIGN.COM A:

   e. Enter COPY CON A:AUTOEXEC.BAT

   f. Enter DATE

   g. Enter TIME

   h. Enter ASSIGN B=C

   i. Enter EW

   j. Enter ASSIGN

   k. Press the F6 key. Now press the Enter key.

   The fixed disk can now be used to store and read your EasyWriter data files. Every time you load your EasyWriter program the fixed disk will automatically be assigned as drive B.

Fact Track

To use Fact Track with DOS 2.00 you need 96K bytes of memory.

If you have DOS 1.10, you should use it to follow one of the setup procedures documented in the Fact Track user manual. The setup procedure copies the necessary DOS programs onto the Fact Track program diskette. From then on, start the Fact Track program diskette in drive A.

If you have only DOS 2.00, and if you have an IBM Personal Computer with one or two disk drives, you should start up DOS in drive A. Enter the date and time as requested. When the system responds:

```
A>
```

you enter:

```
BASICA
```

When the system responds:

```
OK
```

remove the DOS diskette from drive A and insert the Fact Track diskette into drive A and close the door.

Enter:

```
RUN "COLOR
```

Do this every time you want to run the Fact Track program.

If you have DOS 2.00, but do not have DOS 1.10, and an IBM Personal Computer with a fixed disk that contains DOS or BASICA, you should start up DOS from the fixed disk. Enter the date and time as requested. When the system responds:

```
C>
```

put the Fact Track program diskette in drive A and close the door.

You enter:

```
A:
```

When the system responds:

```
A>
```

you enter:

```
C:BASICA COLOR
```

Do this every time you want to run the Fact Track program.

PFS:File

To use PFS:File with DOS 2.00 you need a minimum of 128K bytes of memory.

Follow the instructions in Part 1 of the Introduction "Getting Ready to Use File." Use your DOS 2.00 diskette whenever the instructions ask for the DOS diskette.

Using PFS:File with the IBM Fixed Disk

You can use PFS:File with the IBM fixed disk in two different ways. First, you can store your files on the fixed disk, thus allowing larger files to be stored, and faster access to forms in the stored files. Second, you can copy the PFS:File program to the fixed disk and then load it from there. This allows you to load the program faster, without using the program diskette.

Storing a PFS:File on the Fixed Disk

You can store a PFS:File on the fixed disk provided you include the drive identifier for the fixed disk as part of the file name. If you specify the fixed disk drive as the default drive, then it is not necessary to include the drive identifier as part of the file name.

Copying PFS:File to the Fixed Disk

Follow these steps to copy the PFS:File program to the fixed disk:

1. Insert the DOS 2.00 diskette into drive A and turn on your IBM Personal Computer. Enter the date and time when the computer asks you to do so.

2. When the DOS prompt appears, remove the DOS 2.00 diskette and replace it with the PFS:File program diskette.

3. Follow the instructions in Appendix D of the PFS:File manual called "Setting Up a Serial Printer and the Work Drive" to run the setup program. Change the work drive name to the drive name of your fixed disk. For example, if your fixed drive is drive C, then the work drive item on the setup menu should be drive C.

4. If you have a serial printer, enter the correct values for the other items on the setup menu.

5. Press the F10 key to complete the setup program. Then enter FTRANS in response to the DOS prompt.

The in use lights will come on alternately as the program is copied from the diskette to the fixed disk. The copy is placed in the current directory. The DOS prompt reappears when the copy is complete.

If you have more than one fixed disk, the copy will be made to the drive whose name is last in the alphabet. For example, if two drives are named C and D, the copy will automatically be made to drive D.

Error Conditions

The following error conditions might occur during the copy procedure.

Message	Explanation	Corrective Action
CAN'T COPY PROGRAM FILE	Your program diskette has been damaged.	Try the copy procedure with the backup copy of the program diskette.
	Your fixed disk is improperly formatted.	Copy to diskette any files on the fixed disk. Then reformat the fixed disk, and try the copy procedure again.
CAN'T CREATE PROGRAM FILE	Your root level directory is full.	If you have unnecessary files in the root level directory, delete them and start the copy procedure again.
NAME ERROR	Possible DOS error.	Reload DOS 2.00 and start the copy procedure again.
WRONG VERSION	Cannot find the fixed disk.	Make sure that your fixed disk is properly connected, and that you are using DOS 2.00.

Running the PFS:File Program from a Fixed Disk

To run the PFS:File program from a fixed disk, make sure that you have followed the instructions under "Copying PFS:File to the Fixed Disk." Then in response to the DOS prompt, type and enter the drive identifier for the fixed disk followed by the name of the program. For example, to run the File program from drive C, enter C:FILE. If the default drive is C, you need only enter FILE in response to the DOS prompt.

Changing Settings When Using the Fixed Disk

To change the work drive or the information stored for your serial printer after copying the PFS:File program to the fixed disk, you need to insert the PFS:File program diskette into drive A and run the setup program from that diskette. Use the COPY command to copy the file named IBMSETUP.PFS from the diskette in drive A to the fixed disk.

PFS:Report

To use PFS:Report with DOS 2.00 you must have a minimum of 128K bytes of memory, and you should do the following:

1. Copy the sample file called STAFF, which is on the Report program diskette, to a blank formatted diskette or to a fixed disk. See "Formatting Diskettes" and "Copying a File" in Appendix C of the PFS:Report manual.

2. Erase the STAFF file from the Report program diskette using the ERASE command of DOS 2.00.

3. Now follow the instructions in Part 1 of the Introduction in the PFS:Report manual "Getting Ready to Use Report." Use your DOS 2.00 diskette when the instructions ask for the DOS diskette.

Using PFS:Report with the IBM Fixed Disk

You can use PFS:Report with the IBM fixed disk in two different ways. First, you can store your files on the fixed disk, thus allowing larger files to be stored, and faster access to forms in the stored files. Second, you can copy the PFS:Report program to the fixed disk and then load it from there. This allows you to load the program faster, without using the program diskette.

Storing a PFS:File on the Fixed Disk

You can store a PFS:File on the fixed disk provided you include the drive identifier for the fixed disk as part of the file name. If you specify the fixed disk drive as the default drive, then it is not necessary to include the drive identifier as part of the file name.

Copying PFS:Report to the Fixed Disk

Follow these steps to copy the PFS:Report program to the fixed disk:

1. Insert the DOS 2.00 diskette into drive A and turn on your IBM Personal Computer. Enter the date and time when the computer asks you to do so.

2. When the DOS prompt appears, remove the DOS 2.00 diskette and replace it with the PFS:Report program diskette.

3. Follow the instructions in Appendix D of the PFS:Report manual called "Setting Up a Serial Printer and the Work Drive" to run the setup program. Change the work drive name to the drive name of your fixed disk. For example, if your fixed drive is drive C, then the work drive item on the setup menu should be drive C.

4. If you have a serial printer, enter the correct values for the other items on the setup menu.

5. Press the F10 key to complete the setup program. Then enter RTRANS in response to the DOS prompt.

   The in use lights will come on alternately as the program is copied from the diskette to the fixed disk. The copy is placed in the current directory. The DOS prompt reappears when the copy is complete.

   If you have more than one fixed disk, the copy will be made to drive whose name is last in the alphabet. For example, if two drives are named C and D, the copy will automatically be made to drive D.

Error Conditions

The following error conditions might occur during the copy procedure.

Message	Explanation	Corrective Action
CAN'T COPY PROGRAM FILE	Your program diskette has been damaged.	Try the copy procedure with the backup copy of the program diskette.
	Your fixed disk is improperly formatted. (also applies to the CREATE message)	Copy to diskette any files on the fixed disk. Then reformat the fixed disk, and try the copy procedure again.
CAN'T CREATE PROGRAM FILE	Your current directory is full.	If you have unnecessary files in the current directory, delete them and start the copy procedure again.
NAME ERROR	Possible DOS error.	Reload DOS 2.00 and start the copy procedure again.
WRONG VERSION	Cannot find the fixed disk.	Make sure that your fixed disk is properly connected, and that you are using DOS 2.00.

Running the PFS:Report Program from a Fixed Disk

To run the PFS:Report program from a fixed disk, make sure that you have followed the instructions under "Copying PFS:Report to the Fixed Disk" (make sure the current directory contains PFS). Then in response to the DOS prompt, enter the drive identifier for the fixed disk followed by the name of the program. For example, to run the Report program from drive C, enter: C:REPORT. If the default drive is C, you need only type and enter REPORT in response to the DOS prompt.

Changing Settings When Using the Fixed Disk

To change the work drive or the information stored for your serial printer after copying the PFS:Report program to the fixed disk, you need to insert the PFS:Report program diskette into drive A and run the setup program from that diskette. Use the COPY command to copy the file named IBMSETUP.PFS from the diskette in drive A to the fixed disk.

The Dow Jones Reporter Version 1.00

The Dow Jones Reporter Version 1.00 directs data to be saved only on diskette drive A. In addition, the program diskette is copy protected, and you cannot install the program on a fixed disk.

Dow Jones is a registered trademark of the Dow Jones Company, Inc.

SNA 3270 Emulation and RJE Support Version 1.00

To use SNA 3270 Emulation and RJE Support Version 1.00 with DOS 2.00 you need a minimum of 128K bytes memory.

To install this application in a subdirectory, SNA, on a fixed disk, use the procedure described below, which assumes the following:

- The fixed disk is drive C.
- The fixed disk drive is the default drive.
- The root directory in the fixed disk contains DOS 2.00.

The Procedure

1. Insert the SNA program diskette into diskette drive A.
2. Edit the files 3270COPY.BAT and SRJECOPY.BAT to change all drive B references to drive C.

3. Enter the following commands to create the subdirectory, SNA, and to copy the needed files from drive A to drive C.

   MD \SNA
   Create the subdirectory

   CS \SNA
   Operate from it

   A:3270COPY
   Copy required files

   A:SRJECOPY
   Copy required files

   COPY C:\BASIC.COM C:\SNA
   Copy a needed file

4. You may now run the SNA program by entering the following commands:

   CD \SNA
   Operate from the SNA subdirectory

   programname
   Enter the correct program name as specified in the SNA manual in response to the DOS prompt

   CD \
   Return to the root directory

   Notes:

   1. The use of the subdirectory name SNA is as an example only. You may use any name you wish.

   2. All file transfers for SRJE are to and from files in the SNA subdirectory.

   3. The SNA program does not accept subdirectory names as part of a filespec.

Typing Tutor

To use Typing Tutor with DOS 2.00 it is recommended that your IBM Personal Computer have at least a minimum of 64K bytes memory. If you have a single diskette drive system, you should use the single drive setup procedure.

VisiCalc Version 1.10 by VisiCorp.

To use VisiCalc Version 1.10 with DOS 2.00 it is recommended that your IBM Personal Computer have a minimum of 128K bytes of memory. The following are procedures to put DOS 2.00 on your VisiCalc program diskette, and to make your program diskette self starting for fixed disk. These procedures assume the following:

- You have at least one diskette drive and one fixed disk.
- You are familiar with loading DOS 2.00 from your fixed disk.
- You are aware that VisiCalc supports a maximum of two secondary storage devices. Only one of these two devices may be accessible during operations.

Putting DOS 2.00 on Your Program Diskette

Follow these steps to put DOS 2.00 on your VisiCalc program diskette and to make that diskette self starting:

1. Start DOS 2.00. You should see the DOS prompt C>.
2. Remove the write protect tab from the VisiCalc program diskette.
3. Place the program diskette into drive A.
4. Enter COPY COMMAND.COM A:
5. Enter SYS A:

6. Enter COPY ASSIGN.COM A:

7. Enter COPY CON A:AUTOEXEC.BAT

8. Enter DATE

9. Enter TIME

10. Enter ASSIGN B=C (or any other drive designator)

11. Type VC80 (space) (press F6) (enter)

12. ASSIGN

13. Remove the program diskette and replace the write protect tab on your program diskette.

The fixed disk can now be used to store and retrieve your VisiCalc data files. Everytime you load your VisiCalc program the fixed disk will be assigned as drive B.

3101 Emulator Version 1.00

To use the 3101 Emulator Version 1.00 with DOS 2.00 requires a minimum of 96K bytes memory.

Use the procedure described below to install the 3101 Emulator in a subdirectory, EM3101, on the fixed disk. The procedure assumes the following:

- The fixed disk is drive C.
- The fixed disk is the default drive.

The Procedure

1. Insert the 3101 Emulator program diskette into drive A.

2. Enter the following commands to create the subdirectory, EM3101, and to copy the needed files from drive A to drive C:

   MD \EM3101
   Create the subdirectory

   CD \EM3101
   Operate from it

   COPY A:*.* C:\EM3101
   Copy all files on the 3101 Emulator program diskette

   ERASE COPYFILS.BAT
   Not needed

   ERASE AUTOEXEC.BAT
   Not needed

   CD \
   Return to root directory

3. You may now run the 3101 Emulator by entering the following commands:

   CD \EM3101
   Operate from the EM3101 subdirectory

   IBM3101
   Run the 3101 Emulator program

   CD \
   Return to the root directory

4. You may run the file conversion program, FILECONV, by entering the following commands:

   CD \EM3101
   Operate from the EM3101 subdirectory

   FILECONV
   Run the file conversion program

   CD \
   Return to the root directory

   Notes:

   1. The use of the subdirectory name EM3101 is as an example only. You may use any name you wish.

   2. All file transfers will be to and from files in the EM3101 subdirectory.

   3. The 3101 Emulator does not accept subdirectory names as part of a filespec.

Notes:

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Notes:

Reader's Comment Form

DOS **6936752**

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your Authorized IBM Personal Computer Dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 321 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Please do not staple

Tape

Continued from inside front cover

SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED PERSONAL COMPUTER DEALER) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassette(s) on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette(s) or cassette(s) not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized IBM PERSONAL COMPUTER dealer with a copy of your receipt, or
2. if IBM or the dealer is unable to deliver a replacement diskette(s) or cassette(s) which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL IBM BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF IBM OR AN AUTHORIZED IBM PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328–W, Boca Raton, Florida 33432.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

International Business Machines Corporation

P.O. Box 1328-W
Boca Raton, Florida 33432

6936752

Printed in United States of America