Chapter 20

Operating Systems Software and Troubleshooting

This chapter focuses on the problems that occur in PC systems because of faulty or incompatible software. First, it describes the structure of DOS and how DOS works with hardware in a functioning system. Topics of particular interest are as follows:

- DOS file structure
- DOS disk organization
- DOS programs for data and disk recovery (their capabilities and dangers)

Additionally, the chapter examines two other important software-related issues: using memory-resident software (and dealing with the problems it can cause) and distinguishing a software problem from a hardware problem.

Disk Operating System (DOS)

Information about DOS may seem out of place in a book about hardware upgrade and repair, but if you ignore DOS and other software when you troubleshoot a system, you can miss a number of problems. The best system troubleshooters and diagnosticians know the entire system—hardware and software.

This book cannot discuss DOS in depth, but if you need to read more about it, Que Corporation publishes some good books on the subject (*Using MS-DOS 6.2*, Special Edition, for example).

This section describes the basics of DOS: where it fits into the PC system architecture, what its components are, and what happens when a system boots (starts up). Understanding the booting process can be helpful when diagnosing startup problems. This section also explains DOS configuration—an area in which many people experience problems—and the file formats DOS uses, as well as how DOS manages information on a disk.

Operating System Basics

DOS is just one component in the total system architecture. A PC system has a distinct hierarchy of software that controls the system at all times. Even when you are operating within an application program such as 1-2-3 or another high-level application software, several other layers of programs are always executing underneath. Usually the layers can be defined distinctly, but sometimes the boundaries are vague.

Communications generally occur only between adjoining layers in the architecture, but this rule is not absolute. Many programs ignore the services provided by the layer directly beneath them and eliminate the middleman by skipping one or more layers. An example is a program that ignores the DOS and ROM BIOS video routines and communicates directly with the hardware in the interest of the highest possible screen performance. Although the high-performance goal is admirable, many operating environments (such as OS/2 and Windows) no longer allow direct access to the hardware. Programs that do not play by the rules must be rewritten to run in these new environments.

The hardware is at the lowest level of the system hierarchy. By placing various bytes of information at certain ports or locations within a system's memory structure, you can control virtually anything connected to the CPU. Maintaining control at the hardware level is difficult; doing so requires a complete and accurate knowledge of the system architecture. The level of detail required in writing the software operating at this level is the most intense. Commands to the system at this level are in *machine language* (binary groups of information applied directly to the microprocessor). Machine language instructions are limited in their function: you must use many of them to perform even the smallest amount of useful work. The large number of instructions required is not really a problem because these instructions are executed extremely rapidly, wasting few system resources.

Programmers can write programs consisting of machine language instructions, but generally they use a tool—an *assembler*—to ease the process. They write programs using an *editor,* and then use the assembler to convert the editor's output to pure machine language. Assembler commands are still very low level, and using them effectively requires that programmers be extremely knowledgeable. No one (in his or her right mind) writes directly in machine code anymore; assembly language is the lowest level of programming environment typically used today. Even assembly language, however, is losing favor among programmers because of the amount of knowledge and work required to complete even simple tasks and because of its lack of portability between different kinds of systems.

When you start a PC system, a series of machine code programs, the ROM BIOS, assumes control. This set of programs, always present in a system, talks (in machine code) to the hardware. The BIOS accepts or interprets commands supplied by programs above it in the system hierarchy and translates them to machine code commands that then are passed on to the microprocessor. Commands at this level typically are called *interrupts* or *services*. A programmer generally can use nearly any language to supply these instructions to the BIOS. A complete list of these services is supplied in the IBM *BIOS Interface Technical Reference Manual*.

DOS itself is made up of several components. It attaches to the BIOS, and part of DOS actually becomes an extension of the BIOS, providing more interrupts and services for other programs to use. DOS provides for communication with the ROM BIOS in PCs and with higher-level software (such as applications). Because DOS gives the programmer interrupts and services to use in addition to those provided by the ROM BIOS, a lot of reinventing the wheel in programming routines is eliminated. For example, DOS provides an extremely rich set of functions that can open, close, find, delete, create, rename, and perform other file-handling tasks. When programmers want to include some of these functions in their programs, they can rely on DOS to do most of the work.

This standard set of functions that applications use to read from and write to disks makes data recovery operations possible. Imagine how tough writing programs and using computers would be if every application program had to implement its own custom disk interface, with a proprietary directory and file retrieval system. Every application would require its own special disks. Fortunately, DOS provides a standard set of documented file storage and retrieval provisions that all software can use; as a result, you can make some sense out of what you find on a typical disk.

Another primary function of DOS is to load and run other programs. As it performs that function, DOS is the *shell* within which another program can be executed. DOS provides the functions and environment required by other software—including operating environments such as GEM and Windows—to run on PC systems in a standard way.

The System ROM BIOS

Think of the system ROM BIOS as a form of compatibility glue that sits between the hardware and an operating system. Why is it that IBM can sell the same DOS to run on the original IBM PC *and* on the latest Pentium systems—two very different hardware platforms? If DOS were written to talk directly to the hardware on all systems, it would be a very hardware-specific program. Instead, IBM developed a set of standard services and functions each system should be capable of performing and coded them as programs in the ROM BIOS. Each system then gets a completely custom ROM BIOS that talks directly to the hardware in the system and knows exactly how to perform each specific function on that hardware only.

This convention enables operating systems to be written to what amounts to a standard interface that can be made available on many different types of hardware. Any applications written to the operating system standard interface can run on that system. Figure 20.1 shows that two very different hardware platforms can each have a custom ROM BIOS that talks directly to the hardware and still provides a standard interface to an operating system.

The two different hardware platforms described in figure 20.1 can run not only the exact same version of DOS, but also the same application programs because of the standard interfaces provided by the ROM BIOS and DOS. Keep in mind, however, that the actual ROM BIOS code differs among the specific machines and that it is not usually possible therefore to run a ROM BIOS designed for one system in a different system. ROM BIOS

upgrades must come from a source that has an intimate understanding of the specific motherboard on which the chip will be placed because the ROM must be custom written for that particular hardware.

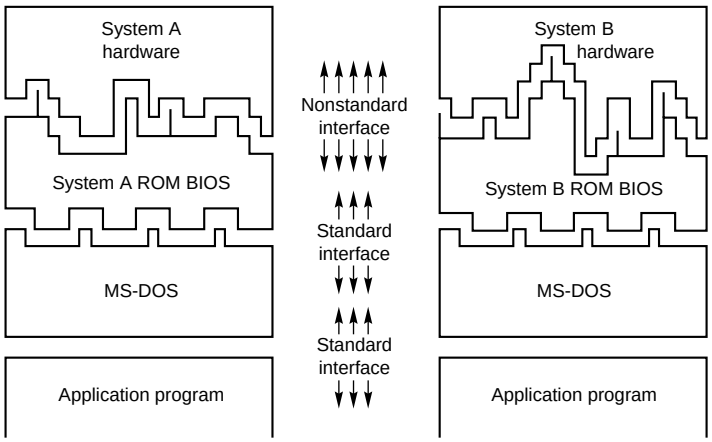

Fig. 20.1
A representation of the software layers in an IBM-compatible system.

The portion of DOS shown in figure 20.1 is the system portion, or core, of DOS. This core is found physically as the two system files on any bootable DOS disk. These hidden system files usually have one of two sets of names, IBMBIO.COM and IBMDOS.COM (used in IBM and Compaq DOS), or IO.SYS and MSDOS.SYS (used in MS-DOS and versions of DOS licensed from Microsoft by original equipment manufacturers [OEMs]). These files must be the first and second files listed in the directory on a bootable DOS disk.

Figure 20.1 represents a simplified view of the system; some subtle but important differences exist. Ideally, application programs are insulated from the hardware by the ROM BIOS and DOS, but in reality many programmers write portions of their programs to talk directly to the hardware, circumventing DOS and the ROM BIOS. A program therefore might work only on specific hardware, even if the proper DOS and ROM BIOS interfaces are present in other hardware.

Programs designed to go directly to the hardware are written that way mainly to increase performance. For example, many programs directly access the video hardware to improve screen update performance. These applications often have install programs that require you to specify exactly what hardware is present in your system so that the program can load the correct hardware-dependent routines into the application.

Additionally, some utility programs absolutely must talk directly to the hardware to perform their function. For example, a low-level format program must talk directly to the hard disk controller hardware to perform the low-level format of the disk. Such programs are very specific to a certain controller or controller type. Another type of system-specific utility, the driver programs, enables extended memory to function as expanded memory

on an 80386-based system. These drivers work by accessing the 80386 directly and utilizing specific features of the chip.

Another way that reality might differ from the simple view is that DOS itself communicates directly with the hardware. In fact, much of the IBMBIO.COM file consists of low-level drivers designed to supplant and supersede ROM BIOS code in the system. People who own both IBM systems and compatibles might wonder why IBM never seems to have ROM BIOS upgrades to correct bugs and problems with its systems, although for vendors of most compatible systems, a ROM upgrade is at least a semiannual occurrence. The reason is simple: IBM distributes its ROM patches and upgrades in DOS. When IBM DOS loads, it determines the system type and ID information from the ROM and loads different routines depending on which version of ROM it finds. For example, at least four different hard disk code sections are in IBM DOS, but only one is loaded for a specific system.

I have taken a single DOS boot disk with only the system files (COMMAND.COM and CHKDSK.COM) on it, and booted the disk on both an XT and an AT system, each one with an identical 640K of memory. After loading DOS, CHKDSK reported different amounts of free memory, which showed that DOS had taken up different amounts of memory in the two systems. This is because of the different code routines loaded, based on the ROM ID information. In essence, DOS, the ROM BIOS, and the hardware are much more closely related than most people realize.

DOS Components

DOS consists of two primary components: the input/output (I/O) system and the shell. The I/O system consists of the underlying programs that reside in memory while the system is running; these programs are loaded first when DOS boots. The I/O system is stored in the form of two files hidden on a bootable DOS disk. The files are called IBMBIO.COM and IBMDOS.COM on an IBM DOS disk, but might go by other names for other manufacturers' versions of DOS. For example, IO.SYS and MSDOS.SYS are the MS-DOS file names. No matter what the exact names are, the function of these two files is basically the same for all versions of DOS. However, each individual system's ROM BIOS looks for the system files by name and often does not recognize them by another name. This is one reason that the OEM version of DOS you use must be the correct one for your system.

The user interface program, or shell, is stored in the COMMAND.COM file, which also is loaded during a normal DOS boot-up. The shell is the portion of DOS that provides the DOS prompt and that normally communicates with the user of the system.

The following sections examine the DOS I/O system and shell in more detail to help you properly identify and solve problems related to DOS rather than to hardware. Also included is a discussion on how DOS allocates disk file space.

The I/O System and System Files

This section briefly describes the two files that make up the I/O system: IBMBIO.COM and IBMDOS.COM.

IBMBIO.COM (or IO.SYS). IBMBIO.COM is one of the hidden files that the CHKDSK command reports on any system (bootable) disk. This file contains the low-level programs that interact directly with devices on the system and the ROM BIOS. IBMBIO.COM usually is customized by the particular original equipment manufacturer (OEM) of the system to match perfectly with that OEM's ROM BIOS. The file contains low-level drivers loaded in accord with a particular ROM BIOS, based on the ROM ID information, as well as on a system initialization routine. During boot-up, the DOS volume boot sector loads the file into low memory and gives it control of the system (see the "DOS Volume Boot Sectors" section, later in this chapter). The entire file, except the system initializer portion, remains in memory during normal system operation.

The name used for the file that performs the functions just described varies among versions of DOS from different OEMs. Many versions of DOS, including Microsoft's MS-DOS, use IO.SYS as the name of this file. Some other manufacturers call the file MIO.SYS, and Toshiba calls it TBIOS.SYS. Using different names for this file is not normally a problem, until you try to upgrade from one OEM version of DOS to a different OEM version. If the different OEMs use different names for this file, the SYS command might fail with the error message `No room for system on destination`. Today, most OEMs use the standard IBMBIO.COM name for this file to eliminate problems in upgrading and otherwise remain standard.

For a disk to be bootable, IBMBIO.COM or its equivalent must be listed as the first file in the directory of the disk and must occupy at least the first cluster on the disk (cluster number 2). The remainder of the file might be placed in clusters anywhere across the rest of the disk (versions 3 and higher). The file normally is marked with hidden, system, and read-only attributes, and is placed on a disk by the FORMAT command or the SYS command.

IBMDOS.COM (or MSDOS.SYS). IBMDOS.COM, the core of DOS, contains the DOS disk handling programs. The routines present in this file make up the DOS disk and device handling programs. IBMDOS.COM is loaded into low memory at system boot-up by the DOS volume boot sector and remains resident in memory during normal system operation.

The IBMDOS.COM program collection is less likely to be customized by an OEM, but still might be present on a system by a name different from IBMDOS.COM. The most common alternative name, MSDOS.SYS, is used by Microsoft's MS-DOS and some OEM versions of DOS. Another name is TDOS.SYS (used by Toshiba). Most OEMs today stick to the IBM convention to eliminate problems in upgrading from one DOS version to another.

IBMDOS.COM or its equivalent must be listed as the second entry in the root directory of any bootable disk. This file usually is marked with hidden, system, and read-only attributes, and normally is placed on a disk by the FORMAT command or the SYS command. There are no special requirements for the physical positioning of this file on a disk.

The Shell or Command Processor (COMMAND.COM). The DOS command processor COMMAND.COM is the portion of DOS with which users normally interact. The commands can be categorized by function, but IBM DOS divides them into two types by how they are made available: *resident* or *transient*.

Resident commands are built into COMMAND.COM and are available whenever the DOS prompt is present. They are generally the simpler, frequently used commands such as CLS and DIR. Resident commands execute rapidly because the instructions for them are already loaded into memory. They are *memory-resident*.

When you look up the definition of a command in the DOS manual, you find an indication of whether the command is resident or transient. You then can determine what is required to execute that command. A simple rule is that, at a DOS prompt, all resident commands are instantly available for execution, with no loading of the program from disk required. Resident commands are also sometimes termed *internal*. Commands run from a program on disk are termed *external*, or *transient*, and also are often called *utilities*.

Transient commands are not resident in the computer's memory, and the instructions to execute the command must be located on a disk. Because the instructions are loaded into memory only for execution and then are overwritten in memory after they are used, they are called *transient commands*. Most DOS commands are transient; otherwise, the memory requirements for DOS would be astronomical. Transient commands are used less frequently than resident commands and take longer to execute because they must be found and loaded before they can be run.

Most executable files operate like transient DOS commands. The instructions to execute the command must be located on a disk. The instructions are loaded into memory only for execution and are overwritten in memory after the program is no longer being used.

DOS Commands. Tables 20.1 through 20.3 show all the resident, batch, and transient DOS commands and in which DOS version they are supported. If you are responsible for providing technical support, you should know what DOS commands are available to the users at the other end of the phone. These tables identify which commands are supported in any version of DOS released to date.

Table 20.1 Resident DOS Commands

Command Name	DOS Version Number										
	1.0	1.1	2.0	2.1	3.0	3.1	3.2	3.3	4.x	5.x	6.x
CD/CHDIR			×	×	×	×	×	×	×	×	×
CHCP									×	×	×
CLS			×	×	×	×	×	×	×	×	×
COPY	×	×	×	×	×	×	×	×	×	×	×
CTTY			×	×	×	×	×	×	×	×	×
DATE	×	×	×	×	×	×	×	×	×	×	×

(continues)

Table 20.1 Continued

Command Name	DOS Version Number 1.0	1.1	2.0	2.1	3.0	3.1	3.2	3.3	4.x	5.x	6.x
DEL/ERASE	×	×	×	×	×	×	×	×	×	×	×
DIR	×	×	×	×	×	×	×	×	×	×	×
EXIT					×	×	×	×	×	×	×
EXPAND										×	×
LOADHI/LH										×	×
MD/MKDIR			×	×	×	×	×	×	×	×	×
PATH			×	×	×	×	×	×	×	×	×
PROMPT			×	×	×	×	×	×	×	×	×
RD/RMDIR			×	×	×	×	×	×	×	×	×
REN/RENAME	×	×	×	×	×	×	×	×	×	×	×
SET			×	×	×	×	×	×	×	×	×
TIME	×	×	×	×	×	×	×	×	×	×	×
TYPE	×	×	×	×	×	×	×	×	×	×	×
VER					×	×	×	×	×	×	×
VERIFY			×	×	×	×	×	×	×	×	×
VOL			×	×	×	×	×	×	×	×	×

Table 20.2 DOS Batch File Commands

Command Name	DOS Version Number 1.0	1.1	2.0	2.1	3.0	3.1	3.2	3.3	4.x	5.x	6.x
CALL								×	×	×	×
ECHO	×	×	×	×	×	×	×	×	×	×	×
FOR	×	×	×	×	×	×	×	×	×	×	×
GOTO	×	×	×	×	×	×	×	×	×	×	×
IF	×	×	×	×	×	×	×	×	×	×	×
PAUSE	×	×	×	×	×	×	×	×	×	×	×
REM	×	×	×	×	×	×	×	×	×	×	×
SHIFT	×	×	×	×	×	×	×	×	×	×	×

Table 20.3 Transient DOS Commands

Command Name	DOS Version Number 1.0	1.1	2.0	2.1	3.0	3.1	3.2	3.3	4.x	5.x	6.x
APPEND								×	×	×	×
ASSIGN			×	×	×	×	×	×	×	×	×
ATTRIB					×	×	×	×	×	×	×
BACKUP			×	×	×	×	×	×	×	×	×

Command Name	DOS Version Number 1.0	1.1	2.0	2.1	3.0	3.1	3.2	3.3	4.x	5.x	6.x
BASIC	×	×	×	×	×	×	×	×	×	×	×
BASICA	×	×	×	×	×	×	×	×	×	×	×
CHCP								×	×	×	×
CHKDSK	×	×	×	×	×	×	×	×	×	×	×
COMMAND			×	×	×	×	×	×	×	×	×
COMP	×	×	×	×	×	×	×	×	×	×	×
DEBUG	×	×	×	×	×	×	×	×	×	×	×
DISKCOMP	×	×	×	×	×	×	×	×	×	×	×
DISKCOPY	×	×	×	×	×	×	×	×	×	×	×
DOSKEY										×	×
DOSSHELL									×	×	×
EDIT										×	×
EDLIN	×	×	×	×	×	×	×	×	×	×	×
EMM386										×	×
EXE2BIN			×	×	×	×	×			×	×
FASTOPEN								×	×	×	×
FC										×	×
FDISK			×	×	×	×	×	×	×	×	×
FIND			×	×	×	×	×	×	×	×	×
FORMAT	×	×	×	×	×	×	×	×	×	×	×
GRAFTABL			×	×	×	×	×	×	×	×	×
GRAPHICS			×	×	×	×	×	×	×	×	×
HELP										×	
JOIN					×	×	×	×	×	×	×
KEYB								×	×	×	×
KEYBFR					×	×	×				
KEYBGR					×	×	×				
KEYBIT					×	×	×				
KEYBSP					×	×	×				
KEYBUK					×	×	×				
LABEL					×	×	×	×	×	×	×
LIB	×	×	×	×	×	×	×				
LINK	×	×	×	×	×	×	×				
MEM									×	×	×
MIRROR										×	×
MODE	×	×	×	×	×	×	×	×	×	×	×
MORE			×	×	×	×	×	×	×	×	×
NLSFUNC								×	×	×	×
PRINT			×	×	×	×	×	×	×	×	×

(continues)

Table 20.3 Continued

Command Name	DOS Version Number 1.0	1.1	2.0	2.1	3.0	3.1	3.2	3.3	4.x	5.x	6.x
QBASIC										×	
RECOVER			×	×	×	×	×	×	×	×	×
REPLACE					×	×	×	×	×	×	×
RESTORE			×	×	×	×	×	×	×	×	×
SETVER										×	×
SHARE						×	×	×	×	×	×
SORT			×	×	×	×	×	×	×	×	×
SUBST					×	×	×	×	×	×	×
SYS	×	×	×	×	×	×	×	×	×	×	×
TREE			×	×	×	×	×	×	×	×	×
UNDELETE										×	×
UNFORMAT										×	×
XCOPY							×	×	×	×	×

LIB, LINK, and EXE2BIN are included with the DOS technical-reference manual for DOS versions 3.3 and higher. EXE2BIN is included with DOS 5.0.

DOS Command File Search Procedure. DOS looks only in specific places for the instructions for a transient command, or a software application's executable file. The instructions that represent the command or program are in files on one or more disk drives. Files that contain execution instructions have one of three specific extensions to indicate to DOS that they are program files: .COM (command files), .EXE (executable files), or .BAT (batch files). .COM and .EXE files are machine code programs; .BAT files contain a series of commands and instructions using the DOS batch facilities. The places in which DOS looks for these files is controlled by the current directory and the PATH command.

In other words, if you type several characters, such as **WIN**, at the DOS prompt and press the Enter key, DOS attempts to find and run a program named WIN. DOS performs a two- or three-level search for program instructions (the file). The first step in looking for command instructions is to see whether the command is a resident one and, if so, run it from the program code already loaded. If the command is not resident, DOS looks in the current directory for .COM, .EXE, and .BAT files, in that order, and loads and executes the first file it finds with the specified name. If the command is not resident and not in the current directory, DOS looks in all the directories specified in the DOS PATH setting (which the user can control); DOS searches for the file within each directory in the extension order just indicated. Finally, if DOS fails to locate the required instructions,

it displays the error message `Bad command or filename`. This error message might be misleading because the command instructions usually are missing from the search areas rather than actually being bad.

Suppose that, at the DOS prompt, I type the command **XYZ** and press Enter. This command sends DOS on a search for the XYZ program's instructions. If DOS is successful, the program starts running within seconds. If DOS cannot find the proper instructions, an error message is displayed. Here is what happens:

1. DOS checks internally to see whether it can find the XYZ command as one of the resident commands whose instructions are already loaded. It finds no XYZ command as resident.
2. DOS looks next in the current directory on the current drive for files named XYZ.COM, then for files named XYZ.EXE, and finally for files named XYZ.BAT. Suppose that I had logged in to drive C:, and the current directory is \ (the root directory); therefore, DOS did not find the files in the current directory.
3. DOS looks to see whether a PATH is specified. If not, the search ends here. In this scenario, I do have a PATH specified when my system was started, so DOS checks every directory listed in that PATH for the first file it can find named XYZ.COM, XYZ.EXE, or XYZ.BAT (in that order). My PATH lists several directories, but DOS does not find an appropriate file in any of them.
4. The search ends, and DOS gives me the message `Bad command or filename`.

For this search-and-load procedure to be successful, I must ensure that the desired program or command file exists in the current directory on the current drive, or I must set my DOS PATH to point to the drive and directory in which the program does exist. This is why the PATH is so powerful in DOS.

A common practice is to place all simple command files or utility programs in one directory and set the PATH to point to that directory. Then each of those programs (commands) is instantly available by simply typing its name, just as though it were resident.

This practice works well only for single-load programs such as commands and other utilities. Major applications software often consists of many individual files and might have problems if they are called up from a remote directory or drive using the DOS PATH. The reason is that when the application looks for its overlay and accessory files, the DOS PATH setting has no effect.

On a hard disk system, users typically install all transient commands and utilities in subdirectories and ensure that the PATH points to those directories. The path literally is a list of directories and subdirectories in the AUTOEXEC.BAT file that tells DOS where to search for files when these files are not in the same directory you are when you enter a command. The system then functions as though all the commands were resident because

DOS finds the necessary files without further thought or effort on the part of the user. A path on such a hard drive may look like this:

```
PATH=C:\DOS;C:\BAT;C:\UTILS;
```

It is important to know that when DOS loads each time you power up your system, it looks for two such text files. The first text file DOS looks for is CONFIG.SYS, which also can be edited by the system user. This file loads device drivers like ANSI.SYS. The following is an example of a common CONFIG.SYS file:

```
FILES=30
BUFFERS=17
SHELL=C:\DOS\COMMAND.COM C:\DOS /E:512 /P
LASTDRIVE=G
DEVICE=C:\DOS\ANSI.SYS
```

The second text file DOS looks for each time you power up your system is AUTOEXEC.BAT, which sets the PATH and loads memory-resident programs and performs other system configuration tasks like creating a C:\> prompt. A typical AUTOEXEC.BAT file might look like the following:

```
@ECHO OFF
PROMPT $P$G
PATH=C:\DOS;C:\BAT;C:\UTILS;
\DOS\MODE CON: RATE=32 DELAY=1
\DOS\DOSKEY
```

The PATH normally cannot exceed 128 characters in length (including colons, semicolons, and backslashes). As a result of that limitation, you cannot have a PATH that contains all your directories if the directory names exceed 128 characters. For more information on the AUTOEXEC.BAT and CONFIG.SYS files, consult Que's *Using DOS 6.2*, Special Edition, or *Using IBM PC DOS 6.1*.

You can completely short-circuit the DOS command search procedure by simply entering at the command prompt the complete path to the file. For example, rather than include C:\DOS in the PATH and enter this command:

```
C:\>CHKDSK
```

You can enter the full name of the program:

```
C:\>C:\DOS\CHKDSK.COM
```

The latter command immediately locates and loads the CHKDSK program with no search through the current directory or PATH setting. This method of calling up a program speeds the location and execution of the program and works especially well to increase the speed of DOS batch file execution.

A few major software applications have problems if they are called up from a remote directory or drive using the DOS PATH. Such an application often is made up of many individual files, including overlay and accessory files. Problems can occur when an application expects you to run it from its own directory by making that directory current and

then running the program's COM or EXE file. Such applications look for their own files in the current directory. If you did not change to the application's directory, because the program does not look for its files by checking the path, the program does not find its own files. The path entry in AUTOEXEC.BAT has no effect.

Such applications can be called up through batch files or aided by programs that "force-feed" a path-type setting to the programs; the software then works as though files are "here" even when they are in some other directory. The best utility for this purpose is the APPEND command in DOS 3.0 and later versions. For information on the use of the APPEND command see Que's *Using MS-DOS 6*.

DOS History

The following section details some of the differences between the DOS versions that have appeared over the years.

IBM and MS DOS 1.x to 3.x Versions. There have been many specific DOS versions in the 1.x to 3.x range from both IBM and Microsoft, as well as a few other OEMs. Table 20.4 lists the file dates and sizes for the major IBM and Microsoft DOS versions. You can see how DOS has grown over the years!

Table 20.4 System File Sizes

DOS Version	File Dates	COMMAND.COM	IO.SYS IBMBIO.COM	MSDOS.SYS IBMDOS.COM
IBM PC 1.0	08-04-81	3,231	1,920	6,400
IBM PC 1.1	05-07-82	4,959	1,920	6,400
IBM PC 2.0	03-08-83	17,792	4,608	17,152
IBM PC 2.1	10-20-83	17,792	4,736	17,024
IBM PC 3.0	08-14-84	22,042	8,964	27,920
IBM PC 3.1	03-07-85	23,210	9,564	27,760
IBM PC 3.2	12-30-85	23,791	16,369	28,477
MS 3.2	07-07-86	23,612	16,138	28,480
IBM PC 3.3	03-17-87	25,307	22,100	30,159
MS 3.3	07-24-87	25,276	22,357	30,128

IBM DOS 4.xx Versions. DOS 4.xx has had many revisions since being introduced in mid-1988. Since the first release, IBM has released different Corrective Service Diskettes (CSDs), which fix a variety of problems with DOS 4. Each CSD is cumulative, which means that the later ones include all previous fixes. Note that these fixes are for IBM DOS and not for any other manufacturer's version.

Table 20.5 shows a summary of the different IBM DOS 4.xx releases and specific information about the system files and shell so that you can identify the release you are using. To obtain the latest Corrective Service Diskettes (CSDs) that update you to the latest release, contact your dealer—the fixes are free.

Table 20.5 IBM DOS 4.xx Releases

File Name	Size	Date	Version	SYSLEVEL	Comments
IBMBIO.COM	32810	06/17/88	4.00	—	Original release.
IBMDOS.COM	35984	06/17/88			
COMMAND.COM	37637	06/17/88			
IBMBIO.COM	32816	08/03/88	4.01	CSD UR22624	EMS fixes.
IBMDOS.COM	36000	08/03/88			
COMMAND.COM	37637	06/17/88			
IBMBIO.COM	32816	08/03/88	4.01	CSD UR24270	Date change fixed.
IBMDOS.COM	36000	11/11/88			
COMMAND.COM	37652	11/11/88			
IBMBIO.COM	33910	04/06/89	4.01	CSD UR25066	"Death disk" fixed.
IBMDOS.COM	37136	04/06/89			
COMMAND.COM	37652	11/11/88			
IBMBIO.COM	34660	03/20/90	4.01	CSD UR29015	SCSI support added.
IBMDOS.COM	37248	02/20/90			
COMMAND.COM	37765	03/20/90			
IBMBIO.COM	34660	04/27/90	4.01	CSD UR31300	HPFS compatibility.
IBMDOS.COM	37264	05/21/90			
COMMAND.COM	37765	06/29/90			
IBMBIO.COM	34692	04/08/91	4.01	CSD UR35280	HPFS and CHKDSK.
IBMDOS.COM	37280	11/30/90			
COMMAND.COM	37762	09/27/91			

IBM DOS 5.xx Versions. DOS 5.xx has had several different revisions since being introduced in mid-1991. Since the first release, IBM has released various Corrective Service Diskettes (CSDs), which fix a variety of problems with DOS 5. Each CSD is cumulative, which means that the later ones include all previous fixes. Note that these fixes are for IBM DOS and not any other manufacturer's version. IBM typically provides more support in the way of fixes and updates than any other manufacturer. Note that IBM now supports the installation of IBM DOS on clone systems.

Table 20.6 shows a summary of the different IBM DOS 5.xx releases and specific information about the system files and shell so that you can identify the release you are using. To obtain the latest Corrective Service Diskettes (CSDs) that update you to the latest release, contact your dealer—the fixes are free.

Table 20.6 IBM DOS 5.xx Releases

File Name	Size	Date	Version	SYSLEVEL	Comments
IBMBIO.COM IBMDOS.COM COMMAND.COM	33430 37378 47987	05/09/91 05/09/91 05/09/91	5.00	—	Original release.
IBMBIO.COM IBMDOS.COM COMMAND.COM	33430 37378 48005	05/09/91 05/09/91 08/16/91	5.00	CSD UR35423	XCOPY and QEDIT fixed.
IBMBIO.COM IBMDOS.COM COMMAND.COM	33430 37378 48006	05/09/91 05/09/91 10/25/91	5.00	CSD UR35748	SYS fixed.
IBMBIO.COM IBMDOS.COM COMMAND.COM	33446 37378 48006	11/29/91 11/29/91 11/29/91	5.00	CSD UR35834	EMM386, FORMAT, and BACKUP fixed.
IBMBIO.COM IBMDOS.COM COMMAND.COM	33446 37378 48006	02/28/92 11/29/91 02/28/92	5.00.1 Rev. A	CSD UR36603	Many fixes; clone support; new retail version.
IBMBIO.COM IBMDOS.COM COMMAND.COM	33446 37362 48042	05/29/92 05/29/92 09/11/92	5.00.1 Rev. 1	CSD UR37387	RESTORE and UNDELETE fixed; >1GB HD fixed.
IBMBIO.COM IBMDOS.COM COMMAND.COM	33718 37362 47990	09/01/92 09/01/92 09/01/92	5.02 Rev. 0	—	New retail version; several new commands added.

IBM and MS DOS 6.xx Versions. There are several different versions of DOS 6.xx from both Microsoft and IBM. The original release of MS-DOS 6.0 came from Microsoft. One of the features included in 6.0 was the new DoubleSpace disk compression. Unfortunately, DoubleSpace had some problems with certain system configurations and hardware types. In the meantime, IBM took DOS 6.0 from Microsoft, updated it to fix several small problems, removed the disk compression, and sold it as IBM DOS 6.1. Microsoft had many problems with the DoubleSpace disk compression used in 6.0 and released 6.2 as a free bug fix upgrade. Microsoft then ran into legal problems in a lawsuit brought by Stacker Corporation. Microsoft was found to have infringed on the Stacker software and was forced to remove the DoubleSpace compression from DOS 6.2, which was released as 6.21. Microsoft then quickly developed a noninfringing disk compression utility called DriveSpace, which was released in 6.22, along with several minor bug fixes. IBM skipped over the 6.2 version number and released DOS 6.3 (now called PC DOS), which also included a different type of compression program than that used by Microsoft. By avoiding the DoubleSpace software, IBM also avoided the bugs and legal problems that Microsoft had encountered. Also included in the updated IBM releases are enhanced PCMCIA and power management commands.

Table 20.7 shows a summary of the different IBM DOS 6.xx releases.

Table 20.7 IBM and Microsoft DOS 6.xx Releases

File Name	Size	Date	Version	SYSLEVEL	Comments
IO.SYS MSDOS.SYS COMMAND.COM	40470 38138 52925	03/10/93 03/10/93 03/10/93	MS 6.00 Rev. A	—	Original Microsoft release.
IBMBIO.COM IBMDOS.COM COMMAND.COM	40694 38138 52589	06/29/93 06/29/93 06/29/93	IBM 6.10 Rev. 0	—	Original IBM release. Has fixes over MS version.
IBMBIO.COM IBMDOS.COM COMMAND.COM	40964 38138 52797	09/30/93 09/30/93 09/30/93	PC 6.10 Rev. 0	—	SuperStor/DS compression; enhanced PCMCIA.
IO.SYS MSDOS.SYS COMMAND.COM	40566 38138 54619	09/30/93 09/30/93 09/30/93	MS 6.20 Rev. A	—	DoubleSpace fixes. Enhanced cleanboot and data recovery.
IO.SYS MSDOS.SYS COMMAND.COM	40774 38138 54645	05/31/94 05/31/94 05/31/94	MS 6.22 Rev. A	—	New DriveSpace disk compression software and minor fixes.
IBMBIO.COM IBMDOS.COM COMMAND.COM	40758 38138 54804	12/31/93 12/31/93 08/12/94	IBM 6.30 Rev. 0	—	Numerous bug fixes; new disk compression software.

The Boot Process

The term *boot* comes from the term *bootstrap* and describes the method by which the PC becomes operational. Just as you pull on a large boot by the small strap attached to the back, a PC can load a large operating system program by first loading a small program that then can pull in the operating system. A chain of events begins with the application of power and finally results in an operating computer system with software loaded and running. Each event is called by the event before it and initiates the event after it.

Tracing the system boot process might help you find the location of a problem if you examine the error messages the system displays when the problem occurs. If you can see an error message displayed only by a particular program, you can be sure that the program in question was at least loaded and partially running. Combine this information with the knowledge of the boot sequence, and you can at least tell how far along the system's startup procedure is. You usually want to look at whatever files or disk areas were being accessed during the failure in the boot process. Error messages displayed during the boot process as well as those displayed during normal system operation can be hard to decipher, but the first step in decoding an error message is to know where the message came from—what program actually sent or displayed the message. The following programs are capable of displaying error messages during the boot process:

- Motherboard ROM BIOS
- Adapter card ROM BIOS extensions
- Master partition boot sector
- DOS volume boot sector

- System files (IBMBIO.COM and IBMDOS.COM)
- Device drivers (loaded through CONFIG.SYS)
- Shell program (COMMAND.COM)
- Programs run by AUTOEXEC.BAT

This section examines the system startup sequence and provides a detailed account of many of the error messages that might occur during this process.

How DOS Loads and Starts

If you have a problem with your system during startup and you can determine where in this sequence of events your system has stalled, you know what events have occurred and you probably can eliminate each of them as a cause of the problem. The following steps occur in a typical system startup:

1. You switch on electrical power to the system.
2. The power supply performs a self-test. When all voltages and current levels are acceptable, the supply indicates that the power is stable and sends the Power Good signal to the motherboard. The time from switch-on to Power Good is normally between .1 and .5 seconds.
3. The microprocessor timer chip receives the Power Good signal, which causes it to stop generating a reset signal to the microprocessor.
4. The microprocessor begins executing the ROM BIOS code, starting at memory address FFFF:0000. Because this location is only 16 bytes from the very end of the available ROM space, it contains a JMP (jump) instruction to the actual ROM BIOS starting address.
5. The ROM BIOS performs a test of the central hardware to verify basic system functionality. Any errors that occur are indicated by audio codes because the video system has not yet been initialized.
6. The BIOS performs a video ROM scan of memory locations C000:0000 through C780:0000, looking for video adapter ROM BIOS programs contained on a video adapter card plugged into a slot. If a video ROM BIOS is found, it is tested by a checksum procedure. If it passes the checksum test, the ROM is executed; the video ROM code initializes the video adapter, and a cursor appears on-screen. If the checksum test fails, the following message appears:

   ```
   C000 ROM Error
   ```

7. If the BIOS finds no video adapter ROM, it uses the motherboard ROM video drivers to initialize the video display hardware, and a cursor appears on-screen.
8. The motherboard ROM BIOS scans memory locations C800:0000 through DF80:0000 in 2K increments for any other ROMs located on any other adapter cards. If any ROMs are found, they are checksum-tested and executed. These adapter ROMs can alter existing BIOS routines as well as establish new ones.

9. Failure of a checksum test for any of these ROM modules causes this message to appear:

```
XXXX ROM Error
```

10. The address XXXX indicates the segment address of the failed ROM module.

11. The ROM BIOS checks the word value at memory location 0000:0472 to see whether this start is a cold start or a warm start. A word value of 1234h in this location is a flag that indicates a warm start, which causes the memory test portion of the POST (Power-On Self Test) to be skipped. Any other word value in this location indicates a cold start and full POST.

12. If this is a cold start, the POST executes. Any errors found during the POST are reported by a combination of audio and displayed error messages. Successful completion of the POST is indicated by a single beep.

13. The ROM BIOS searches for a DOS volume boot sector at cylinder 0, head 0, sector 1 (the very first sector) on the A drive. This sector is loaded into memory at 0000:7C00 and tested. If a disk is in the drive but the sector cannot be read, or if no disk is present, the BIOS continues with the next step.

14. If the first byte of the DOS volume boot sector loaded from the floppy disk in drive A is less than 06h, or if the first byte is greater than or equal to 06h, and the first nine words contain the same data pattern, this error message appears and the system stops:

```
602-Diskette Boot Record Error
```

15. If the disk was prepared with FORMAT or SYS using DOS 3.3 or an earlier version and the specified system files are not the first two files in the directory, or if a problem was encountered loading them, the following message appears:

```
Non-System disk or disk error
Replace and strike any key when ready
```

16. If the disk was prepared with FORMAT or SYS using DOS 3.3 or an earlier version and the boot sector is corrupt, you might see this message:

```
Disk Boot failure
```

17. If the disk was prepared with FORMAT or SYS using DOS 4.0 and later versions and the specified system files are not the first two files in the directory, or if a problem was encountered loading them, or the boot sector is corrupt, this message appears:

```
Non-System disk or disk error
Replace and press any key when ready
```

18. If no DOS volume boot sector can be read from drive A:, the BIOS looks for a master partition boot sector at cylinder 0, head 0, sector 1 (the very first sector) of the first fixed disk. If this sector is found, it is loaded into memory address 0000:7C00 and tested for a signature.

19. If the last two (signature) bytes of the master partition boot sector are not equal to 55AAh, software interrupt 18h (Int 18h) is invoked on most systems. On an IBM PS/2 system, a special character graphics message is displayed that depicts inserting a floppy disk in drive A: and pressing the F1 key. For non-PS/2 systems made by IBM, an Int 18h executes the ROM BIOS–based Cassette BASIC Interpreter. On any other IBM-compatible system, a message indicating some type of boot error is displayed. For example, systems with Phoenix AT ROM BIOS display this message:

```
No boot device available -
strike F1 to retry boot, F2 for setup utility
```

20. The master partition boot sector program searches its partition table for an entry with a system indicator byte indicating an extended partition. If the program finds such an entry, it loads the extended partition boot sector at the location indicated. The extended partition boot sector also has a table that is searched for another extended partition. If another extended partition entry is found, that extended partition boot sector is loaded from the location indicated, and the search continues until either no more extended partitions are indicated or the maximum number of 24 total partitions has been reached.

21. The master partition boot sector searches its partition table for a boot indicator byte marking an active partition.

22. On an IBM system, if none of the partitions is marked active (bootable), ROM BIOS–based Cassette BASIC is invoked. On most IBM-compatible systems, some type of disk error message is displayed.

23. If any boot indicator in the master partition boot record table is invalid, or if more than one indicates an active partition, the following message is displayed, and the system stops:

```
Invalid partition table
```

24. If an active partition is found in the master partition boot sector, the volume boot sector from the active partition is loaded and tested.

25. If the DOS volume boot sector cannot be read successfully from the active partition within five retries because of read errors, this message appears and the system stops:

```
Error loading operating system
```

26. The hard disk DOS volume boot sector is tested for a signature. If the DOS volume boot sector does not contain a valid signature of 55AAh as the last two bytes in the sector, this message appears and the system stops:

```
Missing operating system
```

27. The volume boot sector is executed as a program. This program checks the root directory to ensure that the first two files are IBMBIO.COM and IBMDOS.COM. If these files are present, they are loaded.

28. If the disk was prepared with FORMAT or SYS using DOS 3.3 or an earlier version and the specified system files are not the first two files in the directory, or if a problem is encountered loading them, the following message appears:

```
Non-System disk or disk error
Replace and strike any key when ready
```

29. If the disk was prepared with FORMAT or SYS using DOS 3.3 or an earlier version and the boot sector is corrupt, you might see this message:

```
Disk Boot failure
```

30. If the disk was prepared with FORMAT or SYS using DOS 4.0 or a later version and the specified system files are not the first two files in the directory, or if a problem is encountered loading them, or the boot sector is corrupt, the following message appears:

```
Non-System disk or disk error
Replace and press any key when ready
```

31. If no problems occur, the DOS volume boot sector executes IBMBIO.COM.

32. The initialization code in IBMBIO.COM copies itself into the highest region of contiguous DOS memory and transfers control to the copy. The initialization code copy then relocates IBMDOS over the portion of IBMBIO in low memory that contains the initialization code, because the initialization code no longer needs to be in that location.

33. The initialization code executes IBMDOS, which initializes the base device drivers, determines equipment status, resets the disk system, resets and initializes attached devices, and sets the system default parameters.

34. The full DOS filing system is active, and the IBMBIO initialization code is given back control.

35. The IBMBIO initialization code reads CONFIG.SYS four times.

36. During the first read, all the statements except DEVICE, INSTALL, and SHELL are read and processed in a predetermined order. Thus, the order of appearance for statements other than DEVICE, INSTALL, and SHELL in CONFIG.SYS is of no significance.

37. During the second read, DEVICE statements are processed in the order in which they appear, and any device driver files named are loaded and executed.

38. During the third read, INSTALL statements are processed in the order in which they appear, and the programs named are loaded and executed.

39. During the fourth and final read, the SHELL statement is processed and loads the specified command processor with the specified parameters. If the CONFIG.SYS file contains no SHELL statement, the default \COMMAND.COM processor is loaded with default parameters. Loading the command processor overwrites the initialization code in memory (because the job of the initialization code is finished).

40. If AUTOEXEC.BAT is present, COMMAND.COM loads and runs AUTOEXEC.BAT. After the commands in AUTOEXEC.BAT have been executed, the DOS prompt appears (unless the AUTOEXEC.BAT calls an application program or shell of some kind, in which case the user might operate the system without ever seeing a DOS prompt).
41. If no AUTOEXEC.BAT is present, COMMAND.COM executes the internal DATE and TIME commands, displays a copyright message, and displays the DOS prompt.

These are the steps performed by IBM AT systems, and most IBM-compatible systems closely emulate them. Some minor variations from this scenario are possible, such as those introduced by other ROM programs in the various adapters that might be plugged into a slot. Also, depending on the exact ROM BIOS programs involved, some of the error messages and sequences might vary. Generally, however, a computer follows this chain of events in "coming to life."

You can modify the system startup procedures by altering the CONFIG.SYS and AUTOEXEC.BAT files. These files control the configuration of DOS and allow special startup programs to be executed every time the system starts. The *User's Guide and Reference* that comes with DOS 5 has an excellent section on DOS configuration.

File Management

DOS uses several elements and structures to store and retrieve information on a disk. These elements and structures enable DOS to communicate properly with the ROM BIOS as well as application programs to process file storage and retrieval requests. Understanding these structures and how they interact helps you to troubleshoot and even repair these structures.

DOS File Space Allocation

DOS allocates disk space for a file on demand (space is not preallocated). The space is allocated one *cluster* (or allocation unit) at a time. A cluster is always one or more sectors. (For more information about sectors, refer to Chapter 14, "Hard Disk Drives and Controllers.")

The clusters are arranged on a disk to minimize head movement for multisided media. DOS allocates all the space on a disk cylinder before moving to the next cylinder. It does this by using the sectors under the first head, and then all the sectors under the next head, and so on until all sectors of all heads of the cylinder are used. The next sector used is sector 1 of head 0 on the next cylinder. (You find more information on floppy disks and drives in Chapter 13, "Floppy Disk Drives and Controllers," and on hard disks in Chapter 14, "Hard Disk Drives and Controllers.")

DOS version 2.x uses a simple algorithm when it allocates file space on a disk. Every time a program requests disk space, DOS scans from the beginning of the FAT until it finds a free cluster in which to deposit a portion of the file; then the search continues for the next cluster of free space, until all the file is written. This algorithm, called the *First Available Cluster algorithm*, causes any erased file near the beginning of the disk to be overwritten during the next write operation because those clusters would be the first

available to the next write operation. This system prevents recovery of that file and promotes file fragmentation because the first available cluster found is used regardless of whether the entire file can be written there. DOS simply continues searching for free clusters in which to deposit the remainder of the file.

The algorithm used for file allocation in DOS 3.0 and later versions is called the *Next Available Cluster algorithm*. In this algorithm, the search for available clusters in which to write a file starts not at the beginning of the disk, but rather from where the last write occurred. Therefore, the disk space freed by erasing a file is not necessarily reused immediately. Rather, DOS maintains a *Last Written Cluster pointer* indicating the last written cluster and begins its search from that point. This pointer is maintained in system RAM and is lost when the system is reset or rebooted, or when a disk is changed in a floppy drive.

In working with 360K drives, all versions of DOS always use the First Available Cluster algorithm because the Last Written Cluster pointer cannot be maintained for floppy disk drives that do not report a disk change (DC) signal to the controller, and because 360K drives do not supply the DC signal. With 360K floppy drives, therefore, DOS always assumes that the disk could have been changed, which flushes any buffers and resets the Last Written Cluster pointer.

The Next Available Cluster algorithm in DOS 3.0 and later versions is faster than the First Available Cluster algorithm and helps minimize fragmentation. Sometimes this type of algorithm is called *elevator seeking* because write operations occur at higher and higher clusters until the end of the disk area is reached. At that time, the pointer is reset, and writes work their way from the beginning of the disk again.

Files still end up becoming fragmented using the new algorithm, because the pointer is reset after a reboot, a disk change operation, or when the end of the disk is reached. Nevertheless, a great benefit of the newer method is that it makes unerasing files more likely to succeed even if the disk has been written to since the erasure, because the file just erased is not likely to be the target of the next write operation. In fact, it might be some time before the clusters occupied by the erased file are reused.

Even when a file is overwritten under DOS 3.0 and later versions, the clusters occupied by the file are not actually reused in the overwrite. For example, if you accidentally save on a disk a file using the same name as an important file that already exists, the existing file clusters are marked as available, and the new file (with the same name) is written to the disk in other clusters. It is possible, therefore, that the original copy of the file can still be retrieved. You can continue this procedure by saving another copy of the file with the same name, and each file copy is saved to higher numbered clusters, and each earlier version overwritten might still be recoverable on the disk. This process can continue until the system is rebooted or reset, or until the end of the available space is reached. Then the pointer is set to the first cluster, and previous file data is overwritten.

Because DOS always uses the first available directory entry when it saves or creates a file, the overwritten or deleted files whose data is still recoverable on the disk no longer appear in a directory listing. No commercial quick unerase or other unerase utilities therefore can find any record of the erased or overwritten file on the disk—true, of course, because these programs look only in the directory for a record of an erased file. Some newer undelete programs have a memory-resident delete tracking function that, in essence, maintains a separate directory listing from DOS. Unless an unerase program has a memory-resident delete tracking function, and that function has been activated before the deletion, no program can recall the files overwritten in the directory entry.

Because unerase programs do not look at the FAT, or at the data clusters themselves (unless they use delete tracking), they see no record of the files' existence. By scanning the free clusters on the disk one by one using a disk editor tool, you can locate the data from the overwritten or erased file and manually rebuild the FAT and directory entries. This procedure enables you to recover erased files even though files have been written to the disk since the erasure took place.

Interfacing to Disk Drives

DOS uses a combination of disk management components to make files accessible. These components differ slightly between floppies and hard disks and between disks of different sizes. They determine how a disk appears to DOS and to applications software. Each component used to describe the disk system fits as a layer into the complete system. Each layer communicates with the layer above and below it. When all the components work together, an application can access the disk to find and store data. Table 20.8 lists the DOS format specifications for floppy disks.

The four primary layers of interface between an application program running on a system and any disks attached to the system consist of software routines that can perform various functions, usually to communicate with the adjacent layers. These layers are shown in the following list:

- DOS Interrupt 21h (Int 21h) routines
- DOS Interrupt 25/26h (Int 25/26h) routines
- ROM BIOS disk Interrupt 13h (Int 13h) routines
- Disk controller I/O port commands

Each layer accepts various commands, performs different functions, and generates results. These interfaces are available for both floppy disk drives and hard disks, although the floppy disk and hard disk Int 13h routines differ widely. The floppy disk controllers and hard disk controllers are very different as well, but all the layers perform the same functions for both floppy disks and hard disks.

Table 20.8 Floppy Disk Format Specifications

	Current Formats		
Disk Size (in.)	**3 1/2"**	**3 1/2"**	**3 1/2"**
Disk Capacity (KB)	**2,880**	**1,440**	**720**
Media Descriptor Byte	F0h	F0h	F9h
Sides (Heads)	2	2	2
Tracks per Side	80	80	80
Sectors per Track	36	18	9
Bytes per Sector	512	512	512
Sectors per Cluster	2	1	2
FAT Length (Sectors)	9	9	3
Number of FATs	2	2	2
Root Dir. Length (Sectors)	15	14	7
Maximum Root Entries	240	224	112
Total Sectors per Disk	5,760	2,880	1,440
Total Available Sectors	5,726	2,847	1,426
Total Available Clusters	2,863	2,847	713

Interrupt 20h. The DOS Int 21h routines exist at the highest level and provide the most functionality with the least amount of work. For example, if an application program needs to create a subdirectory on a disk, it can call Int 21h, Function 39h. This function performs all operations necessary to create a subdirectory on the disk, including updating the appropriate directory and FAT sectors. The only information this function needs is the name of the subdirectory to create. DOS Int 21h would do much more work by using one of the lower-level access methods to create a subdirectory on the disk. Most applications programs you run access the disk through this level of interface.

Interrupt 25h and Int 26h. The DOS Int 25h and Int 26h routines provide much lower-level access to the disk than the Int 21h routines. Int 25h reads only specified sectors from a disk, and Int 26h only writes specified sectors to a disk. If you were to write a program that used these functions to create a subdirectory on a disk, the work required would be much greater than that required by the Int 21h method. For example, your program would have to perform all these tasks:

- Calculate exactly which directory and FAT sectors need to be updated
- Use Int 25h to read these sectors
- Modify the sectors appropriately to contain the new subdirectory information
- Use Int 26h to write the sectors back out

	Obsolete Formats			
5 1/4" 1,200	**5 1/4" 360**	**5 1/4" 320**	**5 1/4" 180**	**5 1/4" 160**
F9h	FDh	FFh	FCh	FEh
2	2	2	1	1
80	40	40	40	40
15	9	8	9	8
512	512	512	512	512
1	2	2	1	1
7	2	1	2	1
2	2	2	2	2
14	7	7	4	4
224	112	112	64	64
2,400	720	640	360	320
2,371	708	630	351	313
2,371	354	315	351	313

The number of steps would be even greater considering the difficulty in determining exactly what sectors have to be modified. According to Int 25/26h, the entire DOS-addressable area of the disk consists of sectors numbered sequentially from 0. A program designed to access the disk using Int 25h and Int 26h must know the location of everything by this sector number. A program designed this way might have to be modified to handle disks with different numbers of sectors or different directory and FAT sizes and locations. Because of all the overhead required to get the job done, most programmers would not choose to access the disk in this manner, and instead would use the higher-level Int 21h, which does all the work automatically.

Only disk- and sector-editing programs typically access a disk drive at the Int 25h and Int 26h level. Programs that work at this level of access can edit only areas of a disk that have been defined to DOS as a logical volume (drive letter). For example, DEBUG can read sectors from and write sectors to disks with this level of access.

Interrupt 13h. The next lower level of communications with drives, the ROM BIOS Int 13h routines, usually are found in ROM chips on the motherboard or on an adapter card in a slot; however, an Int 13h handler also can be implemented by using a device driver loaded at boot time. Because DOS requires Int 13h access to boot from a drive (and a device driver cannot be loaded until after boot-up), only drives with ROM BIOS–based Int 13h support can become bootable. Int 13h routines need to talk directly to the controller using the I/O ports on the controller. Therefore, the Int 13h code is very controller specific.

Table 20.9 lists the different functions available at the Interrupt 13h BIOS interface. Some functions are available to floppy drives or hard drives only, whereas others are available to both types of drives.

Table 20.9 Int 13h BIOS Disk Functions

Function	Floppy Disk	Hard Disk	Description
00h	×	×	Reset disk system.
01h	×	×	Get status of last operation.
02h	×	×	Read sectors.
03h	×	×	Write sectors.
04h	×	×	Verify sectors.
05h	×	×	Format track.
06h		×	Format bad track.
07h		×	Format drive.
08h	×	×	Read drive parameters.
09h		×	Initialize drive characteristics.
0Ah		×	Read long.
0Bh		×	Write long.
0Ch		×	Seek.
0Dh		×	Alternate hard disk reset.
0Eh		×	Read sector buffer.
0Fh		×	Write sector buffer.
10h		×	Test for drive ready.
11h		×	Recalibrate drive.
12h		×	Controller RAM diagnostic.
13h		×	Controller drive diagnostic.
14h		×	Controller internal diagnostic.
15h	×	×	Get disk type.
16h	×		Get floppy disk change status.
17h	×		Set floppy disk type for format.
18h	×		Set media type for format.
19h		×	Park hard disk heads.
1Ah		×	ESDI—Low-level format.
1Bh		×	ESDI—Get manufacturing header.
1Ch		×	ESDI—Get configuration.

Table 20.10 shows the error codes that may be returned by the BIOS INT 13h routines. In some cases, you may see these codes be referred to when running a low-level format program, disk editor, or other program that can directly access a disk drive through the BIOS.

Table 20.10 INT 13h BIOS Error Codes

Code	Description
00h	No error.
01h	Bad command.
02h	Address mark not found.
03h	Write protect.
04h	Request sector not found.
05h	Reset failed.
06h	Media change error.
07h	Initialization failed.
09h	Cross 64K DMA boundary.
0Ah	Bad sector flag detected.
0Bh	Bad track flag detected.
10h	Bad ECC on disk read.
11h	ECC corrected data error.
20h	Controller has failed.
40h	Seek operation failed.
80h	Drive failed to respond.
AAh	Drive not ready.
BBh	Undefined error.
CCh	Write fault.
0Eh	Register error.
FFh	Sense operation failed.

If you design your own custom disk controller device, you need to write an IBM-compatible Int 13h handler package and install it on the card using a ROM BIOS that will be linked into the system at boot time. To use Int 13h routines, a program must use exact cylinder, head, and sector coordinates to specify sectors to read and write. Accordingly, any program designed to work at this level must be intimately familiar with the parameters of the specific disk on the system on which it is designed to run. Int 13h functions exist to read the disk parameters, format tracks, read and write sectors, park heads, and reset the drive.

A low-level format program for ST-506/412 drives needs to work with disks at the Int 13h level or lower. Most ST-506/412 controller format programs work with access at the Int 13h level because virtually any operation a format program needs is available through the Int 13h interface. This is not true, however, for other types of controllers (such as IDE, SCSI, or ESDI), for which defect mapping and other operations differ considerably from the ST-506/412 types. Controllers that must perform special operations during a low-level format, such as defining disk parameters to override the motherboard ROM BIOS drive tables, would not work with any formatter that used only the standard Int 13h interface. For these reasons, most controllers require a custom formatter designed to bypass the Int 13h interface. Most general purpose, low-level reformat programs that

perform a nondestructive format (such as Norton Calibrate and SpinRite II) access the controller through the Int 13h interface (rather than going direct) and therefore cannot be used for an initial low-level format; the initial low-level format must be done by a controller-specific utility.

Few high-powered disk utility programs, other than some basic formatting software, can talk to the disk at the Int 13h level. The Kolod Research hTEST/hFORMAT utilities also can communicate at the Int 13h level, as can the DOS FDISK program. The Norton DISKEDIT and NU programs can communicate with a disk at the Int 13h level when these programs are in their absolute sector mode; they are two of the few utilities that can do so. These programs are important because they can be used for the worst data recovery situations, in which the partition tables have been corrupted. Because the partition tables as well as any non-DOS partitions exist outside the area of a disk that is defined by DOS, only programs that work at the Int 13h level can access them. Most utility programs for data recovery, such as the Mace Utilities MUSE program, work only at the DOS Int 25/26h level, which makes them useless for accessing areas of a disk outside of DOS domain.

Disk Controller I/O Port Commands. In the lowest level of interface, programs talk directly to the disk controller in the controller's own specific native language. To do this, a program must send controller commands through the I/O ports to which the controller responds. These commands are specific to the particular controller and sometimes differ even among controllers of the same type, such as different ESDI controllers. The ROM BIOS in the system must be designed specifically for the controller because the ROM BIOS talks to the controller at this I/O port level. Most manufacturer-type low-level format programs also need to talk to the controller directly because the higher-level Int 13h interface does not provide enough specific features for many of the custom ST-506/412 or ESDI and SCSI controllers on the market.

Figure 20.2 shows that most application programs work through the Int 21h interface, which passes commands to the ROM BIOS as Int 13h commands; these commands then are converted into direct controller commands by the ROM BIOS. The controller executes the commands and returns the results through the layers until the desired information reaches the application. This process enables applications to be written without worrying about such low-level system details, leaving such details up to DOS and the ROM BIOS. It also enables applications to run on widely different types of hardware, as long as the correct ROM BIOS and DOS support is in place.

Any software can bypass any level of interface and communicate with the level below it, but doing so requires much more work. The lowest level of interface available is direct communication with the controller using I/O port commands. As figure 20.2 shows, each different type of controller has different I/O port locations as well as differences among the commands presented at the various ports, and only the controller can talk directly to the disk drive.

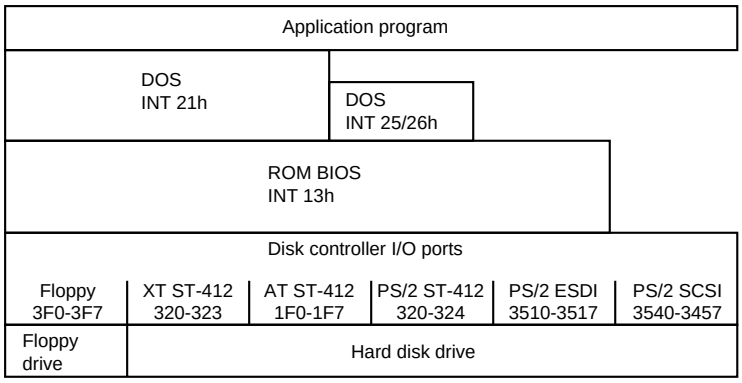

Fig. 20.2
Relationships between various interface levels.

If not for the ROM BIOS Int 13h interface, a unique DOS would have to be written for each available type of hard and floppy disk drive and disk. Instead, DOS communicates with the ROM BIOS using standard Int 13h function calls translated by the Int 13h interface into commands for the specific hardware. Because of the standard ROM BIOS interface, DOS can be written relatively independently of specific disk hardware and can support many different types of drives and controllers.

DOS Structures

To manage files on a disk and enable all application programs to see a consistent disk interface no matter what type of disk is used, DOS uses several structures. The following list shows all the structures and areas that DOS defines and uses to manage a disk, in roughly the order in which they are encountered on a disk:

- Master and extended partition boot sectors
- DOS volume boot sector
- Root directory
- File allocation tables (FAT)
- Clusters (allocation units)
- Data area
- Diagnostic read-and-write cylinder

A hard disk has all these DOS disk management structures allocated, and a floppy disk has all but the master and extended partition boot sectors and the diagnostic cylinder. These structures are created by the DOS FDISK program, which has no application on a floppy disk because floppy disks cannot be partitioned. Figure 20.3 is a simple diagram showing the relative locations of these DOS disk management structures on the 32M hard disk in an IBM AT Model 339.

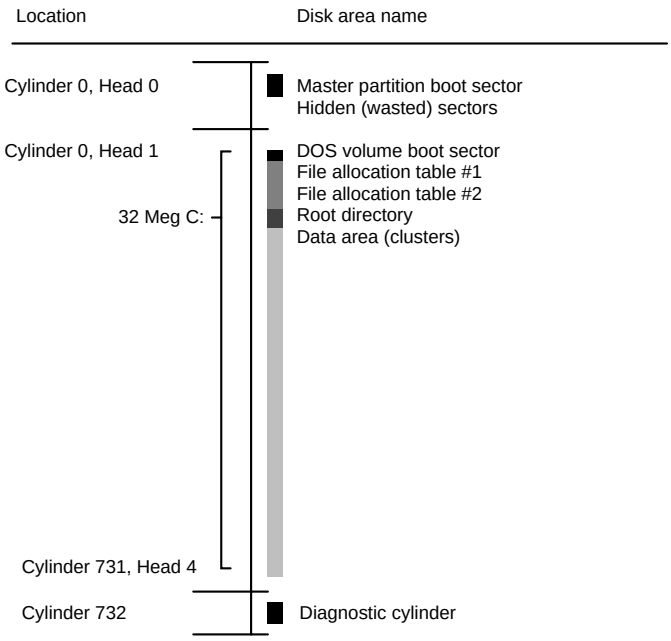

Fig. 20.3
DOS disk management structures on an IBM AT Model 339 32M hard disk.

Each disk area has a purpose and function. If one of these special areas is damaged, serious consequences can result. Damage to one of these sensitive structures usually causes a domino effect and limits access to other areas of the disk or causes further problems in using the disk. For example, DOS cannot read and write files if the FAT is damaged or corrupted. You therefore should understand these data structures well enough to be able to repair them when necessary. Rebuilding these special tables and areas of the disk is essential to the art of data recovery.

Master Partition Boot Sectors. To share a hard disk among different operating systems, the disk might be logically divided into one to four master partitions. Each operating system, including DOS (through versions 3.2), might own only one partition. DOS 3.3 and later versions introduced the extended DOS partition, which allows multiple DOS partitions on the same hard disk. With the DOS FDISK program, you can select the size of each partition. The partition information is kept in several partition boot sectors on the disk, with the main table embedded in the master partition boot sector. The master partition boot sector is always located in the first sector of the entire disk (cylinder 0, head 0, sector 1). The extended partition boot sectors are located at the beginning of each extended partition volume.

Each DOS partition contains a DOS volume boot sector as its first sector. With the DOS FDISK utility, you might designate a single partition as active (or bootable). The master partition boot sector causes the active partition's volume boot sector to receive control when the system is started or reset. Additional master disk partitions can be set up for

Novell NetWare and for OS/2 HPFS, PCIX (UNIX), XENIX, CP/M-86, or other operating systems. None of these foreign operating system partitions can be accessible under DOS, nor can any DOS partitions normally be accessible under other operating systems. (OS/2 and DOS share FAT partitions, but high-performance file system [HPFS] partitions are exclusive to OS/2.)

A hard disk must be partitioned to be accessible by an operating system. You must partition a disk even if you want to create only a single partition. Table 20.11 shows the format of the Master Boot Record (MBR) with partition tables.

Table 20.11 Master Boot Record (Partition Table)

Offset	Length	Description
Partition Table Entry #1		
1BEh 446	1 byte	Boot Indicator Byte (80h = Active, else 00h)
1BFh 447	1 byte	Starting Head (or Side) of Partition
1C0h 448	16 bits	Starting Cylinder (10 bits) and Sector (6 bits)
1C2h 450	1 byte	System Indicator Byte (see table 20.12)
1C3h 451	1 byte	Ending Head (or Side) of Partition
1C4h 452	16 bits	Ending Cylinder (10 bits) and Sector (6 bits)
1C6h 454	1 dword	Relative Sector Offset of Partition
1CAh 458	1 dword	Total Number of Sectors in Partition
Partition Table Entry #2		
1CEh 462	1 byte	Boot Indicator Byte (80h = Active, else 00h)
1CFh 463	1 byte	Starting Head (or Side) of Partition
1D0h 464	16 bits	Starting Cylinder (10 bits) and Sector (6 bits)
1D2h 466	1 byte	System Indicator Byte (see table 20.12)
1D3h 467	1 byte	Ending Head (or Side) of Partition
1D4h 468	16 bits	Ending Cylinder (10 bits) and Sector (6 bits)
1D6h 470	1 dword	Relative Sector Offset of Partition
1DAh 474	1 dword	Total Number of Sectors in Partition
Partition Table Entry #3		
1DEh 478	1 byte	Boot Indicator Byte (80h = Active, else 00h)
1DFh 479	1 byte	Starting Head (or Side) of Partition
1E0h 480	16 bits	Starting Cylinder (10 bits) and Sector (6 bits)
1E2h 482	1 byte	System Indicator Byte (see table 20.12)
1E3h 483	1 byte	Ending Head (or Side) of Partition
1E4h 484	16 bits	Ending Cylinder (10 bits) and Sector (6 bits)
1E6h 486	1 dword	Relative Sector Offset of Partition
1EAh 490	1 dword	Total Number of Sectors in Partition

(continues)

Table 20.11 Continued

Offset	Length	Description
Partition Table Entry #4		
1EEh 494	1 byte	Boot Indicator Byte (80h = Active, else 00h)
1EFh 495	1 byte	Starting Head (or Side) of Partition
1F0h 496	16 bits	Starting Cylinder (10 bits) and Sector (6 bits)
1F2h 498	1 byte	System Indicator Byte (see table 20.12)
1F3h 499	1 byte	Ending Head (or Side) of Partition
1F4h 500	16 bits	Ending Cylinder (10 bits) and Sector (6 bits)
1F6h 502	1 dword	Relative Sector Offset of Partition
1FAh 506	1 dword	Total Number of Sectors in Partition
Signature Bytes		
1FEh 510	2 bytes	Boot Sector Signature (55AAh)

A WORD equals two bytes read in reverse order, and a DWORD equals two WORDs read in reverse order.

Table 20.12 shows the standard values and meanings of the System Indicator Byte.

Table 20.12 Partition Table System Indicator Byte Values

Value	Description
00h	No allocated partition in this entry
01h	Primary DOS, 12-bit FAT (Partition < 16M)
04h	Primary DOS, 16-bit FAT (16M <= Partition <= 32M)
05h	Extended DOS (Points to next Primary Partition)
06h	Primary DOS, 16-bit FAT (Partition > 32M)
07h	OS/2 HPFS Partition
02h	MS-XENIX Root Partition
03h	MS-XENIX usr Partition
08h	AIX File System Partition
09h	AIX Boot Partition
50h	Ontrack Disk Manager READ-ONLY Partition
51h	Ontrack Disk Manager READ/WRITE Partition
56h	Golden Bow Vfeature Partition
61h	Storage Dimensions Speedstor Partition
63h	IBM 386/ix or UNIX System V/386 Partition
64h	Novell NetWare Partition
75h	IBM PCIX Partition

Value	Description
DBh	Digital Research Concurrent DOS/CPM-86 Partition
F2h	Some OEM's DOS 3.2+ Second Partition
FFh	UNIX Bad Block Table Partition

DOS Volume Boot Sectors. The *volume boot sector* is the first sector on any area of a drive addressed as a volume (or logical DOS disk). On a floppy disk, for example, this sector is the first one on the floppy disk because DOS recognizes the floppy disk as a volume with no partitioning required. On a hard disk, the volume boot sector or sectors are located as the first sector within any disk area allocated as a nonextended partition, or any area recognizable as a DOS volume.

This special sector resembles the master partition boot sector in that it contains a program as well as some special data tables. The first volume boot sector on a disk is loaded by the system ROM BIOS for floppies or by the master partition boot sector on a hard disk. This program is given control; it performs some tests and then attempts to load the first DOS system file (IBMBIO.COM). The volume boot sector is transparent to a running DOS system; it is outside the data area of the disk on which files are stored.

You create a volume boot sector with the DOS FORMAT command (high-level format). Hard disks have a volume boot sector at the beginning of every DOS logical drive area allocated on the disk, in both the primary and extended partitions. Although all the logical drives contain the program area as well as a data table area, only the program code from the volume boot sector in the active partition on a hard disk is executed. The others are simply read by the DOS system files during boot-up to obtain their data table and determine the volume parameters.

The volume boot sector contains program code and data. The single data table in this sector is called the *media parameter block* or *disk parameter block*. DOS needs the information it contains to verify the capacity of a disk volume as well as the location of important features such as the FAT. The format of this data is very specific. Errors can cause problems with booting from a disk or with accessing a disk. Some non-IBM OEM versions of DOS have not adhered to the standards set by IBM for the format of this data, which can cause interchange problems with disks formatted by different versions of DOS. The later versions can be more particular, so if you suspect that boot sector differences are causing inability to access a disk, you can use a utility program such as DOS DEBUG or Norton Utilities to copy a boot sector from the newer version of DOS to a disk formatted by the older version. This step should enable the new version of DOS to read the older disk and should not interfere with the less particular older version. This has never been a problem in using different DOS versions from the same OEM, but might occur in mixing different OEM versions.

Table 20.13 shows the format and layout of the DOS Boot Record (DBR).

Table 20.13 DOS Boot Record (DBR) Format

Offset Hex	Offset Dec	Field Length	Description
00h	0	3 bytes	Jump Instruction to Boot Program Code
03h	3	8 bytes	OEM Name and DOS Version (“IBM 5.0”)
0Bh	11	1 word	Bytes / Sector (usually 512)
0Dh	13	1 byte	Sectors / Cluster (Must be a power of 2)
0Eh	14	1 word	Reserved Sectors (Boot Sectors, usually 1)
10h	16	1 byte	FAT Copies (usually 2)
11h	17	1 word	Maximum Root Directory Entries (usually 512)
13h	19	1 word	Total Sectors (If Partition <= 32M, else 0)
15h	21	1 byte	Media Descriptor Byte (F8h for Hard Disks)
16h	22	1 word	Sectors / FAT
18h	24	1 word	Sectors / Track
1Ah	26	1 word	Number of Heads
1Ch	28	1 dword	Hidden Sectors (If Partition <= 32M, 1 word only)
For DOS 4.0 or Higher Only, Else 00h			
20h	32	1 dword	Total Sectors (If Partition > 32M, else 0)
24h	36	1 byte	Physical Drive No. (00h=floppy, 80h=hard disk)
25h	37	1 byte	Reserved (00h)
26h	38	1 byte	Extended Boot Record Signature (29h)
27h	39	1 dword	Volume Serial Number (32-bit random number)
2Bh	43	11 bytes	Volume Label (“NO NAME“ stored if no label)
36h	54	8 bytes	File System ID (“FAT12“ or “FAT16“)
For All Versions of DOS			
3Eh	62	450 bytes	Boot Program Code
1FEh	510	2 bytes	Signature Bytes (55AAh)

A WORD is two bytes read in reverse order, and a DWORD is two WORDs read in reverse order.

Root Directory. A *directory* is a simple database containing information about the files stored on a disk. Each record in this database is 32 bytes long, and no delimiters or separating characters are between the fields or records. A directory stores almost all the information that DOS knows about a file: name, attribute, time and date of creation, size, and where the beginning of the file is located on the disk. (The information a directory does *not* contain about a file is where the file continues on the disk and whether the file is contiguous or fragmented. The FAT contains that information.)

Two basic types of directories exist: the *root directory* and *subdirectories*. They differ primarily in how many there can be and in where they can be located. Any given volume can have only one root directory, and the root directory is always stored on a disk in a fixed location immediately following the two FAT copies. Root directories vary in size because of the varying types and capacities of disks, but the root directory of a given disk is fixed. After a root directory is created, it has a fixed length and cannot be extended to hold more entries. Normally, a hard disk volume has a root directory with room for 512 total entries. Subdirectories are stored as files in the data area of the disk and therefore have no fixed length limits.

Every directory, whether it is the root directory or a subdirectory, is organized in the same way. A directory is a small database with a fixed record length of 32 bytes. Entries in the database store important information about individual files and how files are named on a disk. The directory information is linked to the FAT by the starting cluster entry. In fact, if no file on a disk were longer than one single cluster, the FAT would be unnecessary. The directory stores all the information needed by DOS to manage the file, with the exception of all the clusters that the file occupies other than the first one. The FAT stores the remaining information about other clusters the file uses.

To trace a file on a disk, you start with the directory entry to get the information about the starting cluster of the file and its size. Then you go to the file allocation table. From there, you can follow the chain of clusters the file occupies until you reach the end of the file.

DOS Directory entries are 32 bytes long and are in the format shown in table 20.14.

Table 20.14 DOS Directory Format

Offset			
Hex	**Dec**	**Field Length**	**Description**
00h	0	8 bytes	File name
08h	8	3 bytes	File extension
0Bh	11	1 byte	File attributes
0Ch	12	10 bytes	Reserved (00h)
16h	22	1 word	Time of creation
18h	24	1 word	Date of creation
1Ah	26	1 word	Starting cluster
1Ch	28	1 dword	Size in bytes

Note

File names and extensions are left-justified and padded with spaces (32h). The first byte of the file name indicates the file status as follows:

Hex	File Status
00h	Entry never used; entries past this point not searched.
05h	Indicates first character of file name is actually E5h.
E5h	"σ" (lowercase sigma) indicates file has been erased.
2Eh	"." (period) indicates this entry is a directory. If the second byte is also 2Eh, the cluster field contains the cluster number of parent directory (0000h if the parent is the root).

Table 20.15 describes the DOS Directory file attribute byte.

Table 20.15 DOS Directory File Attribute Byte

Bit Positions 7 6 5 4 3 2 1 0	Hex Value	Description
0 0 0 0 0 0 0 1	01h	Read-only file
0 0 0 0 0 0 1 0	02h	Hidden file
0 0 0 0 0 1 0 0	04h	System file
0 0 0 0 1 0 0 0	08h	Volume label
0 0 0 1 0 0 0 0	10h	Subdirectory
0 0 1 0 0 0 0 0	20h	Archive (updated since backup)
0 1 0 0 0 0 0 0	40h	Reserved
1 0 0 0 0 0 0 0	80h	Reserved
Examples		
0 0 1 0 0 0 0 1	21h	Read-only, archive
0 0 1 1 0 0 1 0	32h	Hidden, subdirectory, archive
0 0 1 0 0 1 1 1	27h	Read-only, hidden, system, archive

File Allocation Tables (FATs). The file allocation table (FAT) is a table of number entries describing how each cluster is allocated on the disk. The data area of the disk has a single entry for each cluster. Sectors in the nondata area on the disk are outside the range of the disk controlled by the FAT. The sectors involved in any of the boot sectors, file allocation table, and root directory are outside the range of sectors controlled by the FAT.

The FAT does not manage every data sector specifically, but rather allocates space in groups of sectors called *clusters* or *allocation units*. A cluster is one or more sectors designated by DOS as allocation units of storage. The smallest space a file can use on a disk is one cluster; all files use space on the disk in integer cluster units. If a file is one byte

larger than one cluster, two clusters are used. DOS determines the size of a cluster when the disk is high-level formatted by the DOS FORMAT command.

You can think of the FAT as a sort of spreadsheet that controls the cluster use of the disk. Each cell in the spreadsheet corresponds to a single cluster on the disk; the number stored in that cell is a sort of code telling whether the cluster is used by a file and, if so, where the next cluster of the file is located.

The numbers stored in the FAT are hexadecimal numbers that are either 12 or 16 bits long. The 16-bit FAT numbers are easy to follow because they take an even two bytes of space and can be edited fairly easily. The 12-bit numbers are 1 1/2 bytes long, which presents a problem when most disk sector editors show data in byte units. To edit the FAT, you must do some hex/binary math to convert the displayed byte units to FAT numbers. Fortunately (unless you are using the DOS DEBUG program), most of the available tools and utility programs have a FAT editing mode that automatically converts the numbers for you. Most of them also show the FAT numbers in decimal form, which most people find easier to handle.

The DOS FDISK program determines whether a 12-bit or 16-bit FAT is placed on a disk, even though the FAT is written during the high-level format (FORMAT). All floppy disks use a 12-bit FAT, but hard disks can use either. On hard disk volumes with more than 16 megabytes (32,768 sectors), DOS creates a 16-bit FAT; otherwise, DOS creates a 12-bit FAT.

DOS keeps two copies of the FAT. Each one occupies contiguous sectors on the disk, and the second FAT copy immediately follows the first. Unfortunately, DOS uses the second FAT copy only if sectors in the first FAT copy become unreadable. If the first FAT copy is corrupted, which is a much more common problem, DOS does not use the second FAT copy. Even the DOS CHKDSK command does not check or verify the second FAT copy. Moreover, whenever DOS updates the first FAT, large portions of the first FAT automatically are copied to the second FAT. If, therefore, the first copy was corrupted and then subsequently updated by DOS, a large portion of the first FAT would be copied over the second FAT copy, damaging it in the process. After the update, the second copy is usually a mirror image of the first one, complete with any corruption. Two FATs rarely stay out of sync for very long. When they are out of sync and DOS writes to the disk and causes the first FAT to be updated, it also causes the second FAT to be overwritten by the first FAT. Because of all this, the usefulness of the second copy of the FAT is limited to manual repair operations, and even then it is useful only if the problem is caught immediately, before DOS has a chance to update the disk.

Clusters (Allocation Units). The term *cluster* was changed to *allocation unit* in DOS 4.0. The newer term is appropriate because a single cluster is the smallest unit of the disk that DOS can handle when it writes or reads a file. A cluster is equal to one or more sectors, and although a cluster can be a single sector, it is usually more than one. Having more than one sector per cluster reduces the size and processing overhead of the FAT and enables DOS to run faster because it has fewer individual units of the disk to manage. The trade-off is in wasted disk space. Because DOS can manage space only in full cluster units, every file consumes space on the disk in increments of one cluster.

Table 20.16 shows default cluster (or allocation unit) sizes used by DOS for the various floppy disk formats.

Table 20.16 Default Floppy Disk Cluster (Allocation Unit) Sizes

Drive Type	Cluster (Allocation Unit) Size
5 1/4-inch 360K	2 sectors (1,024 bytes)
5 1/4-inch 1.2M	1 sector (512 bytes)
3 1/4-inch 720K	2 sectors (1,024 bytes)
3 1/4-inch 1.44M	1 sector (512 bytes)
3 1/4-inch 2.88M	2 sectors (1,024 bytes)

It seems strange that the high-density disks, which have many more individual sectors than low-density disks, sometimes have smaller cluster sizes. The larger the FAT, the more entries DOS must manage, and the slower DOS seems to function. This sluggishness is due to the excessive overhead required to manage all the individual clusters; the more clusters to be managed, the slower things become. The trade-off is in the minimum cluster size.

Smaller clusters generate less *slack* (space wasted between the actual end of each file and the end of the cluster). With larger clusters, the wasted space grows larger. High-density floppy drives are faster than their low-density counterparts, so perhaps IBM and Microsoft determined that the decrease in cluster size balances the drive's faster operation and offsets the use of a larger FAT.

For hard disks, the cluster size can vary greatly among different versions of DOS and different disk sizes. Table 20.17 shows the cluster sizes IBM and most other OEM versions of DOS select for a particular volume size.

Table 20.17 Default Hard Disk Cluster (Allocation Unit) Sizes

Hard Disk Volume Size	Cluster (Allocation Unit) Size	FAT type
0M to less than 16M	8 sectors or 4,096 bytes	12-bit
16M through 128M	4 sectors or 2,048 bytes	16-bit
Over 128M through 256M	8 sectors or 4,096 bytes	16-bit
Over 256M through 512M	16 sectors or 8,192 bytes	16-bit
Over 512M through 1,024M	32 sectors or 16,384 bytes	16-bit
Over 1,024M through 2,048M	64 sectors or 32,768 bytes	16-bit

In most cases, these cluster sizes, selected by the DOS FORMAT command, are the minimum possible for a given partition size. Therefore, 8K clusters are the smallest possible for a partition size of greater than 256M. Although most non-IBM OEM versions of DOS work like the IBM version, some versions might use cluster sizes different from what this table indicates. For example, Compaq DOS 3.31 shifts to larger cluster sizes much earlier

than IBM DOS does. Compaq DOS shifts to 4K clusters at 64M partitions, 8K clusters at 128M partitions, and 16K clusters at 256M partitions. A 305M partition that uses 8K clusters under IBM DOS has clusters of 16K under Compaq DOS 3.31.

The effect of these larger cluster sizes on disk use can be substantial. A drive containing about 5,000 files, with average slack of one-half of the last cluster used for each file, wastes about 20 megabytes [5000*(.5*8)K] of file space on a disk set up with IBM DOS or MS-DOS. Using Compaq DOS 3.31, this wasted space doubles to 40 megabytes for the same 5,000 files. Someone using a system with Compaq DOS 3.31 could back up, repartition, and reformat with IBM DOS, and after restoring all 5,000 files, gain 20 megabytes of free disk space.

Compaq DOS 3.31 does not use the most efficient (or smallest) cluster size possible for a given partition size because its makers were interested in improving the performance of the system at the expense of great amounts of disk space. Larger cluster sizes get you a smaller FAT, with fewer numbers to manage; DOS overhead is reduced when files are stored and retrieved, which makes the system seem faster. For example, the CHKDSK command runs much faster on a disk with a smaller FAT. Unfortunately, the trade-off for speed here is a tremendous loss of space on the disk. (Compaq DOS 4.0 and 5.0 use IBM DOS and MS-DOS conventions.)

The Data Area. The data area of a disk is the area that follows the boot sector, file allocation tables, and root directory on any volume. This area is managed by the FAT and the root directory. DOS divides it into allocation units sometimes called clusters. These clusters are where normal files are stored on a volume.

Diagnostic Read-and-Write Cylinder. The FDISK partitioning program always reserves the last cylinder of a hard disk for use as a special diagnostic read-and-write test cylinder. That this cylinder is reserved is one reason FDISK always reports fewer total cylinders than the drive manufacturer states are available. DOS (or any other operating system) does not use this cylinder for any normal purpose, because it lies outside the partitioned area of the disk.

On systems with IDE, SCSI, or ESDI disk interfaces, the drive and controller might allocate an additional area past the logical end of the drive for a bad-track table and spare sectors for replacing bad ones. This situation may account for additional discrepancies between FDISK and the drive manufacturer.

The diagnostics area enables diagnostics software such as the manufacturer-supplied Advanced Diagnostics disk to perform read-and-write tests on a hard disk without corrupting any user data. Low-level format programs for hard disks often use this cylinder as a scratch-pad area for running interleave tests or preserving data during nondestructive formats. This cylinder is also sometimes used as a head landing or parking cylinder on hard disks that do not have an automatic parking facility.

Potential DOS Upgrade Problems

You already know that the DOS system files have special placement requirements on a hard disk. Sometimes these special requirements cause problems when you are upgrading from one version of DOS to another.

If you have attempted to upgrade a PC system from one version of DOS to another, you know that you use the DOS SYS command to replace old system files with new ones. The SYS command copies the existing system files (stored on a bootable disk with hidden, system, and read-only attributes) to the disk in the correct position and with the correct names and attributes. The COPY command does not copy hidden or system files (nor would it place the system files in the required positions on the destination disk if their other attributes had been altered so that they could be copied using COPY).

In addition to transferring the two hidden system files from one disk to another, SYS also updates the DOS volume boot sector on the destination disk so that it is correct for the new version of DOS. Common usage of the SYS command is as follows:

SYS C: (for drive C)

or

SYS A: (to make a floppy in drive A bootable)

The syntax of the command is as follows:

SYS[*d:*][*path*]***d*:**

In this command line, *d:/path* specifies an optional source drive and path for the system files. If the source drive specification is omitted, the boot drive is used as the source drive. This parameter is supported in DOS 4.0 and later versions only. Versions of DOS older than 4.0 automatically look for system files on the default drive (not on the boot drive). The ***d*:** in the syntax specifies the drive to which you want to transfer the system files.

When the SYS command is executed, you usually are greeted by one of two messages:

```
System transferred
```

or

```
No room for system on destination disk
```

If a disk has data on it before you try to write the system files to it, the SYS command from DOS versions 3.3 and earlier probably will fail because these versions are not capable of moving other files out of the way. The SYS command in DOS 4.0 and higher versions rarely fails because these versions can and do move files out of the way.

Some users think that the cause of the `No room` message on a system that has an older version of DOS is that the system files in any newer version of DOS are always larger than the previous version, and that the new version files cannot fit into the space allocated for older versions. Such users believe that the command fails because this space cannot be provided at the beginning without moving other data away. This belief is wrong. The SYS command fails in these cases because you are trying to install a version of DOS that has file names different from the names already on the disk. There is no normal reason for the SYS command to fail when you update the system files on a disk that already has those files.

Although the belief that larger system files cannot replace smaller ones might be popular, it is wrong for DOS 3.0 and later versions. The system files can be placed virtually anywhere on the disk, except that the first clusters of the disk must contain the file IBMBIO.COM (or its equivalent). After that requirement is met, the IBMDOS.COM file might be fragmented and placed just about anywhere on the disk, and the SYS command implements it with no problems whatsoever. In version 3.3 or later, even the IBMBIO.COM file can be fragmented and spread all over the disk, as long as the first cluster of the file occupies the first cluster of the disk (cluster 2). The only other requirement is that the names IBMBIO.COM and IBMDOS.COM (or their equivalents) must use the first and second directory entries.

DOS 4.0 and Later Versions. Under DOS 4.0 and later versions, the SYS command is much more powerful than under previous versions. Because the system files must use the first two entries in the root directory of the disk as well as the first cluster (cluster 2) of the disk, the DOS 4.0 and later versions' SYS command moves any files that occupy the first two entries but that do not match the new system file names to other available entries in the root directory. The SYS command also moves the portion of any foreign file occupying the first cluster to other clusters on the disk. Whereas the SYS command in older versions of DOS would fail and require a user to make adjustments to the disk, the DOS 4.0 and later versions' SYS command automatically makes the required adjustments. For example, even if you are updating a Phoenix DOS 3.3 disk to IBM DOS 4.0, the IBM DOS SYS command relocates the Phoenix IO.SYS and MSDOS.SYS files so that the new IBMBIO.COM and IBMDOS.COM files can occupy the correct locations in the root directory as well as on the disk.

DOS 5.0 and 6.0. The SYS command in DOS versions 5.0 and 6.0 goes one step farther: it replaces old system files with the new ones. Even if the old system files had other names, DOS 5.0 and higher ensure that they are overwritten by the new system files. If you are updating a disk on which the old system file names match the new ones, the SYS command of any version of DOS overwrites the old system files with the new ones with no moving of files necessary. With the enhanced SYS command in DOS 4.0 and later versions, it is difficult to make a DOS upgrade fail.

DOS 3.3. The DOS 3.3 SYS command does not move other files out of the way (as SYS does in DOS 4.0 and later versions); therefore, you must ensure that the first two root directory entries are either free or contain names that match the new system file names. As in DOS 4.0 and later versions, the first cluster on the disk must contain the first portion of IBMBIO.COM; unlike DOS 4.0 and later versions, however, the SYS command under DOS 3.3 does not move any files for you. Necessary manual adjustments, such as clearing the first two directory entries or relocating a file that occupies the first cluster on the disk, must be done with whatever utility programs you have available. The DOS 3.3 system files can be fragmented and occupy various areas of the disk.

SYS under DOS 3.3 does not automatically handle updating from one version of DOS to a version that has different system file names. In that case, because the system file names are not the same, the new system files do not overwrite the old ones. If you are making this kind of system change, use a directory editing tool to change the names of the current system files to match the new names so that the system file overwrite can occur.

DOS 3.2. DOS 3.2 or earlier requires that the entire IBMBIO.COM file be contiguous starting with cluster 2 (the first cluster) on the disk. The other system file (IBMDOS.COM) can be fragmented or placed anywhere on the disk; it does not have to follow the first system file physically on the disk.

DOS 2.x. DOS 2.x requires that both system files (IBMBIO.COM and IBMDOS.COM) occupy contiguous clusters on the disk starting with the first cluster (cluster 2). The DOS 2.1 system files are slightly larger than the DOS 2.0 files in actual bytes of size, but the size change is not enough to require additional clusters on the disk. A SYS change from DOS 2.0 to DOS 2.1, therefore, is successful in most cases.

Upgrading DOS from the Same OEM. Updating from one version of DOS to a later version from the same OEM by simply using the SYS command has never been a problem. I verified this with IBM DOS by installing IBM DOS 2.0 on a system with a hard disk through the normal FORMAT /S command. I copied all the subsequent DOS transient programs into a \DOS subdirectory on the disk and then updated the hard disk, in succession, to IBM DOS 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 5.0, and even 6.3 using nothing more than the SYS and COPY (or XCOPY or REPLACE) commands. Between each version change, I verified that the hard disk would boot the new version of DOS with no problems. Based on this experiment, I have concluded that you never have to use the FORMAT command to update one DOS version to a later version, as long as both versions are from the same OEM. I also verified the same operations on a floppy disk. Starting with a bootable floppy disk created by IBM DOS 2.0, I used SYS and COPY to update that disk to all subsequent versions of DOS through 6.3 without ever reformatting it. After each version change, the floppy disk was bootable with no problems.

You should be able to update a bootable hard disk or floppy disk easily from one DOS version to another without reformatting the disk. If you are having problems, you probably are attempting to upgrade to a version of DOS that uses names for the system files different from those used by the existing DOS, which means that you are moving from a DOS made by one OEM to a DOS made by a different company. If you are having trouble and this is not the case, carefully examine the list of requirements at the beginning of this section. Your problem must be that one of those requirements is not being met.

Downgrading DOS. One important and often overlooked function of the SYS command is its capability to update the DOS volume boot sector of a disk on which it is writing system files. Later versions of SYS are more complete than earlier versions in the way they perform this update; therefore, using SYS to go from a later version of DOS to an earlier version is sometimes difficult. For example, you cannot use SYS to install DOS 2.1 on a disk that currently boots DOS 3.0 and later versions. Changing from DOS 4.0 or higher versions back to DOS 3.3 usually works, if the partition is less than or equal to 32 megabytes in capacity. You probably will never see a problem with a later version of SYS updating a DOS volume boot sector created by an earlier version, but earlier versions might leave something out when they attempt to change back from a later version. Fortunately, few people ever attempt to install a lower version of DOS over a higher version.

Known Bugs in DOS

Few things are more frustrating than finding out that software you depend on every day has bugs. It's even worse when DOS does. Every version of DOS ever produced has had bugs, and users must learn to anticipate them. Some problems are never resolved; you must live with them.

Sometimes the problems are severe enough, however, that Microsoft, IBM, and other OEM distributors of DOS issue a patch disk that corrects the problems. If you use IBM DOS, you can get them from an IBM dealer or download them from the IBM National Support Center (NSC) BBS (the number is in Appendix B, the "Vendor List"). With MS-DOS, you can request a patch by calling the technical support number in the front of your DOS manual. Or, if you have a modem, you can download patches from the Microsoft Download Service BBS (see Appendix B).

If you have IBM PC DOS, check with your system vendor periodically to find out whether patches are available. You do not have to go to the dealer from which you purchased PC DOS; any dealer must provide the patches for free when you show you have a legal license for PC DOS. The proof-of-license page from the PC DOS 4.0 manual satisfies as a license check. If you ask a dealer who does not know about these patches, or who does not provide them for some reason, try another dealer. PC DOS is a warranted product, and the patches are part of the warranty service.

The following sections detail IBM patches for PC DOS 3.3, PC DOS 4.0, and IBM DOS 5.0. These versions have official IBM-produced patch disks available at no cost from your nearest IBM dealer, or available from the IBM BBS.

PC DOS 3.3 Bugs and Patches. The PC DOS 3.3 official patches and fixes from IBM originally were issued by IBM's National Support Center on September 9, 1987. A second update, issued October 24, 1987, superseded the first update. These disks fix the following two general problems with DOS 3.3:

- BACKUP did not work properly in backing up a large number of subdirectories in a given directory. A new version of BACKUP was created to resolve this problem.
- Systems that had slow serial printers with small input buffers sometimes displayed a false `Out Of Paper` error message when attempting to print. A new program, I17.COM, resolves this problem.

In addition to the two general problems resolved by this patch, IBM PS/2 systems had a particular problem between their ROM BIOS and DOS 3.3; a special DASDDRVR.SYS driver was provided on the patch disk to fix these BIOS problems. The versions of DASDDRVR.SYS supplied on the DOS 3.3 patch disks have been superseded by later versions supplied elsewhere; DASDDRVR.SYS was placed on the IBM PS/2 Reference disks for more widespread distribution, and you can obtain it directly from IBM on a special system update floppy disk. This driver and the problems it can correct are discussed later in this chapter, in the section "PS/2 BIOS Update."

PC DOS 4.0 and 4.01 Bugs and Patches. Six different versions of IBM DOS 4.0 have been issued, counting the first version and the five patch disks subsequently released. The disks are not called patch disks anymore; they are called *corrective service disks* (CSDs). Each level of CSD contains all the previous level CSDs. The first CSD issued for PC DOS 4.0 (UR22624) contained a series of problem fixes that later were incorporated into the standard release version of DOS 4.01. Several newer CSDs have been released since version 4.01 appeared. Unfortunately, these more recent updates were never integrated into the commercially packaged DOS. The only way to obtain these fixes is to obtain the CSDs from your dealer or from the IBM BBS.

The VER command in any level of IBM DOS 4.x always shows 4.00, which causes much confusion about which level of CSD fixes are installed on a specific system. To eliminate this confusion and allow for the correct identification of installed patches, the CSD UR29015 and later levels introduce to DOS 4.x a new command: SYSLEVEL. This command is resident in COMMAND.COM and is designed to identify conclusively to the user the level of corrections installed. On a system running PC DOS 4.x with CSD UR35284 installed, the SYSLEVEL command reports the following:

DOS Version: 4.00 U.S. Date: 06/17/88

CSD Version: UR35284 U.S. Date: 09/20/91

The following list notes each of the IBM DOS 4.0 Corrective Service Diskettes (CSDs) and when they first became available.

CSD	Date Available
UR22624	08/15/88 (this equals 4.01)
UR24270	03/27/89
UR25066	05/10/89
UR29015	03/20/90
UR31300	06/29/90
UR35284	09/20/91

These CSDs are valid only for the IBM version of DOS—PC DOS 4.0. Microsoft and OEM versions of DOS may not have corresponding patches. Some OEMs provide patches or corrections in different ways, and some may not even offer them. Because most OEMs released their versions of DOS after IBM, other manufacturers have had a chance to incorporate fixes in their standard version and may not need to provide patches. If you have a version of DOS by a manufacturer other than Microsoft or IBM, contact its source to find out which patch corrections have been applied to your version of DOS. With a system that can run standard MS-DOS, you can get patches from Microsoft. If you have an IBM, you must rely on a reputable dealer to get the latest version of DOS.

MS-DOS 4. Microsoft released its version of DOS 4 after IBM had fixed most of the bugs in PC DOS 4.0. Yet MS-DOS 4.01 introduced some bugs of its own. A patch for MS-DOS 4.x is available for download from the Microsoft Download Service or by calling

Microsoft technical service. The patch disk for MS-DOS 4.0x is available on the Microsoft Download service as PD0255.EXE.

IBM DOS 5.0 Bugs and Patches. With the release of its DOS version 5.0, IBM changed the product name from PC DOS to IBM DOS (version 6.0 of IBM's DOS has been changed back to PC DOS). IBM DOS version 5.0 has several CSDs that fix a couple of problems. The most significant is a defect in the XCOPY command that causes it to fail sometimes when it uses the /E or /S switches. The following list notes the IBM DOS 5.0 CSDs and indicates when they first became available:

CSD	Date available
UR35423	08/91
UR35748	10/91
UR35834	11/91
UR36603	02/92
UR37387	09/92

Table 20.18 lists the problems fixed by these patch disks.

Table 20.18 IBM DOS 5 Corrective Service Disks (CSDs)

CSD	Item	Problem
UR35423	XCOPY	Wrong output when using /E and /S switches.
UR35423	QBASIC	Enables QBASIC and QEDIT compatibility.
UR35748	SYS	Corrupted hard file after installing UR35423.
UR35834	DOSSHELL	DOSSHELL takes 27 seconds to load.
UR35834	MEUTOINI	4.0 .MEU to 5.0 .INI conversion incomplete.
UR35834	MEM	MEM switch hangs system with PC3270.
UR35834	IBMBIO	L40SX will not SUSPEND or RESUME.
UR35834	DOSSHELL	Can't edit dialog box if length is maximum.
UR35834	EMM386	Int 19H fails with EMM386 and DOS=HIGH.
UR35834	FORMAT	FORMAT on unpartitioned drive; rc = 0.
UR35834	REPLACE	REPLACE /A returns error.
UR35834	XCOPY	XCOPY /S incorrectly sets error level.
UR35834	GRAPHICS	PrtScr of graphics display produces garbage.
UR35834	DOSSHELL	DOSSHELL.INI corrupted from CTRL-ALT-DEL.
UR35834	BACKUP	BACKUP calls wrong FORMAT.COM from OS/2.
UR35834	MIRROR	MIRROR doesn't enable Interrupts correctly.
UR35834	BACKUP	BACKUP /A backs up too large a file.
UR36603	EDIT	Alt ### key combo doesn't work in EDIT.
UR36603	MIRROR	MIRROR fails with /T switch and DOS=UMB.
UR36603	IBMBIO	L40SX will not SUSPEND or RESUME if DOS=LOW.

(continues)

Table 20.18 Continued

CSD	Item	Problem
UR36603	BACKUP	BACKUP fails to back up all files.
UR36603	QBASIC	QBASIC help msgs missing in nls versions.
UR36603	CHKDSK	CHKDSK data loss when sectors per FAT>256.
UR36603	EMM386	DMA transfer may not function on EISA systems.
UR36603	RECOVER	RECOVER can corrupt disks with 12-bit FAT.
UR36603	RECOVER	RECOVER may not adjust file size correctly.
UR36603	DOSSHELL	DOSSHELL incorrectly copies certain file sizes.
UR36603	DOSSWAP	Task Swapper destroys CX register.
UR36603	DOSSWAP	Task Swapper does not swap EMS memory.
UR36603	DOSSWAP	Task Swapper incorrectly swaps large XMS memory.
UR36603	DOSSWAP	DOSSHELL overwrites an interrupt vector.
UR36603	DOSSWAP	DOSSHELL random skip of swapping application memory.
UR36603	DOSSHELL	DOSSHELL uses environment var to set 2nd swap path.
UR37387	RESTORE	RESTORE fails to display backup files.
UR37387	IBMDOS	FASTOPEN causes bad FAT message.
UR37387	KEYB	Pause key doesn't work on PS/2 25 and 30.
UR37387	IBMDOS	Unmapped network drive returns error.
UR37387	MODE	MODE and off-line printer across net hangs.
UR37387	IBMDOS	RAMDRIVE errors with certain combinations.
UR37387	DOSSHELL	Unattended start mode—lose keyboard with DOSSHELL.
UR37387	COMMAND	Error with greater than 1G free space.
UR37387	IBMDOS	INT 27 returns no data after file create.
UR37387	IBMBIO	L40SX loses time during suspend.
UR37387	DOSSHELL	CTRL+ESC hangs when returning to DOSSHELL.
UR37387	IBMDOS	Extended File Open returns incorrect code.
UR37387	HIMEM	Device driver fails to load.
UR37387	HIMEM	HIMEM incorrectly identifies memory on EISA.
UR37387	UNDELETE	Doesn't work if partition is a multiple of 128M.
UR37387	UNDELETE	Doesn't handle foreign characters correctly.
UR37387	BACKUP	Restore does not ask for second diskette.
UR37387	KEYBOARD	Keyboard changes for Latin II countries.
UR37387	MEM	Finland MEM/C displays invalid characters.
UR37387	DOSSHELL	INT 33 DOSSHELL reentry problem with BASIC.
UR37387	MODE	MODE LPT1:, P reports Bad mode.
UR37387	BACKUP	Occasionally gets `Cannot Restore File` error.

MS-DOS 5.0 Bugs and Patches. As mentioned earlier, the Microsoft Download Service DOS files listing includes fixes for MS-DOS versions 4.0 and 5.0. When you call the Microsoft Download Service, you are asked to enter your name and city and to choose a password. Choose a password you will not forget because when you realize how simple it is to always have the most current bug fixes for MS-DOS, you will want to call back. After you enter your name and a password, the following screen appears:

```
************************************
****Microsoft Download Service****
****     Main Menu     ****
[I]nstructions on Using This Service
[D]ownload File
[F]ile Index

[W]indows 3.1 Driver Library Update
[N]ew Files & Complete File Listing

[M]icrosoft Information
[A]lter User Settings
[U]tilities - Comments
[L]ength of Call
[E]xit ... Logoff the System
[H]elp - System Instructions
Command:
```

At the command prompt choose **f** (for file) to display the following screen:

```
************************************
****  File Sections Available  ****
************************************
[1] Windows and MS-DOS
[2] Word, Excel and Multiplan
[3] PowerPoint, Publisher and Project
[4] LAN Manager and Microsoft Mail
[5] Languages and Windows SDK
[6] Works and Flight Simulator
[7] MS FOX, MS Access & MS Money
[8] MS Video for Windows

[F]ile Search

[-]Previous Menu
[M]ain Menu

[L]ength of Call
[E]xit ... Logoff the System

Command:
```

Choose **1** (for Windows and MS-DOS) to display the following screen:

```
Windows and MS-DOS Files
[1] Windows for Workgroups Appnotes
[2] Windows 3.1 Driver Library
[3] Windows 3.1 Application Notes
[4] Windows 3.1 Resource Kit
[5] Windows 3.0 Driver Library-SDL
[6] Windows 3.0 Application Notes
[7] Windows 3.0 Resource Kit
[8] MS DOS Files

[-]Previous Menu
[M]ain Menu

[L]ength of Call
[E]xit ... Logoff the System

Command:
```

Choose **8** (for MS-DOS files).

MS-DOS 5.0 fixes available on the Microsoft Download Service are as follows:

PD0445.EXE	Replacement DOSSWAP.EXE
PD0455.EXE	ADAPTEC.SYS Driver (for adaptec drives)
PD0488.EXE	8514.VID Video Driver for MS-DOS 5.0 Shell
PD0489.EXE	MS-DOS Messages Reference
PD0495.EXE	PRINTFIX.COM patch for PRINT problems
PD0646.EXE	Updated CHKDSK.EXE and UNDELETE.EXE
PD0315.EXE	BACKUP/RESTORE Supplemental Utilities

MS-DOS 6.0 and IBM PC DOS 6.1. As of this writing, there has been one bug fix for MS-DOS version 6.0—a new version of SmartDrive, which fixes a problem on some systems when SmartDrive is used at the same time as DoubleSpace. The fix can be used if you have experienced cross-linked files you think are related to upgrading to DOS 6.0. You can download the file, named PD0805.EXE, from the Microsoft Download Service. The Microsoft Download Service also enables you (free of charge) to download the MS-DOS 6.0 Supplemental Disk. This supplemental disk contains utilities helpful to the handicapped. The disk also contains new versions of various utilities that were part of DOS 5.0, but not included on the DOS 6.0 upgrade disks, including MIRROR, EDLIN, ASSIGN, JOIN, BACKUP (MSBACKUP is a new menu-based backup program included with DOS 6), COMP, PRINTER.SYS, and the DVORAK keyboard. Also available is a utility to fix a problem using the Shift key in Quick Basic 1.1, named PD0415.EXE.

In addition, numerous technical papers on setting up and running MS-DOS 6.0 are available on the download service. They are listed in table 20.19.

Table 20.19 MS-DOS Technical Papers

File Name	Subject
PD0456.TXT	Running MS-DOS in the High Memory Area
PD0457.TXT	HIMEM.SYS "ERROR: Unable to Control A20 Line"
PD0459.TXT	EMM386.EXE: No Expanded Memory Available
PD0460.TXT	Running Both Extended and Expanded Memory
PD0462.TXT	Mouse Doesn't Work with MS-DOS Shell
PD0463.TXT	Using the Setver Command
PD0465.TXT	Problems Formatting or Reading a Floppy Disk
PD0470.TXT	System Fails When You Are Using EMM.386
PD0471.TXT	Explanation of the WINA20.386 File
PD0473.TXT	Installing MS-DOS from Drive B
PD0474.TXT	Windows 3.0 Doesn't Run in 386 Enhanced Mode
PD0476.TXT	IBM PS/1 Fails After MS-DOS Is Installed
PD0477.TXT	Setup Stops Before Completing Upgrade to MS
PD0743.TXT	MS-DOS 6.0 Installation and Partition Q&A
PD0744.TXT	MS-DOS 6.0 General Installation Q&A
PD0745.TXT	DoubleSpace Questions and Answers
PD0746.TXT	MemMaker Questions and Answers
PD0747.TXT	MS-DOS 6.0 Configuration Q&A
PD0748.TXT	Backup and Miscellaneous Q&A
PD0771.TXT	Repartitioning Your Hard Disk To Upgrade to MS-DOS 6.0
PD0785.TXT	Upgrading DR DOS to MS-DOS 6

IBM has since released PC DOS 6.3, which has many fixes and updates. You can get a free upgrade from the IBM BBS if you have earlier versions of IBM DOS.

PS/2 BIOS Update (DASDDRVR.SYS)

The DASDDRVR.SYS (direct access storage device driver) file is a set of software patches that fixes various ROM BIOS bugs in several models of the IBM PS/2. DASDDRVR.SYS is required for specific PS/2 systems using IBM's PC DOS versions 3.30 or later, to correct several bugs in the IBM PS/2 ROM BIOS. Before IBM's PC DOS 4.00 was released, conflicting information indicated that PC DOS 4.00 would include the updates to correct the PS/2 ROM BIOS problems fixed by DASDDRVR.SYS under PC DOS 3.30. This information was not accurate, however. In fact, an IBM PS/2 system needs DASDDRVR.SYS with IBM DOS 5.00 (or any higher version of DOS) even with the most current corrective service disk (CSD) update.

The PS/2 needs the DASDDRVR.SYS fixes only in the DOS environment. Some users assume, therefore, that the PS/2 problems with DOS are DOS bugs; they are not. The DASDDRVR.SYS program is provided on the PS/2 Reference disk (included with every PS/2 system) and is available separately on a special PS/2 system update disk. The disks contain the device driver program (DASDDRVR.SYS) and an installation program.

The PS/2 ROM BIOS bugs in the following list are fixed by DASDDRVR.SYS (the problem numbers are shown in table 20.19, and more detailed information is provided later in this section):

1. Failures occur in reading some 720K program floppy disks (Models 8530, 8550, 8560, and 8580).
2. Intermittent `Not ready` or `General failure` error messages appear (Models 8550, 8560, and 8580).
3. 3 1/2-inch floppy disk format fails when user tries to format more than one floppy disk (Models 8550, 8560, 8570, and 8580).
4. Combined 301 and 8602 error messages appear at power-on or after power interruption (Models 8550 and 8560).
5. System clock loses time, or combined 162 and 163 errors appear during system initialization (Models 8550 and 8560).
6. User is unable to install Power-On Password program with DASDDRVR.SYS installed (Models 8550, 8560, and 8580).
7. Devices attached to COM2:, COM3:, or COM4: are not detected (Model 8530).
8. Devices that use Interrupt Request level 2 (IRQ2) fail (Model 8530).
9. System performance degradation occurs from processor intensive devices (Models 8550, 8555, and 8560).
10. Error occurs in a microcode routine that enhances long-term reliability of 60/120M disk drives (Models 8550, 8555, 8570, and 8573).
11. Time and date errors occur when user resets the time or date. Intermittent date changes occur when the system is restarted by pressing Ctrl-Alt-Del (Model 8530).

If you are an IBM PS/2-system user running PC DOS 3.3 or higher and experiencing any of these problems, load the DASDDRVR.SYS file. The problems are system specific, and DASDDRVR.SYS fixes the problems for only the systems listed. IBM requires its dealers to distribute the System Update disk containing DASDDRVR.SYS to anyone who requests it. Neither the dealer nor the customer pays a fee for the System Update disk. You also can obtain copies directly from IBM by calling (800) IBM-PCTB (800-426-7282) and ordering the PS/2 System Update disk.

Check table 20.20 for detailed descriptions of each of these problems and for the specific systems affected. Models not listed for a particular problem do not need DASDDRVR.SYS, and no benefit results from installing it.

Table 20.20 DASDDRVR.SYS Version Summary

Version	File Size	Problems Fixed	Source
1.10	648 bytes	1-3	DOS 3.3 Fix Disk (08/24/87)
1.20	698 bytes	1-5	Reference Disk, DOS 3.3 Fix Disk (09/09/87)
1.30	734 bytes	1-6	Reference Disk
1.56	1170 bytes	1-9	Reference Disk (03/90), System Update Disk 1.01 (part number 64F1500)
1.56	3068 bytes	1-11	Reference Disk (xx/xx), System Update Disk 1.02 (part number 04G3288)

The first three problem fixes originally were provided by the DASDDRVR.SYS version 1.10 file supplied on the first PC DOS 3.3 fix disk. Fixes for problems 4 and 5 were added in DASDDRVR.SYS version 1.20, included on all IBM PS/2 Reference disks (30-286, 50/60, and 70/80) version 1.02 or later, as well as on an updated version of the PC DOS 3.3 fix disk. The fix for problem 6 was added in DASDDRVR.SYS version 1.30, included on all 50/60 and 70/80 Reference disks version 1.03 or later. Fixes for problems 7 through 9 were added to DASDDRVR.SYS version 1.56, included on all IBM PS/2 Reference disks dated March 1990 or later. This version of DASDDRVR.SYS also was available separately, on the IBM PS/2 System Update disk version 1.01. The latest DASDDRVR.SYS (also called version 1.56, but dated January 1991) can be found on newer Reference disks or on the IBM PS/2 System Update disk version 1.02.

By using the DASDDRVR.SYS driver file, IBM can correct specific ROM BIOS problems and bugs without having to issue a new set of ROM chips for a specific system. Using this file eliminates service time or expense in fixing simple problems, but causes the inconvenience of having to load the driver. The driver does not consume memory, nor does it remain in memory (as does a typical driver or memory-resident program); it either performs functions on boot only and then terminates, or overlays existing code or tables in memory, thereby consuming no additional space. Because DASDDRVR.SYS checks the exact ROM BIOS by model, submodel, and revision, it performs functions only on those for which it is designed. If it detects a BIOS that does not need fixing, the program terminates without doing anything. You can load DASDDRVR.SYS on any system; it functions only on systems for which it is designed.

Because a system BIOS occasionally needs revising or updating, IBM used disk-based BIOS programs for most newer PS/2 systems. The Models 57, P75, 90, and 95, in fact, load the system BIOS from the hard disk every time the system is powered up, during a procedure called *initial microcode load* (IML). You can get a ROM upgrade for these systems by obtaining a new Reference disk and loading the new IML file on the hard disk. This system makes DASDDRVR.SYS or other such patches obsolete.

Installing DASDDRVR.SYS. To install DASDDRVR.SYS, you must update the CONFIG.SYS file with the following entry and restart the system:

DEVICE=[*d:**path*\]DASDDRVR.SYS

The drive and path values must match the location and name of the DASDDRVR.SYS file on your system.

Detailed Problem Descriptions. This section gives a detailed description of the problems corrected by the most current release of DASDDRVR.SYS and indicates for which systems the corrections are necessary.

1. Failures occur in reading some 720K program disks.

   IBM PS/2 systems affected:

   Model 30 286 8530-E01, -E21

   Model 50 8550-021

   Model 60 8560-041, -071

   Model 80 8580-041, -071

   Intermittent read failures on some 720K original application software disks. Example: `Not ready reading drive A:` appears when a user attempts to install an application program. Attempting to perform DIR or COPY commands from the floppy disk also produces the error message.

2. Intermittent `Not ready` or `General failure` error messages are displayed.

   IBM PS/2 systems affected:

   Model 50 8550-021

   Model 60 8560-041, -071

   Model 80 8580-041, -071

   A very intermittent problem with a floppy disk drive `Not ready` or a fixed disk `General failure` message. This problem can be aggravated by certain programming practices that mask off (or disable) interrupts for long periods. The update ensures that interrupts are unmasked on each disk or floppy disk request.

3. A 3 1/2-inch floppy disk format fails when the user tries to format more than one disk.

   IBM PS/2 systems affected:

   Model 50 8550-021

   Model 60 8560-041, -071

   Model 70 8570-Axx (all)

   Model 80 8580-Axx (all)

The DOS FORMAT command fails when a user tries to format multiple 3 1/2-inch disks. The failure appears as an `Invalid media` or `Track 0 bad—disk unusable` message when the user replies **Y**es to the prompt `Format another (Y/N)?` after the format of the first disk is complete. The error message appears when the user tries to format the second disk. If a system is booted from a floppy disk, the problem does not occur. This problem occurs only with DOS 3.3, not with later versions.

4. Combined 301 and 8602 error messages at power-on or after power interruption.

   IBM PS/2 systems affected:

   Model 50 8550-021

   Model 60 8560-041, -071

   When power is interrupted momentarily or a system is otherwise switched on and off quickly, a 301 (keyboard) and 8602 (pointing device) error message may appear during the Power On Self Test (POST). This error occurs because the system powers on before the keyboard is ready. The problem is more likely to occur if the system was reset previously by pressing Ctrl-Alt-Del.

5. System clock loses time or combined 162 and 163 errors during system initialization.

   IBM PS/2 systems affected:

   Model 50 8550-021

   Model 60 8560-041, -071

   Intermittent 162 (CMOS checksum or configuration) and 163 (clock not updating) Power-On Self Test (POST) errors occur. Various time-of-day problems on specified IBM PS/2 Model 50 systems; for example, the user turns on the machine in the morning and finds the time set to the same time the machine was turned off the day before.

6. User is unable to install Power-On Password program with DASDDRVR.SYS installed.

   IBM PS/2 systems affected:

   Model 50 8550-021

   Model 60 8560-041, -071

   Model 80 8580-041, -071

   When a user tries to install the Power-On Password feature with DASDDRVR.SYS version 1.3 or earlier installed, a message appears that states (incorrectly) that a password already exists. The user also may be prompted for a password (on warm boot), even though password security has not been implemented.

7. Devices attached to COM2:, COM3:, or COM4: are not detected.

   IBM PS/2 systems affected:

   Model 30 286 8530-E01, -E21

8. Devices that use Interrupt Request level 2 (IRQ2) fail.

   IBM PS/2 systems affected:

   Model 30 286 8530-E01, -E21

9. System performance degradation occurs from processor intensive devices.

   IBM PS/2 systems affected:

   Model 50 8550-021, -031, -061

   Model 55 SX 8555-031, -061

   Model 60 8560-041, -071

10. Error occurs in a microcode routine that enhances long-term reliability of 60/120M disk drives.

    IBM PS/2 systems affected:

    Model 50 8550-061

    Model 55 SX 8555-061

    Model 70 8570-061, -121, -A61, -A21, -B61, -B21

    Model P70 8573-061, -121

11. Time and date errors occur when the user resets the time or date. Intermittent date changes occur when the system is restarted by pressing Ctrl-Alt-Del.

    IBM PS/2 systems affected:

    Model 30 8530 (all)

OS/2 versions 1.2 and earlier contained these BIOS fixes in the form of a .BIO file for each specific BIOS needing corrections. These files were automatically loaded by OS/2 at boot time, depending on the specific system on which it was being loaded.

OS/2 versions 1.3 and later contain the fixes directly in-line in the system files, not as separate files. When a system running one of these OS/2 versions is booted, OS/2 determines the model, submodel, and revision bytes for the particular BIOS under which it is running. Based on this information, OS/2 determines the correct .BIO file to load or the correct in-line code to run. For example, the IBM PS/2 55SX BIOS is Model F8, Submodel 0C, Revision 00, which causes IBM OS/2 version 1.2 to load the file F80C00.BIO automatically during boot-up. OS/2 versions 1.3 or later use this information to run the proper fix code contained in the system files. This procedure enables execution of only those BIOS fixes necessary for this particular system.

Any symptom described as being resolved by the DASDDRVR.SYS update may have other causes. If you install the DASDDRVR.SYS update and continue to have problems, consider the errors valid and follow normal troubleshooting procedures to find the causes.

DOS Disk and Data Recovery

The CHKDSK, RECOVER, and DEBUG commands are the DOS damaged disk recovery team. These commands are crude, and their actions sometimes are drastic, but at times they are all that is available or needed. RECOVER is best known for its function as a data recovery program, and CHKDSK usually is used for inspection of the file structure. Many users are unaware that CHKDSK can implement repairs to a damaged file structure. DEBUG, a crude, manually controlled program, can help in the case of a disk disaster, if you know exactly what you are doing.

The CHKDSK Command. The useful and powerful DOS CHKDSK command also is generally misunderstood. To casual users, the primary function of CHKDSK seems to be providing a disk space allocation report for a given volume and a memory allocation report. CHKDSK does those things, but its primary value is in discovering, defining, and repairing problems with the DOS directory and FAT system on a disk volume. In handling data recovery problems, CHKDSK is a valuable tool, although it is crude and simplistic compared to some of the after-market utilities that perform similar functions.

The output of the CHKDSK command which runs on a typical (well maybe not typical, but from my own) hard disk is as follows:

```
Volume 4GB_SCSI    created 08-31-1994 5:05p
Volume Serial Number is 1882-18CF

2,146,631,680 bytes total disk space
      163,840 bytes in 3 hidden files
   16,220,160 bytes in 495 directories
  861,634,560 bytes in 10,355 user files
1,268,613,120 bytes available on disk

       32,768 bytes in each allocation unit
       65,510 total allocation units on disk
       38,715 available allocation units on disk

      655,360 total bytes memory
      632,736 bytes free
```

A little known CHKDSK function is reporting a specified file's (or files') level of fragmentation. CHKDSK also can produce a list of all files (including hidden and system files) on a particular volume, similar to a super DIR command. By far, the most important CHKDSK capabilities are its detection and correction of problems with the DOS file management system.

The name of the CHKDSK program is misleading: it seems to be a contraction of CHECK DISK. The program does not actually check a disk, or even the files on a disk, for integrity. CHKDSK cannot even truly show how many bad sectors are on a disk, much less locate and mark them. The real function of CHKDSK is to inspect the directories and FATs to see whether they correspond with each other or contain discrepancies. CHKDSK

does not detect (and does not report on) damage in a file; it checks only the FAT and directory areas (the table of contents) of a disk. Rather than CHKDSK, the command should have been called CKDIRFAT (for CHECK DIRECTORY FAT) because its most important job is to verify that the FATs and directories correspond with one another. The name of the program gives no indication of the program's capability to repair problems with the directory and FAT structures.

CHKDSK also can test files for contiguity. Files loaded into contiguous tracks and sectors of a disk or floppy disk naturally are more efficient. Files spread over wide areas of the disk make access operations take longer. DOS always knows the location of all of a file's fragments by using the pointer numbers in the file allocation table (FAT). These pointers are data that direct DOS to the next segment of the file. Sometimes, for various reasons, these pointers might be lost or corrupted and leave DOS incapable of locating some portion of a file. Using CHKDSK can alert you to this condition and even enable you to reclaim the unused file space for use by another file.

CHKDSK Command Syntax. The syntax of the CHKDSK command is as follows:

CHKDSK [*d*:*path*\] [*filename*] [*/F*] [*/V*]

The *d:* specifies the disk volume to analyze. The *path*\ and *filename* options specify files to check for fragmentation in addition to the volume analysis. Wild cards are allowed in the file-name specification, to include as many as all the files in a specified directory for fragmentation analysis. One flaw with the fragmentation analysis is that it does not check for fragmentation across directory boundaries, only within a specified directory.

The switch */F* (Fix) enables CHKDSK to perform repairs if it finds problems with the directories and FATs. If */F* is not specified, the program is prevented from writing to the disk, and all repairs were not really performed.

The switch */V* (Verbose) causes the program to list all the entries in all the directories on a disk and give detailed information in some cases when errors are encountered.

The drive, path, and file specifiers are optional. If no parameters are given for the command, CHKDSK processes the default volume or drive and does not check files for contiguity. If you specify [*path*] and [*filename*] parameters, CHKDSK checks all specified files to see whether they are stored contiguously on the disk. One of two messages is displayed as a result:

```
All specified file(s) are contiguous
```

or

```
[filename] Contains xxx non-contiguous blocks
```

The second message is displayed for each file fragmented on the disk and displays the number of fragments the file is in. A *fragmented file* is one that is scattered around the disk in pieces rather than existing in one contiguous area of the disk. Fragmented files are slower to load than contiguous files, which reduces disk performance. Fragmented files are also much more difficult to recover if a problem with the FAT or directory on the disk occurs.

Utility programs that can defragment files are discussed in Chapter 19. But if you have only DOS, you have several possible ways to accomplish a full defragmentation. To defragment files on a floppy disk, you can format a new floppy disk and use COPY or XCOPY to copy all the files from the fragmented disk to the replacement. For a hard disk, you must completely back up, format, and then restore the disk. This procedure on a hard disk is time-consuming and dangerous, which is why so many defragmenting utilities have been developed.

CHKDSK Limitations. In several instances, CHKDSK operates only partially or not at all. CHKDSK does not process volumes or portions of volumes that have been created as follows:

- SUBST command volumes
- ASSIGN command volumes
- JOIN command subdirectories
- Network volumes

SUBST Problems. The SUBST command creates a virtual volume, which is actually an existing volume's subdirectory using another volume specifier (drive letter) as an alias. To analyze the files in a subdirectory created with SUBST, you must give the TRUENAME or actual path name to the files. TRUENAME is an undocumented command in DOS 4.0 and later versions that shows the actual path name for a volume that has been created by the SUBST command.

You also can use the SUBST command to find out the TRUENAME of a particular volume. Suppose that you use SUBST to create volume E: from the C:\AUTO\SPECS directory, as follows:

```
C:\>SUBST E: C:\AUTO\SPECS
```

After entering the following two commands to change to the E: volume and execute a CHKDSK of the volume and files there, you see the resulting error message:

```
C:\>E:

E:\>CHKDSK *.*

Cannot CHKDSK a SUBSTed or ASSIGNed drive
```

To run CHKDSK on the files on this virtual volume E:, you must find the actual path the volume represents. You can do so by entering the SUBST command (with no parameters):

```
E:\>SUBST

E: => C:\AUTO\SPECS
```

You also can find the actual path with the undocumented TRUENAME command (in DOS 4.0 and later versions only), as follows:

```
E:\>TRUENAME E:

C:\AUTO\SPECS
```

After finding the path to the files, you can issue the appropriate CHKDSK command to check the volume and files:

```
E:\>CHKDSK C:\AUTO\SPECS\*.*


Volume 4GB_SCSI    created 08-31-1994 5:05p
Volume Serial Number is 1882-18CF

2,146,631,680 bytes total disk space
      163,840 bytes in 3 hidden files
   16,220,160 bytes in 495 directories
  861,634,560 bytes in 10,355 user files
1,268,613,120 bytes available on disk

       32,768 bytes in each allocation unit
       65,510 total allocation units on disk
       38,715 available allocation units on disk

      655,360 total bytes memory
      632,736 bytes free

All specified file(s) are contiguous
```

ASSIGN Problems. Similarly, CHKDSK does not process a disk drive that has been altered by the ASSIGN command. For example, if you have given the command ASSIGN A=B, you cannot analyze drive A: unless you first unassign the disk drive with the ASSIGN command, that is, ASSIGN A=A.

JOIN Problems. CHKDSK does not process a directory tree section created by the JOIN command (which joins a physical disk volume to another disk volume as a subdirectory), nor does it process the actual joined physical drive because such a drive is an invalid drive specification, according to DOS. On volumes on which you have used the JOIN command, CHKDSK processes the actual portion of the volume and then displays this warning error message:

```
Directory is joined
tree past this point not processed
```

This message indicates that the command cannot process the directory on which you have used JOIN. CHKDSK then continues processing the rest of the volume and outputs the requested volume information.

Network Problems. CHKDSK does not process a networked (shared) disk on either the server or workstation side. In other words, at the file server, you cannot use CHKDSK on any volume that has any portion of itself accessible to remote network stations. At any network station, you can run CHKDSK only on volumes physically attached to that specific station and not on any volume accessed through the network software. If you attempt to run CHKDSK from a server or a workstation on a volume shared on a network, you see this error message:

```
Cannot CHKDSK a network drive
```

If you want to run CHKDSK on the volume, you must go to the specific PC on which the volume physically exists and suspend or disable any sharing of the volume during the CHKDSK.

CHKDSK Command Output. CHKDSK normally displays the following information about a disk volume:

- Volume name and creation date
- Volume serial number
- Number of bytes in total disk space
- Number of files and bytes in hidden files
- Number of files and bytes in directories
- Number of files and bytes in user files
- Number of bytes in bad sectors (unallocated clusters)
- Number of bytes available on disk
- Number of bytes in total memory (RAM)
- Number of bytes in free memory
- Error messages if disk errors are encountered

By using optional parameters, CHKDSK also can show the following:

- Names and number of fragments in noncontiguous files
- Names of all directories and files on disk

If a volume name or volume serial number does not exist on a particular volume, that information is not displayed. If no clusters are marked as bad in the volume's FAT, CHKDSK returns no display of bytes in bad sectors.

For example, suppose that a disk was formatted under DOS 6.2 with the following command:

```
C:\>FORMAT A: /F:720 /U /S /V:floppy_disk
```

The output of the FORMAT command looks like this:

```
Insert new diskette for drive A:
and press ENTER when ready...

Formatting 720K
Format complete.
System transferred


      730,112 bytes total disk space
      135,168 bytes used by system
      594,944 bytes available on disk

        1,024 bytes in each allocation unit.
          581 allocation units available on disk.

Volume Serial Number is 266D-1DDC

Format another (Y/N)?
```

The status report at the end of the FORMAT operation is similar to the output of the CHKDSK command. The output of the CHKDSK command when run on this disk would appear as follows:

```
C:\>CHKDSK A:

Volume FLOPPY_DISK created 01-16-1994 10:18p
Volume Serial Number is 266D-1DDC

      730,112 bytes total disk space
       79,872 bytes in 2 hidden files
       55,296 bytes in 1 user files
      594,944 bytes available on disk

        1,024 bytes in each allocation unit
          713 total allocation units on disk
          581 available allocation units on disk

      655,360 total bytes memory
      632,736 bytes free
```

In this case, CHKDSK shows the volume name and serial number information because the FORMAT command placed a volume label on the disk with the /V parameter, and FORMAT under DOS 4.0 and later versions automatically places the volume serial number on a disk. Note that three total files are on the disk, two of which have the HIDDEN attribute. DOS versions earlier than 5.0 report the Volume Label “FLOPPY_DISK” as a third hidden file. To see the names of the hidden files, you can execute the CHKDSK command with the /V parameter, as follows:

```
C:\>CHKDSK A: /V

Volume FLOPPY_DISK created 01-16-1994 10:18p
Volume Serial Number is 266D-1DDC
Directory A:\
A:\IO.SYS
```

```
A:\MSDOS.SYS
A:\COMMAND.COM

      730,112 bytes total disk space
       79,872 bytes in 2 hidden files
       55,296 bytes in 1 user files
      594,944 bytes available on disk

        1,024 bytes in each allocation unit
          713 total allocation units on disk
          581 available allocation units on disk

      655,360 total bytes memory
      632,736 bytes free
```

With the /V parameter, CHKDSK lists the names of all directories and files across the entire disk, which in this example is only three total files. CHKDSK does not identify which of the files are hidden, it simply lists them all. Note that the DIR command in DOS versions 5.0 and higher can specifically show hidden files with the /AH parameter. The DOS system files are the first two files on a bootable disk and normally have HIDDEN, SYSTEM, and READ-ONLY attributes. After listing how many bytes are used by the hidden and normal files, CHKDSK lists how much total space is available on the disk.

If you are using DOS 4.0 or a later version, CHKDSK also tells you the size of each allocation unit (or cluster), the total number of allocation units present, and the number not currently being used.

Finally, CHKDSK counts the total amount of conventional memory or DOS-usable RAM (in this case, 640K or 655,360 bytes) and displays the number of bytes of memory currently unused or free. This information tells you the size of the largest executable program you can run.

CHKDSK under DOS versions 3.3 and earlier does not recognize the IBM PS/2 Extended BIOS Data Area (which uses the highest 1K of addresses in contiguous conventional memory) and therefore reports only 639K, or 654,336 bytes, of total memory. For most IBM PS/2 systems with 640K of contiguous memory addressed before the video wall, the Extended BIOS Data Area occupies the 640th K. DOS 4.0 and later versions provide the correct 640K report.

During the FORMAT of the disk in the example, the FORMAT program did not find any unreadable sectors. Therefore, no clusters were marked in the FAT as bad or unusable, and CHKDSK did not display the *xxxxxxxxx* `bytes in bad sectors` message. Even if the disk had developed bad sectors since the FORMAT operation, CHKDSK still would not display any bytes in bad sectors because it does not test for and count bad sectors: CHKDSK reads the FAT and reports on whether the FAT says that there are any bad sectors. CHKDSK does not really count sectors; it counts clusters (allocation units) because that is how the FAT system operates.

Although bytes in bad sectors sounds like a problem or error message, it is not. The report is simply stating that a certain number of clusters are marked as bad in the FAT and that DOS therefore will never use those clusters. Because nearly all hard disks are

manufactured and sold with defective areas, this message is not uncommon. In fact, the higher quality hard disks on the market tend to have more bad sectors than the lower quality drives, based on the manufacturer defect list shipped with the drive (indicating all the known defective spots). Many of the newest controllers allow for sector and track sparing, in which the defects are mapped out of the DOS readable area so that DOS never has to handle them. This procedure is almost standard in drives that have embedded controllers, such as IDE (Integrated Drive Electronics) or SCSI (Small Computer Systems Interface) drives.

Suppose that you use a utility program to mark two clusters (150 and 151, for example) as bad in the FAT of the 720K floppy disk formatted earlier. CHKDSK then reports this information:

```
Volume FLOPPY_DISK created 01-16-1994 10:18p
Volume Serial Number is 266D-1DDC

      730,112 bytes total disk space
       79,872 bytes in 2 hidden files
       55,296 bytes in 1 user files
        2,048 bytes in bad sectors
      592,896 bytes available on disk

        1,024 bytes in each allocation unit
          713 total allocation units on disk
          579 available allocation units on disk

      655,360 total bytes memory
      632,736 bytes free
```

CHKDSK reports 2,048 bytes in bad sectors, which corresponds exactly to the two clusters just marked as bad. These clusters, of course, are perfectly good—you simply marked them as bad in the FAT. Using disk editor utility programs such as those supplied with the Norton or Mace Utilities, you can alter the FAT in almost any way you want.

CHKDSK Operation. Although bytes in bad sectors do not constitute an error or problem, CHKDSK reports problems on a disk volume with a variety of error messages. When CHKDSK discovers an error in the FAT or directory system, it reports the error with one of several descriptive messages that vary to fit the specific error. Sometimes the messages are cryptic or misleading. CHKDSK does not specify how an error should be handled; it does not tell you whether CHKDSK can repair the problem or whether you must use some other utility, or what the consequences of the error and the repair will be. Neither does CHKDSK tell you what caused the problem or how to avoid repeating the problem.

The primary function of CHKDSK is to compare the directory and FAT to determine whether they agree with one another—whether all the data in the directory for files (such as the starting cluster and size information) corresponds to what is in the FAT (such as chains of clusters with end-of-chain indicators). CHKDSK also checks subdirectory file entries, as well as the special . and .. entries that tie the subdirectory system together.

The second function of CHKDSK is to implement repairs to the disk structure. CHKDSK patches the disk so that the directory and FAT are in alignment and agreement. From a repair standpoint, understanding CHKDSK is relatively easy. CHKDSK almost always modifies the directories on a disk to correspond to what is found in the FAT. In only a couple of special cases does CHKDSK modify the FAT; when it does, the FAT modifications are always the same type of simple change.

Think of CHKDSK's repair capability as a directory patcher. Because CHKDSK cannot repair most types of FAT damage effectively, it simply modifies the disk directories to match whatever problems are found in the FAT.

CHKDSK is not a very smart repair program and often can do more damage repairing the disk than if it had left the disk alone. In many cases, the information in the directories is correct and can be used (by some other utility) to help repair the FAT tables. If you have run CHKDSK with the /F parameter, however, the original directory information no longer exists, and a good FAT repair is impossible. You therefore should never run CHKDSK with the /F parameter without first running it in read-only mode (without the /F parameter) to determine whether and to what extent damage exists.

Only after carefully examining the disk damage and determining how CHKDSK would fix the problems do you run CHKDSK with the /F parameter. If you do not specify the /F parameter when you run CHKDSK, the program is prevented from making corrections to the disk. Rather, it performs repairs in a mock fashion. This limitation is a safety feature because you do not want CHKDSK to take action until you have examined the problem. After deciding whether CHKDSK will make the correct assumptions about the damage, you might want to run it with the /F parameter.

Sometimes people place a CHKDSK /F command in their AUTOEXEC.BAT file—*a very dangerous practice*. If a system's disk directories and FAT system become damaged, attempting to load a program whose directory and FAT entries are damaged might lock the system. If, after you reboot, CHKDSK is fixing the problem because it is in the AUTOEXEC.BAT, it can irreparably damage the file structure of the disk. In many cases, CHKDSK ends up causing more damage than originally existed, and no easy way exists to undo the CHKDSK repair. Because CHKDSK is a simple utility that makes often faulty assumptions in repairing a disk, you must run it with great care when you specify the /F parameter.

Problems reported by CHKDSK are usually problems with the software and not the hardware. You rarely see a case in which lost clusters, allocation errors, or cross-linked files reported by CHKDSK were caused directly by a hardware fault, although it is certainly possible. The cause is usually a defective program or a program that was stopped before it could close files or purge buffers. A hardware fault certainly can stop a program before it can close files, but many people think that these error messages signify fault with the disk hardware—almost never the case.

I recommend running CHKDSK at least once a day on a hard disk system because it is important to find out about file structure errors as soon as possible. Accordingly, placing a CHKDSK command in your AUTOEXEC.BAT file is a good idea, but do not use the /F parameter. Also run CHKDSK whenever you suspect that directory or FAT damage could have occurred. For example, whenever a program terminates abnormally or a system crashes for some reason, run CHKDSK to see whether any file system damage has occurred.

Common Errors. All CHKDSK can do is compare the directory and FAT structures to see whether they support or comply with one another; as a result, CHKDSK can detect only certain kinds of problems. When CHKDSK discovers discrepancies between the directory and the FAT structures, they almost always fall into one of the following categories (these errors are the most common ones you will see with CHKDSK):

- Lost allocation units
- Allocation errors
- Cross-linked files
- Invalid allocation units

The RECOVER Command. The DOS RECOVER command is designed to mark clusters as bad in the FAT when the clusters cannot be read properly. When a file cannot be read because of a problem with a sector on the disk going bad, the RECOVER command can mark the FAT so that those clusters are not used by another file. Used improperly, this program is highly dangerous.

Many users think that RECOVER is used to recover a file or the data within the file in question. What really happens is that only the portion of the file before the defect is recovered and remains after the RECOVER command operates on it. RECOVER marks the defective portion as bad in the FAT and returns to available status all the data after the defect. Always make a copy of the file to be recovered before using RECOVER because the copy command can get all the information, including the portion of the file after the location of the defect.

Suppose that you are using a word processing program. You start the program and tell it to load a file called DOCUMENT.TXT. The hard disk has developed a defect in a sector used by this file, and in the middle of loading it, you see this message appear on-screen:

```
Sector not found error reading drive C
Abort, Retry, Ignore, Fail?
```

You might be able to read the file on a retry, so try several times. If you can load the file by retrying, save the loaded version as a file with a different name, to preserve the data in the file. You still have to repair the structure of the disk to prevent the space from being used again.

After ten retries or so, if you still cannot read the file, the data will be more difficult to recover. This operation has two phases, as follows:

- Preserve as much of the data in the file as possible.
- Mark the FAT so that the bad sectors or clusters of the disk are not used again.

Preserving Data. To recover the data from a file, use the DOS COPY command to make a copy of the file with a different name; for example, if the file you are recovering has the name DOCUMENT.TXT and you want the copy to be named DOCUMENT.NEW, enter the following at the DOS prompt:

```
COPY document.txt document.new
```

In the middle of the copy, you see the `Sector not found` error message again. The key to this operation is to answer with the (I)gnore option. Then the bad sectors are ignored, and the copy operation can continue to the end of the file. This procedure produces a copy of the file with all the file intact, up to the error location and after the error location. The bad sectors appear as gibberish or garbage in the new copied file, but the entire copy is readable. Use your word processor to load the new copy and remove or retype the garbled sectors. If this file were a binary file (such as a part of a program), you probably would have to consider the whole thing a total loss because you generally do not have the option of retyping the bytes that make up a program file. Your only hope then is to replace the file from a backup. This step completes phase one, which recovers as much of the data as possible. Now you go to phase two, in which you mark the disk so that these areas will not be used again.

Marking Bad Sectors. You mark bad sectors on a disk by using the RECOVER command. After making the attempted recovery of the data, you can use the following RECOVER command at the DOS prompt to mark the sectors as bad in the FAT:

```
RECOVER document.txt
```

In this case, the output of the RECOVER command looks like this:

```
Press any key to begin recovery of the file(s) on drive C:
XXXXX of YYYYY bytes recovered
```

The DOCUMENT.TXT file still is on the disk after this operation, but it has been truncated at the location of the error. Any sectors the RECOVER command cannot read are marked as bad sectors in the FAT and will show up the next time you run CHKDSK. You might want to run CHKDSK before and after running RECOVER, to see the effect of the additional bad sectors.

After using RECOVER, delete the DOCUMENT.TXT file because you have already created a copy of it that contains as much good data as possible.

This step completes phase two—and the entire operation. You now have a new file that contains as much of the original file as possible, and the disk FAT is marked so that the defective location will not be a bother.

Caution

Be very careful when you use RECOVER. Used improperly, it can do much damage to your files and the FAT. If you enter the RECOVER command without a file name for it to work on, the program assumes that you want every file on the disk recovered, and operates on every file and subdirectory on the disk; it converts all subdirectories to files, places all file names in the root directory, and gives them new names (FILE0000.REC, FILE0001.REC, and so on). This process essentially wipes out the file system on the entire disk. *Do not use RECOVER without providing a file name for it to work on*. This program should be considered as dangerous as the FORMAT command.

When you get the `Sector not found error reading drive C:`, rather than using the DOS RECOVER command, use the Norton Disk Doctor, or a similar utility, to repair the problem. If the error is on a floppy disk, use Norton's DiskTool before you use Disk Doctor. DiskTool is designed to help you recover data from a defective floppy disk. Disk Doctor and DiskTool preserve as much of the data in the file as possible, and afterward mark the FAT so that the bad sectors or clusters of the disk are not used again. These Norton Utilities also save UNDO information, making it possible for you to reverse any data recovery operation.

The DEBUG Program

The DOS DEBUG program is a powerful debugging tool for programmers who develop programs in assembly language. The following list shows some of the things you can do with DEBUG:

- Display data from any memory location.
- Display or alter the contents of the CPU registers.
- Display the assembly source code of programs.
- Enter data directly into any memory location.
- Input from a port.
- Move blocks of data between memory locations.
- Output to a port.
- Perform hexadecimal addition and subtraction.
- Read disk sectors into memory.
- Trace the execution of a program.
- Write disk sectors from memory.
- Write short assembly language programs.

To use the DEBUG program, make sure that DEBUG.COM is in the current directory or in the current DOS PATH. The following is the DEBUG command syntax:

DEBUG [*d:*][*path*][*filename*][*arglist*]

Entering DEBUG alone at the DOS prompt launches DEBUG. The *d:* and *path* options represent the drive and directory where the file you want to debug is located. The *filename* entry represents the file you want to debug. When you want to use DEBUG to work on a file, the file name is mandatory. The *arglist* entry represents parameters and switches that are passed to a program being debugged and can be used only if the file name is present.

After DEBUG is executed, its prompt is displayed (the DEBUG prompt is a hyphen). At the DEBUG hyphen prompt, you can enter a DEBUG command.

Because more powerful programs are available for debugging and assembling code, the most common use for DEBUG is patching assembly language programs to correct problems, changing an existing program feature, or patching disk sectors.

DEBUG Commands and Parameters. The documentation for DEBUG no longer is provided in the standard DOS manual. If you are serious about using DEBUG, you should purchase the *DOS Technical Reference Manual*, which contains the information you need to use this program. Many third-party books also provide documentation of the DEBUG commands and parameters.

As a quick reference to the DEBUG program, the following is a brief description of each command:

A address assembles macro assembler statements directly into memory.

C range address compares the contents of two blocks of memory.

D address or *D range* displays the contents of a portion of memory.

E address displays bytes sequentially and enables them to be modified.

E address list replaces the contents of one or more bytes, starting at the specified address, with values contained in the list.

F range list fills the memory locations in the range with the values specified.

G processes the program you are debugging without breakpoints.

G =address processes instructions beginning at the address specified.

G =address address processes instructions beginning at the address specified. This command stops the processing of the program when the instruction at the specified address is reached (breakpoint), and displays the registers, flags, and the next instruction to be processed. As many as ten breakpoints can be listed.

H value value adds the two hexadecimal values and then subtracts the second from the first. It displays the sum and the difference on one line.

I portaddress inputs and displays (in hexadecimal) one byte from the specified port.

L address loads a file.

L address drive sector sector loads data from the disk specified by drive and places the data in memory beginning at the specified address.

M range address moves the contents of the memory locations specified by range to the locations beginning at the address specified.

N filename defines file specifications or other parameters required by the program being debugged.

O portaddress byte sends the byte to the specified output port.

P =address value causes the processing of a subroutine call, a loop instruction, an interrupt, or a repeat-string instruction to stop at the next instruction.

Q ends the DEBUG program.

R displays the contents of all registers and flags and the next instruction to be processed.

R F displays all flags.

R registername displays the contents of a register.

S range list searches the range for the characters in the list.

T =address value processes one or more instructions starting with the instructions at CS:IP, or at =address if it is specified. This command also displays the contents of all registers and flags after each instruction is processed.

U address unassembles instructions (translates the contents of memory into assembler-like statements) and displays their addresses and hexadecimal values, together with assembler-like statements.

W address enables you to use the WRITE command without specifying parameters or by specifying only the address parameter.

W address drive sector sector writes data to disk beginning at a specified address.

XA count allocates a specified number of expanded memory pages to a handle.

XD handle deallocates a handle.

XM lpage ppage handle maps an EMS logical page to an EMS physical page from an EMS handle.

XS displays the status of expanded memory.

Changing Disks and Files. DEBUG can be used to modify sectors on a disk. Suppose that you use the following DEBUG command:

```
-L 100 1 0 1
```

This command loads into the current segment at an offset of 100h, sectors from drive B:\ (1), starting with sector 0 (the DOS volume boot sector), for a total of 1 or more sectors. You then could write this sector to a file on drive C:\ by using these commands:

```
-N C:\B-BOOT.SEC
-R CX
CX 0000
:200
-W
Writing 00200 bytes
-Q
```

The Name command sets up the name of the file to read or write.

The Register command enables you to inspect and change the contents of registers. The CX register contains the low-order bytes indicating the size of the file to load or save, and the BX register contains the high-order bytes. You would not need to set the BX register to anything but 0 unless the file was to be more than 65535 (64K) bytes in size. Setting the CX register to 200 indicates a file size of 200h, or 512 bytes.

The Write command saves 512 bytes of memory, starting at the default address of offset 100, to the file indicated in the Name command.

After quitting the program, your C:\ drive will have a file called B-BOOT.SEC that contains an image of the DOS volume boot sector on drive B:\.

Memory-Resident Software Conflicts

One area that gives many users trouble is a type of memory-resident software called Terminate and Stay Resident (TSR) or *pop-up utilities*. This software loads itself into memory and stays there, waiting for an activation key (usually a keystroke combination).

The problem with pop-up utilities is that they often conflict with each other, as well as with application programs and even DOS. Pop-up utilities can cause many types of problems. Sometimes the problems appear consistently, and at other times they are intermittent. Some computer users do not like to use pop-up utilities unless absolutely necessary because of their potential for problems.

Other memory-resident programs, such as MOUSE.COM, are usually loaded in AUTOEXEC.BAT. These memory-resident programs usually do not cause the kind of conflicts that pop-up utilities do, mainly because pop-up utilities are constantly monitoring the keyboard for the hotkey that activates them (and pop-up utilities are known to barge into memory addresses being used by other programs in order to monitor the keyboard, or to activate). Memory-resident programs like MOUSE.COM are merely installed in memory, do not poll the keyboard for a hotkey, and generally do not clash with the memory addresses used by other programs.

Device drivers loaded in CONFIG.SYS are another form of memory-resident software and can cause many problems.

If you are experiencing problems that you have traced to any of the three types of memory-resident programs, a common way to correct the problem is to eliminate the conflicting program. Another possibility is to change the order in which device drivers and memory-resident programs are loaded into system memory. Some programs must be loaded first, and others must be loaded last. Sometimes this order preference is indicated in the documentation for the programs, but often it is discovered through trial and error.

Unfortunately, conflicts between memory-resident programs are likely to be around as long as DOS is used. The light at the end of the tunnel is operating systems like Windows NT 3.1 and OS/2. The problem with DOS is that it establishes no real rules for how resident programs must interact with each other and the rest of the system. Windows NT and OS/2 are built on the concept of many programs being resident in memory at one time, and all multitasking. These operating systems should put an end to the problem of resident programs conflicting with each other.

Hardware Problems versus Software Problems

One of the most aggravating situations in computer repair is opening up a system and troubleshooting all the hardware just to find that the cause of the problem is a software program, not the hardware. Many people have spent large sums of money on replacement hardware such as motherboards, disk drives, adapter boards, cables, and so on, all on the premise that the hardware was causing problems, when software was actually the culprit. To eliminate these aggravating, sometimes embarrassing, and often expensive situations, you must be able to distinguish a hardware problem from a software problem.

Fortunately, making this distinction can be relatively simple. Software problems often are caused by device drivers and memory-resident programs loaded in CONFIG.SYS and AUTOEXEC.BAT on many systems. One of the first things to do when you begin having problems with your system is to boot the system from a DOS disk that has no CONFIG.SYS or AUTOEXEC.BAT configuration files on it. Then test for the problem. If it has disappeared, the cause was probably something in one or both of those files. To find the problem, begin restoring device drivers and memory-resident programs to CONFIG.SYS and AUTOEXEC.BAT one at a time (starting with CONFIG.SYS). For example, add one program back to CONFIG.SYS, reboot your system, and then determine if the problem has reappeared. When you discover the device driver or memory-resident program causing the problem, you might be able to solve the problem by editing CONFIG.SYS and AUTOEXEC.BAT to change the order in which device drivers and memory-resident programs are loaded, or you might have to eliminate the problem device driver or memory-resident program.

DOS can cause other problems, such as bugs or incompatibilities with certain hardware items. For example, DOS 3.2 does not support the 1.44M floppy drive format; therefore, using DOS 3.2 on a system equipped with a 1.44M floppy drive might lead you to believe (incorrectly) that the drive is bad. Make sure that you are using the correct version of DOS and that support is provided for your hardware. Find out whether your version of DOS has any official patches available; sometimes a problem you are experiencing might be one that many others have had, and IBM or Microsoft might have released a fix disk that takes care of the problem. For example, many PS/2 users have a floppy formatting

problem under DOS 3.3. They get a `track 0 bad` message after answering Yes to the `Format another diskette` message. This problem is solved by a special driver file on the DOS 3.3 patch disk.

If you are having a problem related to a piece of application software, a word processor or spreadsheet, for example, contact the company that produces the software and explain the problem. If the software has a bug, the company might have a patched or fixed version available, or it might be able to help you operate the software in a different way to solve the problem.

Summary

This chapter examined the software side of your system. Often a system problem is in the software and is not hardware-related. The chapter also examined DOS and showed how DOS organizes information on a disk. You learned about the CHKDSK, RECOVER, and DEBUG commands to see how DOS can help you with data and disk recovery. Finally, the chapter described memory-resident software and some of the problems it can cause, and informed you how to distinguish a software problem from a hardware problem.