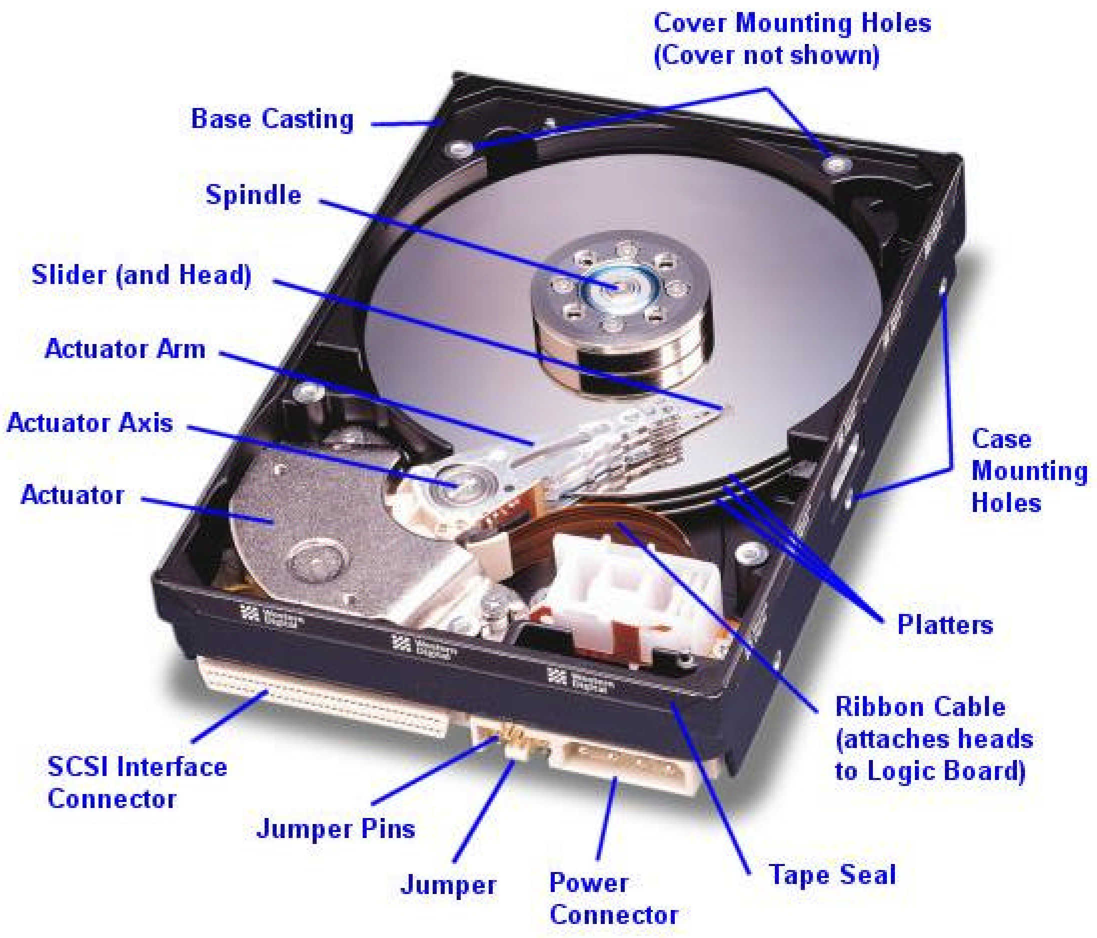
1 Construction and Operation of the Hard Disk Drive:

To many people, a hard disk is a "black box" of sorts--it is thought of as just a small device that "somehow" stores data. There is nothing wrong with this approach of course, as long as all you care about is that it stores data. If you use your hard disk as more than just a place to "keep stuff", then you want to know more about your hard disk. It is hard to really understand the factors that affect performance, reliability and interfacing without knowing how the drive works internally. Fortunately, most hard disks are basically the same on the inside. While the technology evolves, many of the basics are unchanged from the first PC hard disks in the early 1980s.



Photograph of a modern SCSI hard disk, with major components annotated.
The logic board is underneath the unit and not visible from this angle.

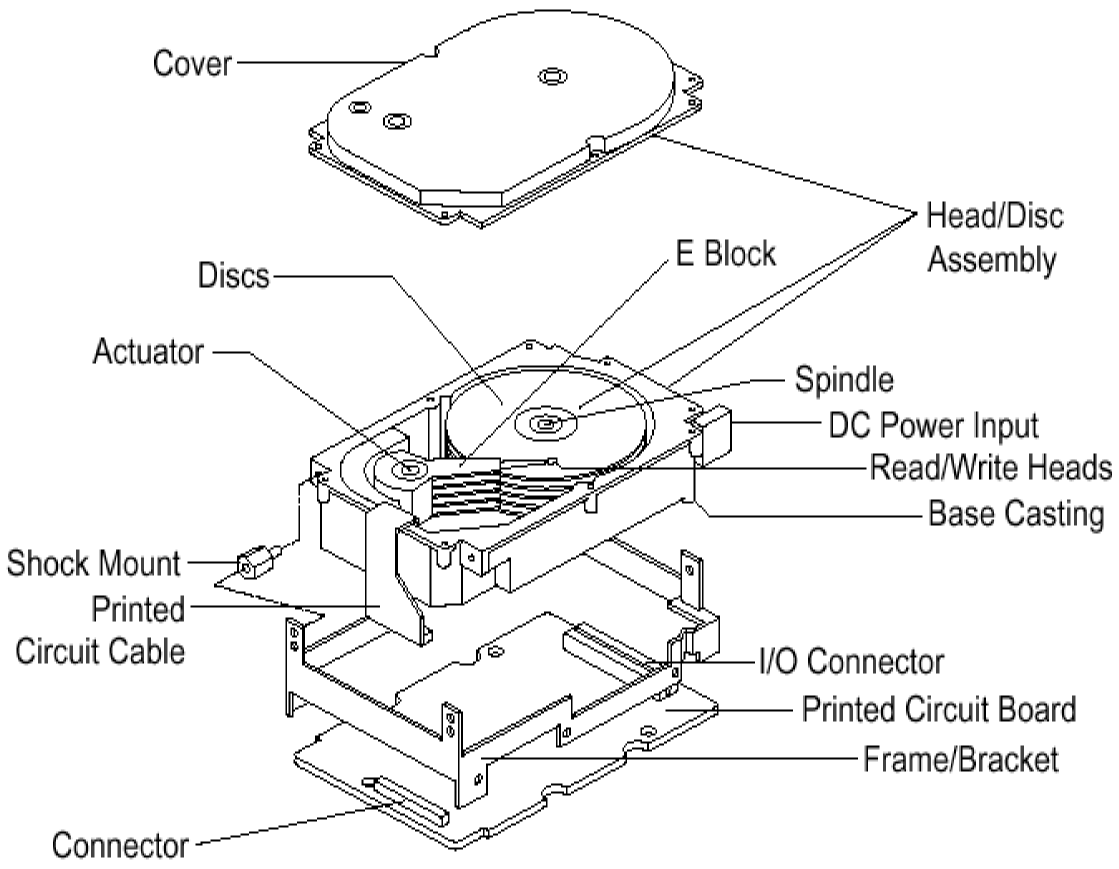
In this section we dive into the guts of the hard disk and discover what makes it tick. We look at the various key components, discuss how the hard disk is put together, and explore the various important technologies and how they work together to let you read and write data to the hard disk. My goal is to go beyond the basics, and help you really understand the design decisions and tradeoffs made by hard disk engineers, and the ways that new technologies are being employed to increase capacity and improve performance.

As an illustration, I'll describe here in words how the various components in the disk interoperate when they receive a request for data. Hopefully this will provide some context for the descriptions of the components that follow in later sections.

A hard disk uses round, flat disks called *platters*, coated on both sides with a special *media* material designed to store information in the form of magnetic patterns. The platters are mounted by cutting a hole in the center and stacking them onto a *spindle*. The platters rotate at high speed, driven by a special *spindle motor* connected to the spindle. Special electromagnetic read/write devices called *heads* are mounted onto *sliders* and used to either record information onto the disk or read information from it. The sliders are mounted onto *arms*, all of which are mechanically connected into a single assembly and positioned over the surface of the disk by a device called an *actuator*. A *logic board* controls the activity of the other components and communicates with the rest of the PC.

Each *surface* of each platter on the disk can hold tens of billions of individual bits of data. These are organized into larger "chunks" for convenience, and to allow for easier and faster access to information. Each platter has two heads, one on the top of the platter and one on the bottom, so a hard disk with three platters (normally) has six surfaces and six total heads. Each platter has its information recorded in concentric circles called *tracks*. Each track is further broken down into smaller pieces called *sectors*, each of which holds 512 bytes of information.

The entire hard disk must be manufactured to a high degree of precision due to the extreme miniaturization of the components, and the importance of the hard disk's role in the PC. The main part of the disk is isolated from outside air to ensure that no contaminants get onto the platters, which could cause damage to the read/write heads.



Exploded line drawing of a modern hard disk, showing the major components. Though the specifics vary greatly between different designs, the basic components you see above are typical of almost all PC hard disks.

Here's an example case showing in brief what happens in the disk each time a piece of information needs to be read from it. This is a highly simplified example because it ignores factors such as disk caching, error correction, and many of the other special techniques that systems use today to increase performance and reliability. For example, sectors are not read individually on most PCs; they are grouped together into continuous chunks called *clusters*. A typical job, such as loading a file into a spreadsheet program, can involve thousands or even millions of individual disk accesses, and loading a 20 MB file 512 bytes at a time would be rather inefficient:

1. The first step in accessing the disk is to figure out where on the disk to look for the needed information. Between them, the application, operating system, system BIOS and possibly any special driver software for the disk, do the job of determining what part of the disk to read.
2. The location on the disk undergoes one or more translation steps until a final request can be made to the drive with an address expressed in terms of its *geometry*. The geometry of the drive is normally expressed in terms of the cylinder, head and sector that the system wants the drive to read. (A cylinder is equivalent to a track for addressing purposes). A request is sent to the drive over the disk drive interface giving it this address and asking for the sector to be read.
3. The hard disk's control program first checks to see if the information requested is already in the hard disk's own internal buffer (or *cache*). It if is then the controller supplies the information immediately, without needing to look on the surface of the disk itself.
4. In most cases the disk drive is already spinning. If it isn't (because power management has instructed the disk to "spin down" to save energy) then the drive's controller board will activate the spindle motor to "spin up" the drive to operating speed.
5. The controller board interprets the address it received for the read, and performs any necessary additional translation steps that take into account the particular characteristics of the drive. The hard disk's logic program then looks at the final number of the cylinder requested. The cylinder number tells the disk which track to look at on the surface of the disk. The board instructs the actuator to move the read/write heads to the appropriate track.
6. When the heads are in the correct position, the controller activates the head specified in the correct read location. The head begins reading the track looking for the sector that was asked for. It waits for the disk to rotate the correct sector number under itself, and then reads the contents of the sector.
7. The controller board coordinates the flow of information from the hard disk into a temporary storage area (buffer). It then sends the information over the hard disk interface, usually to the system memory, satisfying the system's request for data.

2. Integrated Drive Electronics / AT Attachment (IDE/ATA) Interface

The most popular interface used in modern hard disks--by far--is the one most commonly known as *IDE*. This interface is also known by a truly staggering variety of other names such as *ATA*, *ATA/ATAPI, EIDE*, *ATA-2*, *Fast ATA*, *ATA-3, Ultra ATA*, *Ultra DMA* and many more as well. The invention of this interface catapulted hard disks into a new era of performance, reliability, and compatibility. IDE/ATA hard disks are used on the vast majority of modern PCs, and offer excellent performance at relatively low cost. They are challenged only by **SCSI**, which has certain **advantages and disadvantages** when the two interfaces are compared..

One problem with this interface is the ridiculous number of different names that are used to refer to it, and how misleading some of those names are. For starters, the most commonly used name for this interface, "IDE" is a misnomer itself. (**See this page** for a discussion of the history behind this term.) The "proper" name for the IDE interface is *AT Attachment*, or *ATA*. This name is not as commonly used, for historical reasons (people are stubborn). I use the generic term "IDE/ATA" to convey the dual naming conventions used for the various generations and variants of this interface.

In this section I examine the IDE/ATA interface in detail. I begin with a brief overview of the interface, discussing a bit of its history and how it works in general terms. I describe the different generations of the various standards that define the ATA interface. I then discuss the plethora of "unofficial standards" or marketing terms that are used to refer to IDE/ATA, in a hopefully fruitful attempt to clarify what all those strange acronyms are. The following two sections get into the nitty gritty of the interface, describing the various IDE/ATA transfer modes and protocols, and then providing relevant information on how to configure and connect IDE/ATA devices on your PC.

3. Ultra DMA (UDMA) Modes

With the increase in performance of hard disks over the last few years, the use of **programmed I/O modes** became a hindrance to performance. As a result, focus was placed on the use of direct memory access (DMA) modes. In particular, bus mastering DMA on the PCI bus became mainstream due to its efficiency advantages. If you have not yet, you should **read the description of the various DMA modes and how bus mastering DMA works**; this will help you understand this page much better.

Of course, hard disks get faster and faster, and the maximum speed of multiword DMA mode 2, 16.7 MB/s, quickly became insufficient for the fastest drives. However, the engineers who went to work to speed up the interface discovered that this was no simple task. The IDE/ATA interface, and the flat ribbon cable it used, were designed for slow data transfer--about 5 MB/s. Increasing the speed of the interface (by reducing the cycle time) caused all sorts of signaling problems related to interference. So instead of making the interface run faster, a different approach had to be taken: improving the efficiency of the interface itself. The result was the creation of a new type of DMA transfer modes, which were called *Ultra DMA modes*.

The key technological advance introduced to IDE/ATA in Ultra DMA was *double transition clocking*. Before Ultra DMA, one transfer of data occurred on each clock cycle, triggered by the rising edge of the interface clock (or "strobe"). With Ultra DMA, data is transferred on both the rising and falling edges of the clock. Double transition clocking, along with some other minor changes made to the signaling technique to improve efficiency, allowed the data throughput of the interface to be doubled for any given clock speed.

In order to improve the integrity of this now faster interface, Ultra DMA also introduced the use of *cyclical redundancy checking* or *CRC* on the interface. The device sending data uses the CRC algorithm to calculate redundant information from each block of data sent over the interface. This "CRC code" is sent along with the data. On the other end of the interface, the recipient of the data does the same CRC calculation and compares its result to the code the sender delivered. If there is a mismatch, this means data was corrupted somehow and the block of data is resent. (CRC is similar in concept and operation to the way error checking is done on the system memory.) If errors occur frequently, the system may determine that there are hardware issues and thus drop down to a slower Ultra DMA mode, or even disable Ultra DMA operation.

The first implementation of Ultra DMA was specified in the ATA/ATAPI-4 standard and included three Ultra DMA modes, providing up to 33 MB/s of throughput. Several newer, faster Ultra DMA modes were added in subsequent years. This table shows all of the current Ultra DMA modes, along with their cycle times and maximum transfer rates:

Ultra DMA Mode	Cycle Time (nanoseconds)	Maximum Transfer Rate (MB/s)	Defining Standard
Mode 0	240	16.7	ATA/ATAPI-4
Mode 1	160	25.0	ATA/ATAPI-4
Mode 2	120	33.3	ATA/ATAPI-4
Mode 3	90	44.4	ATA/ATAPI-5
Mode 4	60	66.7	ATA/ATAPI-5
Mode 5	40	100.0	ATA/ATAPI-6?

The cycle time shows the speed of the interface clock; the clock's frequency is the reciprocal of this number. The maximum transfer rate is four times the reciprocal of the cycle time--double transition clocking means each cycle has two transfers, and each transfer moves two bytes (16 bits). Only modes 2, 4 and 5 have ever been used in drives; I'm not sure why they even bothered with mode 0, perhaps for compatibility. Ultra DMA mode 5 is the latest, and is implemented in all currently-shipping drives. It is anticipated that it will be included in the forthcoming **ATA/ATAPI-6** standard.

Note: In common parlance, drives that use Ultra DMA are often called "Ultra ATA/xx" where "xx" is the speed of the interface. So, few people really talk about current drives being "Ultra DMA mode 5", they say they are "**Ultra ATA/100**".

Double transition clocking is what allows Ultra DMA mode 2 to have a maximum transfer rate of 33.3 MB/s despite having a clock cycle time identical to "**regular DMA**" multiword mode 2, which has half that maximum. Now, you may be asking yourself: if they had to go to double transition clocking to get to to 33.3 MB/s, how did they get to 66 MB/s, and then 100 MB/s? Well, they did in fact speed up the interface after all. But the use of double transition clocking let them do it while staying at half the speed they would have needed. Without double transition clocking, Ultra DMA mode 5 would have required a cycle time of 20 nanoseconds instead of 40, making implementation much more difficult.

Even with the advantage of double transition clocking, going above 33 MB/s finally exceeded the capabilities of the old **40-conductor standard IDE cable**. To use Ultra DMA modes over 2, a special, 80-conductor IDE cable is required. This cable uses the same 40 pins as the old cables, but adds 40 ground lines between the original 40 signals to separate those lines from each other and prevent interference and data corruption. **I discuss the 80-conductor in much more detail here**. (The 80-conductor cable was actually specified in ATA/ATAPI-4 along with the first Ultra DMA modes, but it was "optional" for modes 0, 1 and 2.)

Today, all modern systems that use IDE/ATA drives should be using one of the Ultra DMA modes. There are several specific requirements for running Ultra DMA:

1. **Hard Disk Support:** The hard disk itself must support Ultra DMA. In addition, the appropriate Ultra DMA mode must be enabled on the drive.
2. **Controller Support:** A controller capable of Ultra DMA transfers must be used. This can be either the interface controller built into the motherboard, or an add-in IDE/ATA interface card.
3. **Operating System Support:** The BIOS and/or operating system must support Ultra DMA transfers, and the hard disk must be set to operate in Ultra DMA in the operating system.

4. **80-Conductor Cable:** For Ultra DMA modes over 2, an 80-conductor cable must be used. If an 80-conductor cable is not detected by the system, 66 MB/s or 100 MB/s operation will be disabled. **See the discussion of the 80-conductor cable for more**.

On new systems there are few issues with running Ultra DMA, because the hardware is all new and designed to run in Ultra DMA mode. With older systems, things are a bit more complex. In theory, new drives *should* be backwards compatible with older controllers, and putting an Ultra DMA drive on an older PC should cause it to automatically run in a slower mode, such as PIO mode 4. Unfortunately, certain motherboards don't function well when an Ultra DMA drive is connected, and this may result in lockups or errors. A BIOS upgrade from the motherboard manufacturer is a good idea, if you are able to do this. Otherwise, you may need to use a special *Ultra DMA software utility* (available from the drive manufacturer) to tell the hard disk not to try to run in Ultra DMA mode. The same utility can be used to enable Ultra DMA mode on a drive that is set not to use it. You should use the utility specific to whatever make of drive you have.

4. Hard Disk Logical Structures and File Systems

The hard disk is of course a medium for storing information. Hard disks grow in size every year, and as they get larger, using them in an efficient way becomes more difficult. The *file system* is the general name given to the logical structures and software routines used to control the access to the storage on a hard disk system. Different operating systems use different ways of organizing and controlling access to data on the hard disk, and this choice is basically independent of the specific hardware being used. The information in this section in fact straddles the fine line between hardware and software, a line which gets more and more blurry every year.

The nature of the logical structures on the hard disk has an important influence on the performance, reliability, expandability and compatibility of your storage subsystem. This section takes a look at the logical structures on the hard disk and how they are set up and used for a typical PC installation. Of particular concern is how setup and operation of the file system has an impact on performance.

The focus in this section is primarily on "standard" file systems used on PCs, and in particular the FAT file system (and its variants) used by DOS (and its successors). This is done because these are by far the most commonly used, and also because my familiarity with them is higher. If you are using Linux (only) for example, much of what is in this section will not apply to you, although much of it still will.

5. **Different operating systems** use different file systems. Some are designed specifically to work with more than one, for compatibility reasons; others work only with their own file system. This section takes brief look at the most common operating systems in use on the PC and the file systems that they use. This enables you to know what parts of the rest of the discussion on file systems is most relevant to your situation.

- ? **DOS (MS-DOS, PC-DOS, etc.)**
- ? **Windows 3.x**
- ? **Windows 95**
- ? **Windows NT**
- ? **UNIX / Linux**

5.1 DOS (MS-DOS, PC-DOS, etc.)

Regular DOS--versions 6.22 and earlier--uses the FAT file system for file access. This includes the FAT12 and FAT16 variations. Windows 95 comes with a "built-in" version of DOS, version 7.x. The OEM Service Release 2 version of **Windows 95** supports FAT32, and therefore, DOS 7.1

that comes with Windows 95 OSR2 also supports FAT32. FAT32 partitions are not compatible with standard, pre-7.x DOS.

5.2 Windows 3.x

Windows 3.x is not a true, independent multi-tasking operating system. It runs on top of DOS and for the most part, uses DOS facilities and routines for disk access. Therefore, it uses the same FAT file system that DOS does.

The last version of Windows 3.x, Windows for Workgroups 3.11, includes an enhancement called "32-Bit File Access". This is really a poorly-named feature that refers to the use of 32-bit protected mode routines for accessing the disk, instead of using the standard 16-bit DOS routines. In fact, this is really the first implementation of the VFAT file system used by Windows 95, although not all of the VFAT features are included--only the use of 32-bit access routines. The only thing different here is how the disk is accessed; the file system structures are "plain" FAT.

5.3 Windows 95

Windows 95 is the great "compromise" operating system. In some respects, it has its own way of handling access to the hard disk, but in other ways it resembles, and even uses, standard DOS. This is how Windows 95 strives for performance while retaining compatibility with older software. Windows 95 in fact includes a version of DOS, DOS 7.x, that is designed to work with it and its file structures.

The "official" file system used by Windows 95 is VFAT, which is supported by both it and the DOS 7.x with which it ships. Starting with Windows 95 OEM Service Release 2 (OSR2), FAT32 is also supported, which allows the use of larger hard disk partitions.

5.4 Windows NT

Windows NT is a new implementation of Windows that was designed from the ground up. Unlike the other Windows variations, Windows NT is *not* based on DOS.

Windows NT supports two different file systems. The first is NTFS, the NT file system. This is an advanced UNIX-like file system. The second is regular FAT, like that used by DOS. Support for FAT is included for compatibility reasons or for setting up PCs that dual boot to both Windows NT and another operating system. Even though Windows NT will read both FAT and NTFS partitions, they are not compatible with each other.

All file systems perform the same basic functions, based on their fundamental goals: the intelligent organization of data and efficient control of and access to that data. As a result, most of them resemble each other to some extent, even if they are also different from each other in important ways. Even if two PC file systems differ greatly in terms of their internal architecture and structures, they can be made to resemble each other closely in their outward appearance. For example, Windows NT will support both FAT and NTFS partitions, which are totally different in terms of their internal structures, but have virtually the same interface for the user (and for software applications).

5.5 UNIX / Linux

UNIX has been around for many decades, making it the oldest of all file systems used on PC hardware. UNIX file systems are also probably the most different from the other file systems used on PC, both internally and externally (referring to how the user accesses the file system). While most Windows users are accustomed to the Explorer-type interface for managing files and folders, UNIX files are usually managed with discrete text commands--similar to how DOS works (in fact, many principles of the FAT file system are based on UNIX.). There are graphical UNIX shells as well, of course, but many UNIX users (myself included) never use them.

The more important differences, however, are internal. UNIX file systems are designed not for easy use, but for robustness, security and flexibility. UNIX file systems offer the following features, and have for many years:

? Excellent expandability, and support for large storage devices.
? Directory-level and file-level security and access controls, including the ability to control which users or groups of users can read, write or execute a file.
? Very good performance and efficient operation.
? The ability to create "flexible" file systems containing many different devices, to combine devices and present them as a single file system, or to remotely mount other storage devices for local use.
? Facilities for effectively dealing with many users and programs in a multitasking environment, while requiring a minimum of administration.
? Ways to create special constructs such as logically linked files.
? Reliability and robustness features such as journaling and support for RAID.
? If these features sound similar to those of NTFS, that's because UNIX and Windows NT/2000 now compete for much of the same market, so NTFS was given most of the capabilities that UNIX has. There are many other features as well, which differ from one implementation to another--there is no single "UNIX file system", any more than there is a single "UNIX operating system". Each UNIX variant (including the popular Linux for the PC, which itself has many different flavors) has a slightly different file system, though they are of course very similar to each other, and it is usually possible for different UNIX implementations to read each other's files.

UNIX file systems were one of the first (if not the first) to use the hierarchical directory structure, with a root directory and nested subdirectories. (Most of us are familiar with this from using it with the FAT file system, which works the same way). One of the key characteristics of UNIX file systems is that virtually everything is defined as being a file--regular text files are of course files, but so are executable programs, directories, and even hardware devices are mapped to file names. This provides tremendous flexibility to programmers and users, even if there is a bit of a learning curve at first.

Since few PC users run UNIX on their own machines, it's unlikely that you will ever actually use a UNIX file system directly. However, you may find yourself using a UNIX file system if you run a web site, for example, so understanding the basics of how UNIX works is a good idea.

6 PC File Systems

There are several different file systems that are commonly used in PCs. This section briefly lists them and discusses how they differ. Note that the rest of the discussion on file systems elaborates significantly on most of the file systems listed here.

✍ **FAT**
✍ **FAT32**
✍ **Virtual FAT (VFAT)**
✍ **NT File System (NTFS)**

6.1 FAT

The primary file system used in the vast majority of PCs is *FAT*. This name actually stands for *file allocation table*, which is one of the main logical structures that the file system uses, so it's not the best name to refer to the whole file system. FAT is used by most DOS or Windows-based PCs. It is sometimes called *FAT16*, in reference to its 16-bit nature, to avoid confusion with its successor, **FAT32**.

6.2 FAT32

The structure that gives the FAT file system its name is the *file allocation table*. In order to understand what this important table does, you must first understand how space on the hard disk is allocated under DOS (and its derivatives that also use FAT).

While data is stored in 512-byte sectors on the hard disk, for performance reasons individual sectors are not normally allocated to files. The reason is that it would take a lot of overhead (time and space) to keep track of pieces of files that were this small. The hard disk is instead broken into larger pieces called *clusters*, or alternatively, *allocation units*. Each cluster contains a number of sectors. Typically, clusters range in size from 2,048 bytes to 32,768 bytes, which corresponds to 4 to 64 sectors each.

The file allocation table is where information about clusters is stored. Each cluster has an entry in the FAT that describes how it used. This is what tells the operating system which parts of the disk are currently used by files, and which are free for use. The FAT entries are used by the operating system to **chain together clusters to form files**

The file allocation tables are stored in the area of the disk immediately following the volume boot sector. Each volume actually contains two identical copies of the FAT; ostensibly, the second one is meant to be a backup of sorts in case of any damage to the first copy. Damage to the FAT can of course result in data loss since this is where the record is kept of which parts of the disk contain which files. The problem with this built-in backup is that the two copies are kept right next to each other on the disk, so that in the event that for example, bad sectors develop on the disk where the first copy of the FAT is stored, chances are pretty good that the second copy will be affected as well.

FAT32 is an enhancement to the standard FAT file system. It is named FAT32 because it allows the use of 32-bit numbers to represent cluster numbers, instead of the 16-bit numbers used by standard FAT (which is also called FAT16 for that reason). FAT32 was introduced in Windows 95's OEM Service Release 2, also sometimes called Windows 95b and is supported by that version of Windows 95, and the version of DOS that comes with it. Earlier operating systems cannot read a disk volume formatted with FAT32.

FAT32 was created primarily for one reason: hard disk manufacturers began making mainstream hard disks larger than 2 GB in size, and FAT16 supports only a maximum of 2 GB per logical disk volume. FAT32 extends this up to 8 GB in its current implementation and can handle even larger disks using the same basic structures.

In addition to allowing much larger individual disk volumes, FAT32 saves wasted space due to **slack**, because it uses much smaller cluster sizes than FAT16 does. The tradeoff for this is that the number of clusters used is much larger, which can mean a small performance hit due to the extra amount of overhead required (plus extra memory to hold the larger FAT).

Aside from the difference in the way clusters are assigned and numbered, FAT32 is at its essence the same as regular FAT and the descriptions of FAT file structures apply to FAT32 as well. The matter of using regular FAT16 vs. FAT32, and choosing partition and cluster sizes

6.3 Virtual FAT (VFAT)

Microsoft incorporated several enhancements into the disk management capabilities of Windows 95. Access to the file system can be done using high-speed, protected-mode, 32-bit drivers, or for compatibility, the older DOS 16-bit routines. Support was added for **long file names** and also for better control over such matters as disk locking, so utilities could access the disk in "exclusive mode" without fear of other programs using it in the meantime.

Despite the new name and new capabilities, VFAT as a file system is basically the same as FAT is. Most of the new capabilities relate to *how* the file system is used, and not the actual structures on the disk. VFAT handles standard FAT16 partitions, and under Windows 95 OSR2 or later, FAT32 partitions as well. The only significant change in terms of actual structures is the addition of long file names. Even here, VFAT supports these using what is basically a hack, as opposed to anything really revolutionary.

With the exception of the long file names, Windows 95, using VFAT, shares the same logical disk structures as DOS or Windows 3.x using FAT.

The *NTFS* file system used by Windows NT is completely different from, and incompatible with, the FAT file system that is used by DOS and the other Windows varieties. NTFS can only be used by Windows NT--other operating systems do not have the ability to use a disk formatted with NTFS.

6.4 NTFS

NTFS (New Technology File System) is in virtually every way, far superior to FAT. It is a robust, full-featured system that includes file-by-file compression, full permissions control and attribute settings, transaction-based operation, and many more features. It also does not have the problems with cluster sizes and hard disk size limitations that FAT does, and has other performance-enhancing features such as **RAID** support. The only way that NTFS is not superior to FAT is in compatibility with older software. NTFS is not nearly as widely-used as FAT, for this reason. For now I am not including a full examination of NTFS on the site, but I may add this at a later time if it seems warranted.

7 Major Disk Structures and the Boot Process

There are several major disk structures that are used to organize and control the storage of information on the PC using the FAT / FAT32 / VFAT file system. These structures are important to understand, since they control the way the hard disk works from a software perspective.

- **Master Boot Record (MBR)**
- **Primary, Extended and Logical Partitions**
- **Volume Boot Sectors**
- **Active Partitions and Boot Managers**
- **The DOS Boot Process**
- **Master Boot Record (MBR)**
- **Boot Sector Viruses**

7.1 Master Boot Record (MBR)

When you turn on your PC, the processor has to begin processing. However, your system memory is empty, and the processor doesn't have anything to execute, or really even know where it is. To ensure that the PC can always boot regardless of which BIOS is in the machine, chip makers and BIOS manufacturers arrange so that the processor, once turned on, always starts executing at the same place, FFFF0h. This is discussed in much more detail here.

In a similar manner, every hard disk must have a consistent "starting point" where key information is stored about the disk, such as how many partitions it has, what sort of partitions they are, etc. There also needs to be somewhere that the BIOS can load the initial boot program that starts the process of loading the operating system. The place where this information is stored is called the *master boot record* (*MBR*). It is also sometimes called the *master boot sector* or even just the *boot sector*.

The master boot record is always located at cylinder 0, head 0, and sector 1, the first sector on the disk. This is the consistent "starting point" that the disk always uses. When the BIOS boots the machine, it will look here for instructions and information on how to boot the disk and load the operating system. The master boot record contains the following structures:

- **Master Partition Table:** This small table contains the descriptions of the partitions that are contained on the hard disk. There is only room in the master partition table for the information describing four partitions. Therefore, a hard disk can have only four true partitions, also called *primary partitions*. Any additional partitions are logical partitions that are linked to one of the primary partitions. **Partitions are discussed here**.
- **Master Boot Code:** The master boot record contains the small initial boot program that the BIOS loads and executes to start the boot process. This program eventually transfers control to the boot program stored on whichever partition is used for booting the PC.

Due to the great importance of the information stored in the master boot record, if it ever becomes damaged or corrupted in some way, serious data loss can be--in fact, often will be--the result. Since the master boot code is the first program executed when you turn on your PC, **this is a favorite place for virus writers to target**.

7.2 Primary, Extended and Logical Partitions

In order to use the space in a hard disk, it must be *partitioned*. Partitioning is the process of dividing the hard disk's space into pieces, so they can be prepared for use, or even dedicated to different uses. Even if the entire disk is intended to be left in one piece, it must be partitioned so that the operating system knows that it is intended to be left in one piece. **There are many different considerations that go into deciding how to partition a hard disk**.

Each hard disk can contain up to four different "true" partitions, which are called *primary partitions*. The limitation of four is one that is imposed on the system by the way that the master boot record is structured. Normally, if you are only using DOS, Windows 3.x or Windows 95, you will have a single primary partition. Multiple primary partitions can be used if you want to set your machine up for use by multiple operating systems. Partitions are also sometimes called *volumes*, especially in reference to DOS.

DOS itself can only have one primary partition per hard disk. You may find this statement confusing, since you have no doubt seen PCs that have four or more DOS partitions. A hard disk that has four DOS partitions still has only (at most) one primary DOS partition. The others are *logical partitions* that are stored in the disk's *extended DOS partition*.

OK, this sounds confusing so I will try to clarify somewhat. The original designers of DOS obviously did not foresee multi-gigabyte hard disks containing large numbers of partitions. Later versions of DOS were enhanced to allow DOS to use up to 24 total disk partitions. In order to preserve compatibility with the original DOS structures that only allow four partitions, the extra partitions are stored in an extended partition.

I mentioned above that there is room for four primary partitions on a hard disk. If you use an extended partition, however, that takes up a "slot" reserved for one of the four primaries, so you are then limited to three primaries plus the extended partition. When you set up an extended partition, it is initially empty; you use up the space in it by adding *logical partitions* (sometimes also called *logical DOS drives* or *logical volumes*). You can store up to 24 logical drives in the extended partition if you are not using a DOS primary partition on the disk, or 23 if you are using a primary. The limiting factor here is drive letters: hard disks start with C: and end with Z:. Of course virtually nobody uses that many partitions on their system.

Internally, the logical drives are stored in a linked structure. The extended partition's information is contained in the master partition table (since the extended partition is one of the four partitions stored in the master boot record). It contains a link to an *extended partition table* that describes the first logical partition for the disk. That table contains information about that first logical partition, and a link to the *next* extended partition table which describes the second logical partition on the disk, and so on. The extended partition tables are lnked in a chain starting from the master partition table.

In terms of how the disk is used, there are only two main differences between a primary and a logical partition or volume. The first is that a primary partition can be set as **bootable (active)** while a logical cannot. The second is that **DOS assigns drive letters** (C:, D: etc.) differently to primary and logical volumes.

Here's an example to hopefully make all of this a bit clearer. Let's suppose you are setting up a new system and starting with an empty 2 GB hard disk. For efficiency purposes you have decided to partition the system into four equal 500 MB partitions. I'm assuming no complicating factors here, just a simple example.

To do this, you will first set up a primary DOS partition 500 MB in size. This is the first of your four partitions. You will then create an extended DOS partition that is 1500 MB in size. Within the extended DOS partition you will create three logical volumes, 500 MB in each, which are your second, third and fourth volumes. The first partition will be your C: drive from which you boot the machine, and DOS will (normally) assign D:, E: and F: to the other three partitions. Your hard disk will have one primary DOS partition, and one extended DOS partition containing three logical DOS volumes.

Once the system is set up, there is no functional difference between these partitions, other than the fact that C: is the only bootable one and is where all the DOS system files reside. Regular files can reside wherever you want them to.

7.3 Volume Boot Sectors

Each DOS partition (also called a DOS volume) has its own *volume boot sector*. This is distinct from the *master* boot sector (or record) that controls the entire disk, but is similar in concept. Each volume boot sector contains the following:

? **Disk Parameter Block:** Also sometimes called the *media parameter block*, this is a data table that contains specific information about the volume, such as its specifications (size, number of sectors it contains, etc.), label name, etc.

? **Volume Boot Code:** This is code that is specific to the operating system that is using this volume and is used to start the load of the operating system. This code is called by the master boot code that is stored in the master boot record, but only for the primary partition that is set as active. For other partitions, this code sits unused.

The volume boot sector is created when you do a **high-level format of a hard disk partition**. The boot sector's code is executed directly when the disk is booted, making it a **favorite target for virus writers**.

7.4 Active Partitions and Boot Managers

Only primary partitions can be used to boot the operating system, and of these, only the primary partition that is set to be bootable. Only one can be set bootable at a time because otherwise, the master boot record does not know to which volume's boot code to give control of the boot process when the machine is turned on. DOS calls the bootable partition the *active* partition.

If you partition a new hard disk and create a primary DOS partition using the standard DOS utility **FDISK**, but forget to set the primary partition active, the BIOS will be unable to boot the operating system. This usually results in an error message like "No boot device available". Some BIOSes will give much more cryptic messages; AMI BIOSes are famous for giving the bizarre "NO ROM BASIC - SYSTEM HALTED" message when it cannot find a boot device. The reason for this error is that older IBM systems had a hard-coded version of the BASIC language built into its BIOS ROM. If no boot device could be found, the BIOS would execute this hard-coded BASIC interpreter instead. Since non-IBM systems don't have this BASIC ROM, their BIOSes must display an error message instead of going into BASIC. Why AMI chose this confusing message is a mystery to me.

Most people are only going to have one primary partition on their PC, because most people only use one operating system. If you are using more than one operating system however--I mean incompatible ones that use different file formats, like Windows 95 with UNIX, not DOS and Windows 95, which use the same file systems generally--then you may want to set up multiple primary partitions, one per operating system. You then have the problem of telling the system at boot time which operating system you want to use.

There are programs specifically designed for this task; they are usually called *boot managers*. What a boot manager does is insert itself into the very beginning of the boot process, normally by setting up a special boot manager partition and making itself the active partition. When you boot up the PC, the code in this partition runs. It analyzes the primary partitions on the disk and then presents a menu to you and asks which operating system you want to use. Whichever one you select, it marks as active, and then continues the boot process from there.

Boot managers are in many ways indispensable when working with multiple operating systems. However, you still want to take care when using one, since it does modify the disk at a very low level. Some boot managers require their own, dedicated partitions to hold their own code, which complicates slightly the setup of the disk. One common boot manager is IBM's boot manager, which ships as one component of ***PowerQuest's PartitionMagic 3***.

7.5 The DOS Boot Process

The system boot sequence is the series of steps that the system performs when it is turned on (or rebooted with the reset switch, for example). This always starts with the special boot program software that is in the system BIOS ROM. The BIOS has several steps that it must perform to test the system and set it up, before any operating system can be loaded. These steps are described in detail here.

Once the BIOS have completed its startup activities, the last thing it does is to begin the process of loading the operating system. It does this by searching for a boot device containing boot code to which it can hand off the boot process. It will search for boot devices in the order specified by the BIOS setting that controls the boot sequence. If it cannot find a boot device it will terminate with an error.

Assuming that the BIOS find a boot sector on a device, the process of loading DOS begins. The process below outlines booting from the hard disk. Booting from the floppy disk differs only in the first few steps, because the floppy disk's structures are slightly different. Floppies cannot be partitioned, and hence have no master boot record or partitions. This means that the steps where the master boot record is searched are skipped.

Here are the steps in the DOS boot process:

10 The BIOS, having completed its functions, loads the boot code in the master boot record and transfers control to it. The master boot record code begins execution. If the boot device is a floppy disk, the process continues with step 6.

10 The master boot code examines the master partition table. It is searching for two things. First, it must determine if there is an extended DOS partition. Second, it must determine if there is a bootable partition specified in the partition table.

10 If the master boot code finds an **extended partition** on the disk, it loads the extended partition table that describes the first logical volume in the extended partition. This extended partition table is examined to see if it points to another extended partition table. If it does, then that table contains information about the *second* logical volume in the extended partition, so it is loaded and examined. (Recall that logical volumes in the extended partition have their extended partition table chained one to the next.) This process is continued until all of the extended partitions have been loaded and recognized by the system.

10 After loading the extended partition information (if any), the code attempts to boot the primary partition that is marked active (bootable). If there are no partitions marked active, then the boot process will terminate with an error. The error message is often the same one that occurs if the BIOS finds no boot device, and is generally something like "No boot device", but can be the infamous "NO ROM BASIC - SYSTEM HALTED".

10 If there is a primary partition marked active, the code will boot it. The rest of the steps assume this is a DOS primary partition.

10 The volume boot sector is loaded into memory and tested, and the boot code that it contains is given control of the remainder of the boot process.

10 The volume boot code examines the structures on the disk that it is booting to ensure that everything is correct and in the right place. If not, the boot process will end in an error here as well.

10 The code searches the root directory of the device being booted for the operating system files that contain the operating system. For a system running MS-DOS these are the files "IO.SYS", "MSDOS.SYS" and "COMMAND.COM".

10 If the operating system files are not found, the boot program will display an error message, which is usually something like "Non-system disk or disk error - Replace and press any key when ready". Some people think that this message means the system was never booted, that the BIOS examined the floppy disk for example and just rejected it because it couldn't boot it. As you can see from this description of the boot process, the volume boot code was indeed loaded and executed, and in fact it is what prints the message when it can't find the operating system files! **See here for an explanation of why this distinction is so important**.

10 If the operating system files are found, the boot program will load them into memory and transfer control to them. These larger files contain the more complete operating system code that loads and initializes the rest of the operating system structures. For MS-DOS, this means loading the command interpreter and then reading and interpreting the contents of the CONFIG.SYS and AUTOEXEC.BAT system control files.

At this point the operating system code itself has control of the PC. In the case of an operating system like Windows 95 the initial operating system files control the loading and execution of many more routines as the boot progresses. In fact, it is surprising in some ways just how many different pieces of code have a hand in starting up the PC.

7.6 Boot Sector Viruses

Computer viruses are small programs designed to attach themselves to your computer, running without your knowledge and spreading them to "infect" other systems. Sometimes malicious, and sometimes just annoying, they are always a concern to the modern computer user.

The boot code that is stored on the hard disk and executed when the disk is booted up, is a prime target for viruses. The reason is simple: the goal of the virus writer is to get the virus code executed as often as possible, to allow it to spread and cause whatever other mischief it is written to create. What better place to put the virus than in code that is executed every time the PC is run, automatically, and before anything else is in memory?

One of the two major classes of viruses, called *boot sector infectors*, targets the vulnerable boot areas of the hard disk. (The other major group of viruses attacks individual files.) Some infect the code in the **master boot record** while others infect the code in the volume boot sector(s). By infecting this code, the virus assures itself of always being able to load into memory when the machine is booted, as long as you boot from the volume that is infected. There is always code in any disk (hard or floppy) that is formatted, whether or not the system files are present on the disk.

Many people think that when you boot a system from a floppy or hard disk that has no system files--because it was formatted without transferring them--that the system doesn't boot. It's worth pointing out that in truth, it *does* boot; the boot process just halts very quickly when no operating system files can be found, **as described here**. In fact, the error message "Non-system disk or disk error - Replace and press any key when ready", is printed by the volume boot code that is read from the volume boot sector on the disk.

The importance of this distinction has to do with the spread of viruses. Since the volume boot code is always executed when the system attempts to boot from a device, a virus can be present on a floppy disk even if you don't format it with the system files on it using "FORMAT /S" or the "SYS" command. As soon as you see the "Non-system disk..." message, the virus could already be in your system memory.

8 FAT File System Disk Volume Structures

The highest-level logical disk structures are the master boot record and partition tables, which define the way the entire disk is sized and organized. The next level down is the definition of each disk volume's structures. This is where we get down to the actual files and directories that are stored on the disk. This section takes a look at the disk volume structures used in the FAT file system.

8.1 Volume Boot Sector

The *volume boot sector* is the "master control" for the disk volume, containing information about what the volume contains, and the volume boot program that is executed when the volume is booted (if it is bootable).

Each DOS partition (also called a DOS volume) has its own *volume boot sector*. This is distinct from the *master* boot sector (or record) that controls the entire disk, but is similar in concept. Each volume boot sector contains the following:

- **Disk Parameter Block:** Also sometimes called the *media parameter block*, this is a data table that contains specific information about the volume, such as its specifications (size, number of sectors it contains, etc.), label name, etc.
- **Volume Boot Code:** This is code that is specific to the operating system that is using this volume and is used to start the load of the operating system. This code is called by the master boot code that is stored in the master boot record, but only for the primary partition that is set as active. For other partitions, this code sits unused.

The volume boot sector is created when you do a **high-level format of a hard disk partition**. The boot sector's code is executed directly when the disk is booted, making it a **favorite target for virus writers**.

After low-level formatting is complete, we have a disk with tracks and sectors--but nothing written on them**.** ***High-level formatting*** is the process of writing the file system structures on the disk that let the disk is used for storing programs and data. If you are using DOS, for example, the **DOS FORMAT command** performs this work, writing such structures as the master boot record and file allocation tables to the disk. High-level formatting is done after the hard disk has been **partitioned**, even if only one partition is to be used. See here for a full description of **DOS structures**, also used for Windows 3.x and Windows 9x systems.

The distinction between high-level formatting and low-level formatting is important. It is not necessary to low-level format a disk to erase it: a high-level format will suffice for most purposes; by wiping out the control structures and writing new ones, the old information is lost and the disk appears as new. (Much of the old data is still on the disk, but the access paths to it have been wiped out.) Under some circumstances a high-level format won't fix problems with the hard disk and a **zero-fill utility** may be necessary.

Different operating systems use different high-level format programs, because they use different file systems. However, the low-level format, which is the real place where tracks and sectors are recorded, is the same.

8.2 File Allocation Tables

Already discussed….above

8.3 Files, Directories, Paths and the Directory Tree

The basis for the storage model on the PC, at a logical level, is the *file*. The universal concept of the file is one of the strengths of the PC file system, and is one that the PC shares with many other successful file systems such as that used by UNIX. A file is simply a collection of bytes stored together with a name to identify it. A file can contain anything: program code, data, pictures, a movie, you name it. The meaningfulness of a file, and what it is used for, is determined by what you put in it, and what software you use it with.

As you probably know, files are stored in virtually every PC-based operating system using a paradigm known as the *directory tree*. The "base" of the tree is the (somewhat appropriately named) root directory. Within the root directory you can create either files or more directories (often called subdirectories). Each directory is a container that holds files or more subdirectories (or both). Taken as a whole, the structure of directories and subdirectories forms a logical tree.

Each file or directory on the hard disk can be uniquely identified using two pieces of information: its filename, and the *path* along the directory tree that you traverse to get to it. For example, let's suppose you have a set of customer information files organized by region. The entire database is in a directory called "Customers". Within this is a directory for each state. Within each of these state directories is a text file called "customer-list.txt". The file containing information about your customers in Texas would then be addressed as "\Customers\Texas\customer-list.txt". The filename is "customer-list.txt", and the path is "\Customers\Texas".

8.4 Internal Directory Structures

Every file on the system is stored in a *directory*. A directory is nothing more than a file itself, except that it is specially structured and marked on the disk so that it has special meaning. A directory is a table that contains information about files (and subdirectories) that it contains, and links to where the file (or subdirectory) data begins on the disk. The paper analogy would be a table of contents to a book, except that directories of course use a **hierarchical tree structure** and books do not.

Each entry in a directory is 32 bytes in length, and stores the following information:

? **File Name and Extension:** This is the 11-character name of the file using the conventional 8.3 DOS file naming standard, for example, COMMAND.COM. Note that the "dot" in "COMMAND.COM" is implied and not actually stored on the disk. See here for more on file naming and also on VFAT long file names, **which use a special structure**. The file name field is also used to indicate **directory entries that have been deleted**.

? **File Attribute Byte:** There are several different attributes which the operating system uses to give special treatment to certain files; these are stored in a single byte in each directory entry. **These attributes are discussed in detail here**. Note that it is one of these file attributes that indicates whether an entry in the directory represents a "real" file, or a subdirectory.

? **Last Change Date/Time:** There is a space for each file to indicate the date and time that it was created or modified. You should know that these fields can be arbitrarily modified by any program to be whatever they want, so this date/time shouldn't be taken too religiously. I occasionally am asked if the date/time on a file can be used to prove when someone did something or not on their PC. It cannot, because it's too easy to change this information.

? **File Size:** The size of the file in bytes.

? **Link to Start Cluster:** The number of the cluster that starts the file (or subdirectory) is stored in the directory. This is what allows the operating system to find a file when it is needed, and how all the different files and directories are linked together on the disk. **See here for more on cluster chaining**.

Every regular directory on the disk has two special entries, which refer to the directory itself and to the parent directory. These are named "." (Single dot) and "...” (Double dot) respectively. These entries are used for navigation purposes; if you type "chdir ..." then DOS will change your current directory to the parent of the one you were in.

8.5 Root Directory and Regular Directories

The directory at the "base" of the directory structure that defines the logical tree that organizes files on a hard disk is the *root directory*. The root directory is special because it follows special rules that do not apply to the other, "regular" directories on the hard disk.

There can only be one root directory for any disk volume; obviously, having more than one would result in chaos, and there isn't any need to have more than one anyway. In order to "anchor" the directory tree, the root directory is fixed in place at the start of the DOS volume. It is located directly below the two copies of the **FAT**, which is itself directly below the other key disk structures. This contrasts with regular (sub) directories, which can be located anywhere on the disk.

In addition to being fixed in location, the root directory is also fixed in size. Regular directories can have an arbitrary size; they use space on the disk much the way files do, and when more space is needed to hold more entries, the directory can be expanded the same way a file can. The root directory is limited to a specific number of entries because of its special status. The number of entries that the root directory can hold depends on the type of volume:

Volume Type	Maximum Number of Root Directory Entries
360KB 5.25" Floppy Disk	112
720KB 3.5" Floppy Disk	112
1.2MB 5.25" Floppy Disk	224
1.44MB 3.5" Floppy Disk	224
2.88MB 3.5" Floppy Disk	448
Hard Disk	512

Note that the newer FAT32 version of the FAT file system does *not* have the restriction on placement and size of the root directory. In this enhancement the root directory is treated like a regular directory and can be relocated and expanded in size like any other.

There are a couple of other special things about the root directory. One is that it cannot be deleted; the reason for this I would think to be obvious. Also, the root directory has no parent, since it is at the top of the tree structure. The root directory still contains a "..." entry, but instead of pointing to the cluster number of the parent directory like a regular directory's parent entry, it contains a null value (zero).

8.6 File Names and Extensions

As virtually every PC user knows, standard PC files are named using a fixed format convention that has been around since the "beginning of time" (i.e., the first IBM PC). The file name is comprised of two parts:

? **File Name:** The base name of the file itself. This part of the file name must be between one and eight characters in length. A special code is used as the first character of the file name to indicate **deleted files**.

? **File Extension:** The extension of the file, which is optional and hence can be from zero to three characters.

The extension can be thought of as a "file type" of sorts. It tells you--and your computer--at a glance what kind of file you are looking at. For example, a file with an extension of "EXE" is normally an executable program file, "HTM" usually means an HTML document, and "BAT" a DOS batch file.

In actual fact, there is nothing special at all about these extension names. The reason I say an EXE file is *normally* an executable file is that the use of the EXE extension to refer to executable files is a convention, and not anything that is enforced by the system. You could open up your editor and type "This is a test", and when you go to save the file, save it as "TEST.EXE", and this will work perfectly fine.

So why are file extensions used? They are essentially a shorthand way of organizing files by type, and they are used by various pieces of software, including DOS itself, to indicate which programs should be used with which files, without having to look into the structure of the file itself. The reason that having an EXE extension on a file matters is that DOS is pre-programmed so that when you type the name of a file, it will look for that file with certain types of extensions to try to execute them. One of those is "EXE". So if you create this silly "TEST.EXE" file with a text line in it and then type "TEST" at the command line, DOS will try to run this "program". What do you think will happen when it does? (Hint: don't try this if you have any unsaved work open anywhere else on your PC...)

Similarly, other programs usually try by default to look only at files that have extensions that they are meant to use. If you run Microsoft Word and go to the "Open" dialog box, by default it will look for files with an extension of "DOC", for "document". This is why using consistent file extensions is important.

This use of file extensions by software programs is quite prevalent, and there are in fact hundreds of different kinds in use. In fact, when you double-click on a file in Windows 95's Windows Explorer, for example, it will automatically launch the program that it knows uses the file you selected, and tell the program to open the file you clicked on. Just remember that Explorer is only determining the program to use by looking at the file extension, and not by analyzing anything within the file itself. (Also, many programs use the same file extensions, which can be confusing at times. If you've ever had Internet Explorer and Netscape Navigator fighting for the "right" to be associated with your "HTM" files you know exactly what I mean.)

The following characters are allowed in legal DOS file names: *A-Z 0-9 $ % ' - _ @ ~ ` ! ( )* ^ # & Note that a space *is* an officially valid character for a file name, but I strongly recommend against using spaces in standard file names because many programs become very confused by file names with spaces in them. Even Windows 95 specifically avoids using spaces in file names when it creates 8.3 aliases for its **long file names**.

8.7 Long File System

Until the release of Windows 95, all file names using DOS or Windows 3.x were limited to the standard eight character file name plus three character file extension. This restriction tends to result in users having to create incredibly cryptic names, and having the situation still like this 15 years after the PC was invented seemed laughable, especially with Microsoft wanting to compare its ease of use to that of the Macintosh. Users want to name their files "Mega Corporation - fourth quarter results.DOC", not "MGCQ4RST.DOC", because the second name will mean zippo to the user a few months after they create it.

Microsoft was determined to bring *long file names* (*LFNs*) to Windows 95 much as it had for Windows NT. The latter, however, has a new file system designed from the ground up to allow long file names. Microsoft had a big problem on its hands with Windows 95: it wanted to maintain compatibility with existing disk structures, older versions of DOS and Windows, and older applications. It couldn't just "toss out" everything that came before and start fresh, because doing this would have meant no older programs could read any files that used the new long file names. File names were restricted to "8.3" (standard file name sizes) within the directories on the disk.

What Microsoft needed was a way to implement long file names so that the following goals were all met:

? Windows 95 and applications written for Windows 95 could use file names much longer than 11 total characters.

? The new long file names could be stored on existing DOS volumes using standard directory structures, for compatibility.

? Older pre-Windows-95 software would still be able to access the files that use these new file names, somehow.

The VFAT file system accomplishes these goals, for the mostpart, as follows. Long file names of up to 255 characters per file can be assigned to any file under Windows 95 or by any program written for Windows 95 (although file names under 100 characters are recommended so that they don't get cumbersome to use). Support for these long file names is also provided by the version of DOS (7.x) that comes with Windows 95. **File extensions** are maintained, to preserve the way that they are used by software. The long file name is limited to the same characters as standard file names are, except that the following additional characters are allowed: +*, ; = [ ]*.

To allow access by older software, each file that uses a long file name also has a standard file name *alias* that is automatically assigned to it. This is done by truncating and modifying the file name as follows:

? The long file name's extension (up to three characters after a ".") are transferred to the extension of the alias file name.
? The first six non-space characters of the long file name are analyzed. Any characters that are valid in long file names but not in standard file names (+,; = [and]) are replaced by underscores. All lower-case letters are converted to upper case. These six characters are stored as the first six characters of the file name.
? The last two characters of the file name are assigned as "~1". If that would cause a conflict because there is already a file with this alias in the directory, then it tries "~2", and so on until it finds a unique alias.

So to take our example from before, "Mega Corporation - fourth quarter results.DOC" would be stored as shown, but also under the alias "MEGACO~1.DOC". If you had previously saved a file called "Mega Corporation - third quarter results.DOC" in the same directory, then *that* file would be "MEGACO~1.DOC" and the new one would be "MEGACO~2.DOC". Any older software can reference the file using this older name. Note that using spaces in long file names really doesn't cause any problems because Windows 95 applications are designed knowing that they will be commonly used, and because the short file name alias has the spaces removed.

Long file names are stored in regular directories using the standard directory entries, but using a couple of tricks. The Windows 95 file system creates a standard directory entry for the file, in which it puts the short file name alias. Then, it uses several additional directory entries to hold the rest of the long file name. A single long file name can use many directory entries (since each entry is only 32 bytes in length), and for this reason it is recommended that long file names not be placed in the **root directory**, where the total number of directory entries is limited.

In order to make sure that older versions of DOS don't get confused by this non-standard usage, each of the extra directory entries used to hold long file name information is tagged with the following odd combination of **file attributes**: read-only, hidden, system and volume label. The objective here is to make sure that no older versions of DOS try to do anything with these long file name entries, and also to make sure they don't try to overwrite these entries because they think they aren't in use. That combination of file attributes causes older software to basically ignore the extra directory entries being used by VFAT.

While long file names are a great idea and improve the usability of Windows 95, Microsoft's streeeeeetch to keep them compatible with old software kind of shows. Basically, the implementation is a *hack* built on top of the standard FAT file system, and there are numerous problems that you should be aware of when using LFNs:

? **Compatibility Problems with Older Utilities:** While marking its extra entries as read-only, hidden, system and volume label will trick standard applications into leaving these entries alone, a disk utility program like Norton Disk Doctor will not be fooled. If you use the DOS version that is not aware of long file names, it will detect these entries as errors on your disk and happily "correct" them for you, and that's that for your long file names. Utilities run under Windows 95 must be aware of long file names to work properly.

- **"Loss" of Long File Names with Older Software:** While older apps will work with the long file names using the short name alias, they have no ability to access the long file name at all. It is easy for one of these applications to "drop" the long file name by accident. A common cause of frustration is that if you use older DOS backup software that doesn't know about long file names, it will save only the alias, and if you have a crash and need to restore, the long file names will be lost.
- **Problems with Conflicting Alias File Names:** There are two problems with the long file name aliasing scheme. The first is that the alias is not permanently linked to the long file name, and can change. Let's suppose we take our "Mega Corporation - fourth quarter results.DOC" and save it in a new empty directory. It will be assigned the alias "MEGACO~1.DOC". Now let's say we copy it to a directory that already has the file "Mega Corporation - Water Cooler Policy.DOC" in it, which is using the same "MEGACO~1.DOC" alias. When we do this, the alias for the fourth quarter results file will magically change (well, the operating system does it) to "MEGACO~2.DOC". This can confuse some people who refer to the file both by the long name and the alias. The second problem is more serious. Let's replay this same scenario using an older file copy application that isn't aware of long file names. Since all it sees is "MEGACO~1.DOC" in two different places, it thinks they are the same file, and will overwrite the water cooler memo with the fourth quarter results when you do the copy! If you are lucky it will ask "Are you sure?" first; otherwise...
- **Problems with Short File Name Re-aliasing:** Copying a file with a long file name from one partition to another, or restoring one from a backup, can cause the short file name alias associated with the long file name to be *changed*. This can cause spurious behavior when hard-coded references to the short file name no longer resolve to the correct target. Note that despite the fact that Windows NT's NTFS file system was rebuilt from the ground up, it has this problem as well because the system aliases long file names for limited backward compatibility.

Overall, long file names are a useful advance in the usability of the FAT file system, but you need to be aware of problems when using them, especially with older software.

8.8 FILE ATTRIBUTE

Each file is stored in a directory, and uses a directory entry that describes its characteristics such as its name and size, and also contains a pointer to where the file is stored on disk. One of the characteristics stored for each file is a set of *file attributes* that give DOS and application software more information about the file and how it is intended to be used.

The use of attributes is "voluntary". What this means is that any software program can look in the directory entry to discern the attributes of a file, and based on them, make intelligent decisions about how to treat the file. For example, a file management program's delete utility, seeing a file marked as a read-only system file, would be well-advised to at least warn the user before deleting it. However, it doesn't have to. Any program that knows what it is doing can override the attributes of a file, and certainly, viruses will do this routinely.

That said, DOS and most other operating systems assign definite meanings to the attributes stored for files, and will alter their behavior according to what they see. If at a DOS prompt you type "DIR" to list the files in the directory, by default you will not see any files that have the "hidden" attribute set. You have to type "DIR /AH" to see the hidden files.

A file can have more than one attribute attached to it, although only certain combinations really make any sense. The attributes are stored in a single byte, with each bit of the byte representing a specific attribute (actually, only six bits are used of the eight in the byte). Each bit that is set to a one means that the file has that attribute turned on. (These are sometimes called *attribute bits* or *attribute flags*). This method is a common way that a bunch of "yes/no" parameters are stored in computers to save space. The following are the attributes and the bits they use in the attribute byte:

Attribute	Bit Code
Read-Only	00000001
Hidden	00000010
System	00000100
Volume Label	00001000
Directory	00010000
Archive	00100000

So, the attribute byte for a hidden, read-only directory would be 00010011, which are simply the codes for those three attributes from the table above, added together. Here is a more detailed description of what these attributes mean (or more accurately, how they are normally used). Note that each of the attributes below applies equally to files and directories (except for the directory attribute of course!):

- **Read-Only:** Most software, when seeing a file marked read-only, will refuse to delete or modify it. This is pretty straight-forward. For example, DOS will say "Access denied" if you try to delete a read-only file. On the other hand, Windows Explorer will happily munch it. Some will choose the middle ground: they will let you modify or delete the file, but only after asking for confirmation.
- **Hidden:** This one is pretty self-explanatory as well; if the file is marked hidden then under normal circumstances it is hidden from view. DOS will not display the file when you type "DIR" unless a special flag is used, as shown in the earlier example.
- **System:** This flag is used to tag important files that are used by the system and should not be altered or removed from the disk. In essence, this is like a "more serious" read-only flag and is for the most part treated in this manner.
- **Volume Label:** Every disk volume can be assigned an identifying label, either when it is formatted, or later through various tools such as the DOS command "LABEL". The volume label is stored in the root directory as a file entry with the label attribute set.
- **Directory:** This is the bit that differentiates between entries that describe files and those that describe subdirectories within the current directory. In theory you can convert a file to a directory by changing this bit, but of course in practice trying to do this would result in a mess because the entry for a directory has to be in a specific format.
- **Archive:** This is a special bit that is used as a "communications link" between software applications that modify files, and those that are used for backup. Most backup software allows the user to do an incremental backup, which only selects for backup any files that have changed since the last backup. This bit is used for this purpose. When the backup software backs up ("archives") the file, it clears the archive bit (makes it zero). Any software that modifies the file subsequently, is supposed to set the archive bit. Then, the next time that the backup software is run, it knows by looking at the archive bits which files have been modified, and therefore which need to be backed up. Again, this use of the bit is "voluntary"; the backup software relies on other software to use the archive bit properly; some programs could modify the file without setting the archive attribute, but fortunately most software is "well-behaved" and uses the bit properly.

Most of the attributes for files can be modified using the DOS ATTRIB command, or by looking at the file's properties through the Windows 95 Windows Explorer or other similar file navigation tools.

9 Clusters and File Allocation

One of the most important things that an operating system does is to manage the use of disks and other storage media. (After all, DOS does stand for *Disk* Operating System). The key question is: given that we have a disk that can store a certain amount of information, and a bunch of files and directories (of varying sizes and descriptions) that need to use this space, how do we organize and manage it efficiently?

Various operating systems have come up with different ways of managing the organization and allocation of disk space to files. This section describes how the FAT file system allocates space to files and manages the use of each disk volume.

9.1 Clusters (Allocation Units)

As described here, the smallest unit of space on the hard disk that any software can access is the **sector**, which contains 512 bytes. It is possible to have an allocation system for the disk where each file is assigned as many individual sectors as it needs. For example, a 1 MB file would require approximately 2,048 individual sectors to store its data.

Under the FAT file system (and in fact, most file systems) individual sectors are not used. There are several performance reasons for this. It can get cumbersome to manage the disk when files are broken into 512-byte pieces. A 2 GB disk volume using 512 byte sectors managed individually would contain over 4 million individual sectors, and keeping track of this many pieces of information is time- and resource-consuming. Some operating systems *do* allocate space to files by the sector, but they require some advanced intelligence to do this properly. FAT was designed many years ago and is a *simple* file system, and is not capable of managing individual sectors.

What FAT does instead is to group sectors into larger blocks that are called *clusters*, or *allocation units*. The cluster size is determined primarily by the size of the disk volume: generally speaking, larger volumes use larger cluster sizes. For hard disk volumes, each cluster ranges in size from 4 sectors (2,048 bytes) to 64 sectors (32,768 bytes). Floppy disks use much smaller clusters, and in some cases use a cluster of size of just 1 sector. The sectors in a cluster are continuous, so each cluster is a continuous block of space on the disk.

Cluster sizing (and hence partition or volume size, since they are directly related) has an important impact on performance and disk utilization. The cluster size is determined when the disk volume is partitioned. Certain utilities (like Partition Magic) can alter the cluster size of an existing partition (within limits) but for the most part, once the partition size is selected it is fixed.

Every file must be allocated an integer number of clusters--a cluster is the smallest unit of disk space that can be allocated to a file, which is why clusters are often called allocation units. This means that if a volume uses clusters that contain 8,192 bytes, an 8,000 byte file uses one cluster (8,192 bytes on the disk) but a 9,000 byte file uses two clusters (16,384 bytes on the disk). This is why cluster size is so important in making sure you maximize the efficient use of the disk--larger cluster sizes result in more wasted space.

9.2 File Chaining and FAT Cluster Allocation

The **file allocation table (FAT)** is used to keep track of which clusters are assigned to each file. The operating system (and hence any software applications) can determine where a file's data is located by using the directory entry for the file and file allocation table entries. Similarly, the FAT also keeps track of which clusters are open and available for use. When an application needs to create (or extend) a file, it requests more clusters from the operating system, which finds them in the file allocation table.

There is an entry in the file allocation table for each cluster used on the disk. Each entry contains a value that represents how the cluster is being used. There are different codes used to represent the different possible statuses that a cluster can have.

Every cluster that is in use by a file has in its entry in the FAT a cluster number that links the current cluster to the next cluster that the file is using. Then that cluster has in its entry the number of the cluster after *it*. The last cluster used by the file is marked with a special code that tells the system that it is the last cluster of the file; this is often a number like 65,535 (16 ones in binary format). Since the clusters are linked one to the next in this manner, they are said to be *chained*. Every file (that uses more than one cluster) is chained in this manner. See the example that follows for more clarification.

In addition to a cluster number or an end-of-file marker, a cluster's entry can contain other special codes to indicate its status. A special code, usually zero, is put in the FAT entry of every open (unused) cluster. This tells the operating system which clusters are available for assignment to files that need more storage space. Another code is used to indicate "bad" clusters. These are clusters where a disk utility (or the user) has previously detected one or more unreliable sectors, due to disk defects. These clusters are marked as bad so that no future attempts will be made to use them.

Accessing the entire length of a file is done by using a combination of the file's directory entry and its cluster entries in the FAT. This is confusing to describe, so let's look at an example. Let's consider a disk volume that uses 4,096 byte clusters, and a file in the C:\DATA directory called "PCGUIDE.html" that is 20,000 bytes in size. This file is going to require 5 clusters of storage (because 20,000 divided by 4,096 is around 4.88).

OK, so we have this file on the disk, and let's say we want to open it up to edit it. We open our editor and ask for the file to be opened. To find the cluster on the disk containing the first part of the file, the system just looks at the file's directory entry to find the starting cluster number for the file; let's suppose it goes there and sees the number 12,720. The system then knows to go to cluster number 12,720 on the disk to load the first part of the file.

To find the second cluster used by this file, the system looks at the FAT entry for cluster 12,720. There, it will find another number, which is the next cluster used by the file. Let's say this is 12,721. So the next part of the file is loaded from cluster 12,721, and the FAT entry for 12,721 is examined to find the next cluster used by the file. This continues until the last cluster used by the file is found. Then, the system will check the FAT entry to find the number of the next cluster, but instead of finding a valid cluster number, it will find a special number like 65,535 (special because it is the largest number you can store in 16 bits). This is the signal to the system that "there are no more clusters in this file". Then it knows it has retrieved the entire file.

Since every cluster is chained to the next one using a number, it isn't necessary for the entire file to be stored in one continuous block on the disk. In fact, pieces of the file can be located anywhere on the disk, and can even be moved after the file has been created. Following these chains of clusters on the disk is done invisibly by the operating system so that to the user, each file appears to be in one continuous chunk of disk space.

9.3 File Deletion & Un-Deletion

One of the advantages of the FAT file system is the ease with which it allows for files to be undeleted, because of the way that it deletes files. Contrary to what many people believe, deleting a file does not result in the contents of the file actually being removed from the disk. All that the system does is mark the file as deleted.

When you delete a file, the system doesn't really delete the file. It places the hex byte code E5h into the first letter of the file name of the file. This is a special tag that tells the system "this file has been deleted". The space that was formerly used by the file is available for use by other files, but it is not cleared. It is just sort of "left there".

Over time, these clusters will eventually probably be reused by other files as they request more clusters for storage. However, if you accidentally delete a file you can very often recover it if you act quickly. If you run a utility like DOS's UNDELETE or Norton Utilities' UNERASE immediately, it can identify and recover the deleted files in a directory. As long as you provide it with the missing first character of the file name (which was overwritten by the E5h code when the file was deleted), it may be able to recover all or most of the file.

The less you do between the time the file is deleted and the time when you try to undelete it, the more likely you will be able to recover the file. Obviously, if you **de-fragment your disk** or does some other large-scale disk work, you will most likely lose the file's contents forever. Finally, many utilities will protect your files after being deleted so they can be recovered easily without worrying about the disk space being reused. For example, Windows 95 sends all deleted files initially to its "Recycle Bin", from which they can be restored if needed. Norton Utilities also includes a form of protection for recently-deleted files.

9.4 Fragmentation and Defragmentation

Since each file is stored as a linked list of clusters, the data that is contained in a file can be located anywhere on the disk. If you have a 10 MB file stored on a disk using 4,096-byte clusters, it is using 2,560 clusters. These clusters can be on different tracks, different platters of the disk, in fact, they can be anywhere.

However, even though a file can be spread all over the disk, this is far from the preferred situation. The reason is performance. As discussed in the **section describing performance**, hard disks are relatively slow devices, mainly because they are mechanical (they have moving parts--your processor, chipset, memory and system bus do not). Each time the hard dsk has to move the heads to a different track, it takes time that is equivalent to thousands and thousands of processor cycles.

Therefore, we want to minimize the degree to which each file is spread around the disk. In the ideal case, every file would in fact be completely contiguous--each cluster it uses would be located one after the other on the disk. This would enable the entire file to be read, if necessary, without a lot of mechanical movement by the hard disk. There are in fact utilities that can optimize the disk by rearranging the files so that they are contiguous. This process is called *defragmentation* or *defragmenting*. The utilities that do this are, unsurprisingly, called *defragmenters*. The most famous one is Norton's SpeedDisk, and Microsoft now includes a DEFRAG program for DOS and a built-in defragmenter for Windows 95 as well.

So the big question is: how does fragmentation occur anyway? Why not just arrange the disk so that all the files are always contiguous? Well, it is in many cases a gradual process--the file system starts out with all or most of its file contiguous, and becomes more and more *fragmented* as a result of the creation and deletion of files over a period of time.

To illustrate, let's consider a very simple example using a teeny hard disk that contains only 12 clusters. The table below represents the usage of the 12 clusters. Initially, the table is empty:

(cluster 1)	(cluster 2)	(cluster 3)	(cluster 4)	(cluster 5)	(cluster 6)
(cluster 7)	(cluster 8)	(cluster 9)	(cluster 10)	(cluster 11)	(cluster 12)

OK, now let's suppose that we create four files: file A takes up 1 cluster, file B takes 4, file C takes 2, and file D takes 3. We store them in the free available space, and they start out all contiguous, as follows:

A	B	B	B	B	C
C	D	D	D		

Next, we decide that we don't need file C, so we delete it. This leaves the disk looking like this:

A	B	B	B	B	
	D	D	D		

Then, we create a new file E that needs 3 clusters. Well, there are no contiguous blocks on the disk left that are 3 clusters long, so we have to split E into two fragments, using part of the space formerly occupied by C. Here's what the "disk" looks like now:

A	B	B	B	B	E
E	D	D	D	E	

Next, we delete files A and E and create file F which takes up 5 clusters. The disk now looks like this:

F	B	B	B	B	F
F	D	D	D	F	F

As you can see, file F ends up being broken into three fragments. This is a highly simplified example of course, because real disks have thousands of files and thousands of clusters, so the problem there is magnified. This gives you the general idea of what happens though. What a defragmentation program does is to rearrange the disk to get the files back into contiguous form. After running the utility, the disk would look something like this:

B	B	B	B	F	F
F	F	F	D	D	D

9.5 FAT File System Errors

As a result of how the FAT file system allocates space and chains file together, there are several common types of errors that can crop up over time. Note that I am talking here about errors in the *logical structure* of the disk, not physical disk errors, bad sectors, etc. Most of these errors can be detected by using a standard disk error-checking program that analyzes the file system's integrity, such as Microsoft's SCANDISK or Norton's Disk Doctor (NDD). In fact, I highly recommend that every hard disk volume be scanned for routine file system problems on a daily basis.

File system errors are occasionally the result of corruption on the disk that can have at its root a real hardware problem. These errors can therefore result from any system problem that can cause disk corruption, such as resource conflicts, bad drivers, etc. Far more often, however, file system problems occur as a result of a software problem. Program crashes, for example, often leave around clusters that had space allocated to them but not assigned to a file.

A power failure on a PC running Windows will often result in one or more file system errors due to files not being closed properly. This is why you are always supposed to exit Windows before shutting down a PC. It is also why the newest version of Windows 95 (OEM SR2) automatically scans the disk for errors when it starts, if it detects that Windows ended without doing a proper file system shutdown.

The following are the most common errors encountered on a FAT disk:

? **Lost Clusters:** Virtually every DOS user has come across this problem. Lost clusters are simply clusters that are marked in the FAT as being in use, but that the system cannot link to any file. **Every file consists of a series of clusters** that can be traced by starting with the directory entry and following the linked list of clusters to the end of the file. Disk checking programs can check an entire disk volume for lost clusters using the following procedure (or something similar to it). First, create a copy in memory of the FAT, noting all of the clusters marked as in use. Starting at the root directory, trace through the clusters used by each file and mark them as "accounted for", since they have been seen to be connected to a file. Then do the same for all the subdirectories of the root directory, and then *their* subdirectories, and so on. When finished, every cluster that is marked in the FAT as in use should be accounted for. Any that are in use but not accounted for are "orphans" that don't belong to any file. Lost clusters are usually the result of interrupted file activity of some sort. The program that detects them will usually give you the choice of clearing them (marking them as "available" and returning them to the pool of free clusters) or saving them as a file. In the latter case, the program generates a file name and links the lost clusters to that name, so that a real file is formed. Usually this file will then be corrupted or damaged in some way, but you can often at least see what this orphaned data was and in some cases, recover it.

? **Cross-Linked Files:** On rare occasions, two files can end up pointing to the same data on the disk. Both files will have the starting cluster number in the directory entry pointing to the same cluster number. Obviously this is a problem, since each time you use either file, you will overwrite the other one. The only solution to this problem is to make new copies of each of the affected files. You will generally lose the contents of one or the other of the files (in fact, by the time you discover this problem, you have already lost the contents of one of them, since no sector can contain information from two files at the same time).

? **Invalid Files or Directories:** Very rarely, the internal structures of file or directories can become damaged so that some entries are no longer following the "rules" for how a file or directory is supposed to be laid out. An example would be a directory that doesn't have a pointer to its parent directory, or a file that has an invalid start cluster. Sometimes files get assigned an invalid date or time by a buggy piece of software. These problems can usually be fixed by the disk scanning software.

? **Allocation or FAT Errors:** Occasionally the entries in the FAT can become corrupted or set to invalid values. Again, most disk-checking utilities will detect and correct these sorts of problems on the fly.

10 Partitioning, Partition Sizes and Drive Lettering

10.1 FAT Sizes: FAT12, FAT16 and FAT32

Partitioning the hard disk is the act of dividing it into pieces; into logical volumes. This is one of the first things done when setting up a new hard disk, because partitions are one of the **major disk structures** that define how the disk is laid out. In fact, you must partition the hard disk, even if only "partitioning" it into a single partition, before you can format and use the disk.

The choice of how the disk is partitioned is important because partition size has an important impact on both performance and on how efficiently the disk's space is utilized. Even though in many cases you can fit an entire disk into one partition, it is rare that you will actually want to do this, for performance reasons. This section explains the reasons why, and also examines the commonly asked question: should I use FAT32?

The file allocation table or **FAT** stores information about the clusters on the disk in a table. There are three different varieties of this file allocation table, which vary based on their size. The system utility that you use to partition the disk will normally choose the correct type of FAT for the volume you are using, but sometimes you will be given a choice of which you want to use.

Since each cluster has one entry in the FAT, and these entries are used to hold the cluster number of the next cluster used by the file, the size of the FAT is the limiting factor on how many clusters any disk volume can contain. The following are the three different FAT versions now in use:

- **FAT12:** The oldest type of FAT uses a 12-bit binary number to hold the cluster number. A volume formatted using FAT12 can hold a maximum of 4,086 clusters, which is 2^12 minus a few values (to allow for reserved values to be used in the FAT). FAT12 is therefore most suitable for smaller volumes, and is used on floppy disks and hard disk partitions smaller than about 16 MB.
- **FAT16:** The FAT used for most hard disk partitions uses a 16-bit binary number to hold cluster numbers. When you see someone refer to a "FAT" volume generically, they are usually referring to FAT16, because it is the de facto standard for hard disks. A volume using FAT16 can hold a maximum of 65,526 clusters, which is 2^16 less a few values (again for reserved values in the FAT). FAT16 is used for hard disk volumes ranging in size from 16 MB to 2,048 MB.
- **FAT32:** The newest FAT type, FAT32 is supported by Windows 95's OEM SR2 release, as well as Windows 98. FAT32 uses a 28-bit binary cluster number--*not* 32, because 4 of the 32 bits are "reserved". 28 bits is still enough to permit ridiculously huge volumes--FAT32 can theoretically handle volumes with over 268 million clusters, and will support (theoretically) drives up to 2 TB in size. However to do this the size of the FAT grows very large; **see here for details on FAT32's limitations**.

Here's a summary table showing how the three types of FAT compare:

Attribute	FAT12	FAT16	FAT32
Used For	Floppies and small hard disk volumes	Small to large hard disk volumes	Medium to very large hard disk volumes
Size of Each FAT Entry	12 bits	16 bits	28 bits
Maximum Number of Clusters	4,086	65,526	~268,435,456
Cluster Size Used	0.5 KB to 4 KB	2 KB to 32 KB	4 KB to 32 KB
Maximum Volume Size	16,736,256	2,147,123,200	about 2^41

10.2 FAT Partition Efficiency: Slack

One issue related to the FAT file system that has gained a lot more attention recently is the concept of *slack*. As larger and larger hard disks are being shipped on systems--especially low-end systems that are not being divided into multiple partitions--users have begun noticing that large quantities of their hard disk seem to "disappear". In many cases this can amount to hundreds of megabytes.

Of course the space doesn't really disappear (actually, in some cases lost clusters can make space really unusable on a disk unless you use a scanning utility to recover it). The space is simply wasted as a result of the cluster system that FAT uses. A cluster is the minimum amount of space that can be assigned to any file. No file can use part of a cluster under the FAT file system.

This means, essentially, that the amount of space a file uses on the disk is "rounded up" to an integer multiple of the cluster size. If you create a file containing exactly one byte, it will still use an entire cluster's worth of space. Then, you can expand that file in size until it reaches the maximum size of a cluster, and it will take up no additional space. As soon as you make the file larger than what a single cluster can hold, a second cluster will be allocated, and the file's disk usage will double, even though the file only increased in size by one byte. Think of this in terms of collecting rain water in quart-sized glass bottles. Even if you collect just one ounce of water, you have to use a whole bottle. Once the bottle is in use, however, you can fill it with 31 more ounces, until it is full. Then you'll need another whole bottle to hold the 33rd ounce.

Since files are always allocated whole clusters, this means that on average, the larger the cluster size of the volume, the more space that will be wasted. (It's more efficient to use smaller, cup-sized bottles instead of quart-sized ones, if minimizing the amount of storage space is a concern). If we take a disk that has a truly random distribution of file sizes, then on average each file wastes half a cluster. (They use any number of whole clusters and then a random amount of the last cluster; so on average half a cluster is wasted). This means that if you double the cluster size of the disk, you double the amount of storage that is wasted. Storage space that is wasted in this manner, due to space left at the end of the last cluster allocated to the file is commonly called *slack*.

The situation is in reality usually worse than this theoretical average. The files on most hard disks don't follow a random size pattern; in fact most files tend to be small in size. (Take a look in your web browser's cache directory sometime.) A hard disk that uses more small files will in fact result in far more space being wasted. There are utilities that you can use to analyze the amount of wasted space on your disk volumes, such as the fantastic ***Partition Magic***. It is not uncommon for very large disks that are in single FAT partitions to waste up to 40% of their space due to slack, although 25-30% is more common.

Let's take an example to illustrate the situation. Let's consider a hard disk volume that is using 32 KB clusters. There are 17,000 files in the partition. If we assume that each file has half a cluster of slack, then this means that we are wasting 16 KB of space per file. Multiply that by 17,000 files, and we get a total of 265 MB of slack space. If we assume that most of the files are smaller, and so therefore on average each file has slack space of around two-thirds of a cluster instead of one-half, this jumps to 354 MB!

If we were able to use a smaller cluster size for this disk, the amount of space wasted would reduce dramatically. The table below shows a comparison of the slack for various cluster sizes for this example. The more files on the disk, the worse the slack gets. To consider the percentage of disk space wasted in this example, divide the slack figure by the size of the disk. So if this were a (full) 1.2 GB disk using 32 KB clusters, a full 30% of that space is slack. If the disk is 2.1 GB in size, the slack percentage is 17%:

Cluster Size	Sample Slack Space, 50% Cluster Slack Per File	Sample Slack Space, 67% Cluster Slack Per File
2 KB	17 MB	22 MB
4 KB	33 MB	44 MB
8 KB	66 MB	89 MB
16 KB	133 MB	177 MB
32 KB	265 MB	354 MB

As you can see, the larger the cluster size used, the more of the disk's space is wasted due to slack. Therefore, it is better to use smaller cluster sizes whenever possible. This is, unfortunately, often easier said than done. The number of clusters we can use is limited by the nature of the FAT file system, and there are also performance tradeoffs in using smaller cluster sizes. Therefore, it isn't always possible to use the absolute smallest cluster size in order to maximize free space.

Also realize that there will always be some space wasted regardless of the cluster size chosen. Most people consider the amount of slack obtained when using 4 KB or 8 KB partitions to be acceptable; most consider the slack of 32 KB cluster size partitions excessive; and the 16 KB partitions seem to go both ways. I personally only avoid the 32 KB partitions like the plague, but I (more than many others) also dislike having my disk broken into many pieces. **See this discussion of the tradeoffs between slack space waste and "end of volume" space waste** as well for more perspective on choosing cluster sizes.

Tip: Do remember not to go overboard in your efforts to avoid slack. To keep it all in perspective, let's take the worst case above, where 354 MB of space is wasted. With the cost per megabyte of disk now below 10 cents, this means that the "cost" of this problem is well below $50. That doesn't mean that wasting hundreds of megabytes of storage is smart; obviously I don't think that or I wouldn't have written so much about slack and partitioning. 🙂But on the other hand, spending 20 hours and $100 on utility software may not be too smart either. Moderation is often the key to using partitioning to reduce slack, so don't be taken in by some of the "partitioning fanatics" who seem to have lost sight of the fact that disk space is really very cheap today.

10.3 Relationship of Partition Size and Cluster Size

Since the size of the FAT is fixed, there is a hard maximum on the number of clusters that the FAT can hold. The maximum depends on the **size of FAT used by the volume**. For FAT12 partitions it is only 4,086; for FAT16 it is 65,526, and for FAT32 it is in the hundreds of millions.

FAT12 is only used for small partitions, so it isn't really very interesting in terms of an analysis of partition and cluster size; virtually no hard disks today use partitions below 16 MB, which is FAT12's limit. FAT32, on the other hand, is only limited in its FAT size by performance considerations: we don't *want* a FAT with 100 million entries because this can cause serious performance problems.

The place where partition size and cluster size interact is in the use of FAT16, the standard FAT type used for hard disks between 16 MB and 2,048 MB in size. The partitioning utility that you use to create your disk partitions will normally make the decision of what cluster size to use, by trying to use the smallest possible clusters in order to reduce **slack**. Since FAT16 is limited in size to 65,525 clusters, what happens is that each cluster size has a maximum partition size that it can support. If you want to make a partition larger than that maximum, you are forced to go up to the next larger cluster size.

The table below shows the FAT16 partition limits. The values are simply 65,526 times the cluster size, in bytes. It's important to realize that since there are only 65,526 clusters maximum, which is less than 64K (65,536), the maximum partition sizes fall short of the "round" MB numbers that everyone always uses. Thus the maximum partition size for 8 KB clusters is 511.92 MB, not 512 MB:

Cluster Size	Maximum Partition Size (binary MB)	Maximum Partition Size (bytes)
2 KB	127.98 MB	134,197,248
4 KB	255.96 MB	268,394,496
8 KB	511.92 MB	536,788,992
16 KB	1,023.84 MB	1,073,577,984
32 KB	2,047.69 MB	2,147,155,968

Note: Windows NT supports a 64 KB cluster size in FAT16, allowing a maximum partition of just under 4,096 MB. The amount of slack waste in a partition of this size is astronomical, and since the 64 KB cluster partitions aren't supported by Windows 95 or other FAT-using operating systems, this isn't a popular option. I would recommend using NTFS under NT for a partition this large.

Despite what you will read in many articles, there really is no hard-set *minimum* for these cluster sizes. (Mind you, I've never tried a really teeny partition, I'm sure there is *some* limit, but it's much smaller than most people usually say it is). It sometimes appears that 512 MB is the minimum size for a 16 KB cluster size partition, for example, because most utilities are pre-programmed to always pick the smallest possible cluster size, in order to cut down on slack. This means that if you use FDISK for example to create a partition that is 300 MB in size, it will always pick 8 KB for the cluster size and not 16 KB or 32 KB. But that doesn't mean you can't have a 300 MB partition that uses 32 KB clusters; Partition Magic will let you set this up if you want, and **in certain special cases you may in fact want to** (although usually not).

Note: FAT32 really does have a minimum size: it is only supported for partitions 512 MB or larger.

What does this all mean in terms of setting up your hard disk? It generally means that if you are using FAT16 (as opposed to FAT32) and have a hard disk that is greater than about 512 MB in size, you need to carefully consider how many partitions you want to use for it. If you are using a hard disk that is larger than 1,024 MB in size (this generally means a 1.2 GB or larger hard disk, since hard disk manufacturers specify disks in decimal GB) then you are *strongly* advised to partition your hard disk into more than one volume to avoid the tremendous slack space waste of partitions that use 32 KB clusters.

10.4 FAT32 Performance Tradeoff: FAT32 Cluster Sizes and FAT Sizes

It is generally believed to be a "rule" of cluster size selection that "smaller is better". As FAT16 partitions have gotten larger and **slack waste has gone through the roof**, the push toward using FAT32 to reduce cluster sizes has been tremendous. While FAT32 does allow the use of larger hard disks and greatly reduced cluster sizes, there is an important performance consideration in using FAT32 that is not often talked about.

Let's consider a partition that is just under 2,048 MB, the largest that FAT16 can support. If this partition is set up under FAT16, it will result in a file allocation table with 65,526 clusters in it, with each cluster taking up 32 KB of disk space. The large cluster size will indeed result in a great deal of wasted space on the disk in most cases. Therefore, often it will be recommended that FAT32 be used on this volume, which will result in the cluster size being knocked down from 32 KB to 4 KB. This will, in fact, reduce slack on the disk by an enormous amount, often close to 90%, and potentially free hundreds of megabytes of previously wasted disk space.

However, there is another side to this; you don't get this reduced cluster size for free. Since each cluster is smaller, there have to be more of them to cover the same amount of disk. So instead of 65,526 clusters, we will now have 524,208. Further more, the FAT entries in FAT32 are 32 bits wide (4 bytes), as opposed to FAT16's 16-bit entries (2 bytes each). The end result is that the size of the FAT is 16 times larger for FAT32 than it is for FAT16! The following table summarizes:

FAT Type	FAT16	FAT32
Cluster Size	32 KB	4 KB
Number of FAT Entries	65,526	524,208
Size of FAT	~ 128 KB	~ 2 MB

Still worse, if we increase the size of the FAT32 volume from 2 GB in size to 8 GB, the size of the FAT increases from around 2 MB to a rather hefty 8 MB. The significance of this is not the fact that the FAT32 volume will have to waste several megabytes of space on the disk to hold the FAT (after all, it is saving far more space than that by reducing slack a great deal). The real problem is that the FAT is referred to a *lot* during normal use of the disk, since it holds all the cluster pointers for every file in the volume. Having the FAT greatly increase in size can negatively impact system speed.

Virtually every system employs **disk caching** to hold in memory disk structures that are frequently accessed, like the FAT. The disk cache employs part of main memory to hold disk information that is needed regularly, to save having to read it from the disk each time (which is very slow compared to the memory). When the FAT is small, like the 128 KB FAT used for FAT16, the entire FAT can be held in memory easily, and any time we need to look up something in the FAT it is there at our "fingertips". When the table increases in size to 8 MB for example, the system is forced to choose between two unsavory alternatives: either use up a large amount of memory to hold the FAT, or don't hold it in memory.

For this reason, it is important to limit the size of the file allocation table to a reasonably-sized number. In fact, in most cases it is a matter of finding a balance between cluster size and FAT size. A good illustration of this is the cluster size selections made by FAT32 itself. Since FAT32 can handle around 268 million maximum clusters, the 4 KB cluster size is technically able to support a disk volume 1 TB (1,024 GB) in size. The small problem here is that the FAT size would be over 1 GB! (268 million times 4 bytes per entry).

For this reason, FAT32 only uses 4 KB clusters for volumes up to 8 GB in size, and then quickly "up shifts" to larger clusters, as this table shows:

Cluster Size	**"Minimum" Partition Size (binary MB)**	**"Maximum" Partition Size (binary MB)**
4 KB	0.5 GB	8 GB
8 KB	8 GB	16 GB
16 KB	16 GB	32 GB
32 KB	32 GB	64 GB?

Note: I don't know what Microsoft plans to do at 64 GB. They could decide to go to 64 KB clusters, or just leave the clusters at 32 KB and let the FAT size increase linearly as the disk size increases from there.

As you can see, despite the claims that FAT32 would solve large cluster problems for a long time, it really doesn't. As soon as you hit 32 GB partitions, you are back to the dreaded 32 KB we all knew and hated in FAT16. Obviously 32 GB is a lot better than having this happen at 1 GB, of course, but still, FAT32 is more of a temporary solution than a permanent one. If you're saying to yourself "Yeah, but 32 GB is *huge*. I'll never have a disk that large and even if I do, I won't care about a little wasted disk space", then remember that people said the same thing about 2 GB hard disks only a few years ago, and some people today now call 2 GB disks "small"!

The table below is a little exercise I did for fun, to show the size of the FAT (in MB) as the partition size increases, for various cluster sizes. You can see why FAT32 doesn't stick with 4 KB clusters for very long--if it did, you'd need enormous amounts of memory just to hold the table. The entries in bold show what FAT32 will choose for a partition of the given size; by going up in cluster size Microsoft keeps the size of the FAT to no more than about 8 MB:

Partition Size	**4 KB Clusters**	**8 KB Clusters**	**16 KB Clusters**	**32 KB Clusters**
8 GB	**8 MB**	4 MB	2 MB	1 MB
16 GB	16 MB	**8 MB**	4 MB	2 MB
32 GB	32 MB	16 MB	**8 MB**	4 MB
64 GB	64 MB	32 MB	16 MB	**8 MB**
2 TB (2,048 GB)	--	*1,024 MB*	*512 MB*	*256 MB*

The last entry, 2 TB, is for fun, to show how laughable I find it when people go on about 2 TB hard disks being supported by FAT32. Technically they are, I guess, if you want to deal with a FAT that is 256 MB in size and go back to having 40% of your disk wasted by slack. We had better hope our system memory goes up by a factor of 1,000 before our hard disks do again. (Really, what this all shows is the FAT file system is stretched beyond its limits even with FAT32. To get both good performance and disk space efficiency for very large volumes requires a clean break with the past and the use of a high performance file system like NTFS.)

10.5 Using Partitioning with Hard Disks Over 2 GB

There is one aspect to partitioning that is pretty clear-cut: if you have a hard disk that is over 2 GB in size, and you are not using FAT32, you must divide the disk into partitions such that each is no larger than 2 GB, in order to access the entire disk. If you do not, you will not be able to access more than the first 2 GB on the disk. (You will in most cases want to use partitions smaller than 2 GB in size anyway, to reduce slack).

10.6 Using Partitioning to Reduce Slack

Since slack is dependent on the cluster size used for the partition, and the **cluster size is directly linked to partition size**, it is possible to dramatically improve the efficient utilization of the hard disk simply by dividing it into multiple partitions. The larger the current partitions are, and the more files on the disk, the greater the opportunity for improvement. (This applies only to systems using FAT16, since FAT32 supports disks up to 8 GB using 4 KB clusters).

Let's take the example of a 2 GB disk (usually called a 2.1 GB disk by hard disk manufacturers, since they talk in decimal gigabytes). Let's say that we have 24,000 files on our disk and each has an average amount of slack amounting to 60% of a cluster. Now consider various partitioning alternatives; we can either keep the disk in one piece or break it into smaller pieces. Here is the impact on (approximate) slack space:

Cluster Size	Size of Each Partition	Number of Partitions	Typical Total Slack (All Partitions)
2 KB	128 MB	16	28 MB
4 KB	256 MB	8	56 MB
8 KB	512 MB	4	112 MB
16 KB	1 GB	2	225 MB
32 KB	2 GB	1	450 MB

As you can see, putting a 2 GB disk in a single partition is madness; typically 20% or more of the disk is going to be wasted, and you can cut that basically in half just by going to a pair of partitions. You can cut it much further by going to more partitions. In fact, the best solution might seem to be just going to 128 MB partitions, which drops slack down to a very small number. There's only one problem with this: you have to use 16 partitions! Do you really want your files spread over 16 disk volumes, from C: to R:? Most people don't. (I cringe at the very thought. 🙂)

The table shows that there is a tradeoff between saving slack and causing your disk to be broken into a large number of small pieces. Which option makes the most sense for you depends entirely on what you are doing with your disk, and on your personal preferences. I cannot stand having my disk chopped into zillions of pieces; I usually use 8 KB or 16 KB cluster-size partitions and sacrifice some more storage for the sake of what I consider "order". Others prefer the use of more, smaller partitions. You should also bear in mind the space tradeoff in using multiple partitions, something the "partitioning fanatics" (my pet name for people that like to chop a 1 GB disk into eight 128 MB partitions) often don't realize.

Note: You aren't required to make partitions that are exactly 256 MB, 512 MB, or whatever; they can be any size as long as they don't exceed the maximum for a given cluster size. So you could divide a 2 GB disk into five 400 MB partitions for example, but each of these would use the same cluster size that the 500 MB disk does in my example above.

10.7 Partition Size Tradeoff: Slack Waste and "End of Volume" Space Waste

One sensible way to combat the large amount of wasted slack space that results from the use of large cluster sizes, is to split larger hard disks into multiple smaller partitions. Unless FAT32 is being used, this is necessary to keep cluster sizes to a reasonable level and ensure reasonably efficient utilization of the hard disk.

Unfortunately, there are some people who don't understand the concept called "too much of a good thing". They tend to go overboard and chop their hard disks into ridiculous numbers of tiny partitions, thinking that they are maximizing their use of disk space this way. The ironic thing is that, in addition to making life confusing for themselves (was that file on H:? Or was it K:?) they end up not saving nearly as much space as they thought they would. The reason is that the smaller a disk volume is, the larger a percentage of it has to be left empty in order to avoid the possibility of running out of disk space. Running out of disk space can lead to data loss, and letting a hard disk get close to the point where it is running out of space can result in increased fragmentation and performance degradation if you are doing a lot of work on the disk. I call space that is reserved to ensure that volumes don't run out of space *end of volume space*.

Generally speaking, most people have a "comfort zone" regarding how little disk space they feel comfortable with having on a disk. If the amount of free space gets below this amount, they will tend to want to find something to delete, or if they are able to, get more storage. The problem is that if you have oodles of tiny partitions, it is very easy to run out of space on one while having another half empty.

Let's suppose that our comfort factor for free space at the end of a volume is 20 MB. (For me, this is way too low. I get nervous if a regular volume ever gets below about 50 MB.) Now, let's re-examine the same 2 GB with 24,000 files that we looked at in **the discussion of partitioning**, only this time also looking at the end of volume space, assuming 12 MB per partition for this:

Cluster Size	Size of Each Partition	Number of Partitions	Typical Total Slack (All Partitions)	Total End of Volume Space	Sum of Slack and End of Volume Space
2 KB	128 MB	16	28 MB	320 MB	336 MB
4 KB	256 MB	8	56 MB	160 MB	216 MB
8 KB	512 MB	4	112 MB	80 MB	192 MB
16 KB	1 GB	2	225 MB	40 MB	265 MB
32 KB	2 GB	1	450 MB	20 MB	470 MB

The answer isn't so clear-cut now, is it? I mean, the 32 KB cluster size option is still ridiculous, but really, so is the 2 KB cluster size option. This is one reason why I personally believe in not using excessive numbers of partitions. The other is simply that I get tired of always trying to figure out where my stuff is when I have to look through 8 or 10 partitions for it. I think fewer partitions keep the disk **better organized** than more partitions do.

10.8 Do More Partitions Keep the Disk "Organized"?

One argument that I commonly hear for over-partitioning is that using large numbers of partitions helps to "keep the disk organized". For example, some people say "I'd rather have eight 256 MB partitions so I can put my code on one, applications on another, games on a third, and so on, and keep everything separated". Seems to make sense, except it really doesn't, in my opinion.

The reason why is simple: anything you can organize with separate partitions and drive letters, you can better organize with top-level directory names. Contrast the two schemes outlined in the table below:

File Type	Multiple Partition Scheme	Single Partition Scheme
System Utilities	C:	C:\UTIL
Office Applications	D:	C:\OFFICE
Games	E:	C:\GAMES
Customer Data	F:	C:\DATA
Images	G:	C:\IMAGES
Sound Files	H:	C:\SOUNDS

Anything you can do with separate letters, you can do just by using the directory structure. In fact, isn't "C:\IMAGES" a lot more descriptive than "G:", which has no inherently different meaning than "H:" or any other letter? (Well, I guess "I:" would work, but that doesn't help much for your sound files.)

And the funny thing is, this isn't even the best reason to avoid using many partitions. Neither is the **reduced end-of-volume space**, though that is a factor too. The best reason is *flexibility*. If you have your 2 GB disk in eight 256 MB partitions, each devoted to a specific purpose, what do you do when, say, your games partition fills up? If you're like most people, you find the partition that is the most empty, and put your "overflow" games into it, say your sound files partition. Then say you start doing a lot of sound editing; you may then put some sound files into the images partition. The end result of all of this is that your tidy system is totally screwed up and you will have a hard time finding anything, because you won't know which games are in which partition, etc. I know that this happens because I've had it happen myself.

So, from an organizational and flexibility standpoint, I think you are generally better off with a single large partition. I think the only reason to use multiple partitions is for performance reasons (slack space reduction, primarily). This is why I generally use only a few partitions, even if I have to give up a bit of slack space as a result.

10.9 Special-Purpose Partitions and Other Partitioning Issues

There are some circumstances in which you will want to set up partitions that are smaller in size than usual, or where you might want to dedicate a partition to a specific use, or ensure that it occupies a particular place on the disk. Here are some of the extra issues you may want to take into account when considering how to partition your disk(s):

- **Partition Placement:** Most hard disks use zoned bit recording, which means they hold more data per track at the outermost edge of the disk than they do at the innermost edge; as a result, the outer tracks tend to deliver better performance than the inner tracks do. Since the outer tracks are used first, this means that the first partition on a physical disk volume will be slightly faster than subsequent ones. If you have certain files that require higher performance than others, placing them in a partition at the beginning of the disk is preferred.
- **Dedicated Partitions:** Notwithstanding my long argument against splitting the disk into many partitions where each is for one type of file, there are special situations where it may make sense to dedicate a partition to one use. The most common case where a partition is dedicated to a specific use is for the virtual memory swap file for a multitasking operating system. This file is very important since it is used often for certain types of heavy processing, and being able to control the exact properties and location of the partition that it uses can be advantageous.
- **Cluster Sizes for Special-Purpose Partitions:** Again, having specific partitions for certain types of files can cost you flexibility, but in some cases it can make sense. If you are doing a lot of work with large multimedia files, you may want to intentionally bump up the cluster size to a larger value. The main reason for this is that there is less overhead when using larger clusters--doing a sequential read of a 10 MB file on a volume that uses 32 KB clusters means 319 "next cluster" lookups in the FAT. Reading this entire file on a volume with 2 KB clusters increases this to 5,119 lookups. Another reason is that since every cluster is a contiguous block on the disk, having a larger cluster size means a greater percentage of the file is in continuous blocks--less fragmentation. This means better performance for long sequential accesses (but frequent Defragmentation of a disk with smaller clusters will mitigate this effect).

The three points above mean that the ideal place in many cases for the swap file under Windows 95, for example, is in a dedicated partition at the start of the second disk in a two-disk system, and this is what I use in my own personal PC. I have two disks that are approximately the same size and speed, and the swap file takes up the entire contents of the first partition of my second hard disk, which is about 63 MB in size. I used Partition Magic to set this partition's cluster size to 32 KB, even though a partition of this size would normally only use 2 KB clusters.

10.10 Drive Letter Assignment and Choosing Primary vs. Logical Partitions

Most people know that disk volumes using the FAT file system under DOS and its derivative operating systems are accessed using drive letters, such as C:, D: and so on. However, it can be a bit confusing figuring out how these letters are determined. Of course, as you use more disks (and other devices) and more partitions, this can get even more difficult to understand. A common confusion encountered by someone upgrading their PC is adding a hard disk to it and finding that suddenly some their files have moved from D:, say, to E:. This happens because DOS assigns drive letters using a specific sequence when the PC is booted. Drive letters are not permanently assigned to the drive, and so adding new hard disk volumes can interrupt the previous order.

Drive letters A: and B: are reserved for floppy disks. The first disk, as determined by the physical and/or BIOS configuration of the drives is A:, and the second disk is B:. Even if there is only one floppy disk in the system, B: is still reserved for use by floppies and cannot be used for any other purpose.

Hard disks are lettered starting from C:, with each partition getting a separate letter. It is essential to realize that DOS will first assign a letter to all primary DOS partitions on all hard disks in the system, before it will assign any letters to any logical DOS volumes (in extended DOS partitions) on any hard disk. This is the primary reason why adding a hard disk to an existing system can cause drive letters to shift.

Let's take an example system that contains a single 1.2 GB hard disk broken into three partitions. The first partition is the DOS primary, and then an extended partition is defined containing the other two partitions as logical volumes. The primary partition will be C:, and the other two D: and E:. Now let's say we add a second hard disk that has two partitions itself, one primary and the other logical (extended). When the system is booted up with the new hard disk in it, the first hard disk's primary partition will still be C:, but the primary partition in the second hard disk will grab D:. Then the extended partitions will be assigned letters. The result is that the logical partitions on the first hard disk will have their letters shift:

Partition	Before	After
Hard Disk 1: Primary Partition	C:	C:
Hard Disk 1: Logical Partitions	D:, E:	E:, F:
Hard Disk 2: Primary Partition	--	D:
Hard Disk 2: Logical Partitions	--	G:

Having drive letters change is not only confusing, it can cause programs to malfunction, because many programs record what disk and directory they are in and expect that they will remain stationary (which stinks, but hey, that's life.)

There is one relatively simple way to avoid having this happen: don't create a primary partition on any hard disks in the system other than the first one. It is perfectly legal to only create an extended partition on a hard disk, and put all of the partitions in it. The only place that a primary partition is absolutely needed is on the first hard disk, because it is required to boot the operating system. You cannot normally boot from an extended partition volume anyway (although some motherboards will apparently let you).

Note: One other minor consideration is that you lose a small amount of disk space if you create only extended partitions on a disk. The extended partition will begin using (logical) cylinder 1 of the disk, and therefore you lose the space in cylinder 0. This will generally cost you a handful of megabytes of storage, but is pretty much unavoidable if you want your partitions in a sensible order.

So in the example above, if our second hard disk had its two partitions both defined as logicals, then they would have been assigned F: and G:, and the partitions on the first disk would have been unchanged. This is in fact the way you will normally want to partition second and third hard disks in a system. What if you are adding a hard disk from another system that has already been set up with a primary partition? Unless you don't care about the second hard disk's primary partition being assigned D:, you have to delete the primary partition (after copying any data from it of course!) and expand the size of the extended partition to take its place. For a job of this sort, a utility like **Partition Magic** is indispensable. Otherwise you will basically have to wipe the second disk clean and start over.

Once all of the hard disk partitions have been assigned drive letters, the system will allocate letters for other devices that are operated using drivers. Devices like CD-ROMs, Zip drives, etc. are operated through software drivers that present the device to the operating system as if they were a disk, and then they are assigned a drive letter so they can be accessed. Normally, these devices are assigned a letter immediately after the last one used by hard disks, but this can be changed by using different parameters for the driver software. For example, the drive letter for a CD-ROM drive under Windows 95 can be set in software using the Device Manager.

Finally, some software programs can change drive letters. A very common example is the use of **disk compression** agents such as DriveSpace. These will often take a hard disk, say D:, change its letter to something out of the way like R:, create a compressed volume on R:, and then map the compressed volume back to D:. The net result is that the disk is compressed while appearing to still be uncompressed.

11 Disk Partitioning and Formatting Programs

Virtually every operating system today ships with a complete set of utilities for partitioning and formatting disks, both hard and floppy. The tools used for operating systems that use the FAT file system have not changed much in the last 10 or more years; if you are using DOS, Windows 3.x or Windows 95, you will still use the same set of DOS utilities. A complete coverage of DOS commands is beyond the scope of this site, but I wanted to include this section that make brief mention of the primary utilities (both DOS and third-party) that are used for setting up disks.

Warning: All hard disk partitioning and formatting utilities work with the disk at an intimate level. There is always a chance of data loss when using these programs. As always, performing a complete backup before changing your system is prudent. If you are careful, in most cases you will have no problems, but especially if you are attempting to change existing disk structures that contain data, there is a slight risk of problems. In particular, loss of power during sensitive disk operations can leave a disk volume in an unusable state. Using a UPS is prudent; at the very least, don't do the obviously silly (say, trying to resize all your disk's partitions during a big thunderstorm...)

- **FDISK**
- **FORMAT**
- **SYS**
- **Partition Magic**

11.1 FDISK

The program that DOS supplies for setting up hard disk partitions is called *FDISK*, which I believe stands for "fixed disk", an older term for hard disk. FDISK is used only for DOS (FAT) partitioning, and allows you to perform the following functions:

- **Create Partitions:** FDISK allows you to create a primary DOS partition or logical DOS volumes. To create a logical DOS volume you must of course first create an extended DOS partition, since the logicals are contained within the extended partition.
- **Set Active Partition:** You can use FDISK to set the primary partition on your boot disk active, so that it can boot. It's quite silly that FDISK doesn't do this automatically when you create the boot primary partition (since there can only be one primary DOS partition anyway), but in fact you must do this manually in many cases. (At least FDISK warns you when no disk is set active, via a message at the bottom of the screen.)

? **Delete Partitions:** FDISK will let you delete partitions as well. This is the only way to change the size of a partition in FDISK: delete the old one and create a new one with the new size. If you want to change the size of the primary DOS partition using FDISK you must delete every FAT partition on the disk and start over...
? **Display Partition Information:** The last option that FDISK gives is to display the partition information for the system. It will first show the primary and extended partitions and then ask you if you want to see the logical drives within the extended partition. In fact, if you want to see this information, you can just do "FDISK /STATUS" from the DOS command line. This will show you the partition information without actually taking you into FDISK, and therefore, you run no risk of accidentally doing something you'll wish you hadn't.

Some important points that you should keep in mind when using FDISK:

? **Be Careful:** With just a few keystrokes, FDISK can wipe out part or all of your hard disk. Generally speaking, don't use FDISK unless you need to, and make sure you understand what you are doing before you begin.
? **Run It From DOS:** Windows 95 allows you to run FDISK direct from the graphical user interface, and even while other applications are open and running. Since FDISK alters critical disk structures at a very low level, running it while files are open and other applications are using the disk is asking for trouble. To be safe, always exit to DOS ("Restart the computer in MS-DOS mode") before using FDISK (except for using "FDISK /STATUS", will work safely from within a DOS box in Windows 95).
? **FAT32 Support:** The version of FDISK that comes with Windows 95 OEM SR2 supports the creation of partitions that use the FAT32 enhanced file system for larger volumes. Some clever person at Microsoft decided not to call it FAT32 however within this program. Instead, when you run FDISK on a system that has Windows 95 OEM SR2 installed, and a hard disk over 512 MB (the minimum for using FAT32), you will receive a message asking you if you want to "enable large disk support". If you answer "Y" then any new partitions created in that session will be FAT32 partitions.

Note: It is often useful to include FDISK as one of the programs on a bootable floppy. This way you can use it when setting up new hard disks.

Considering how important it is, FDISK is a rather primitive program. It works, but it's cryptic and hard to use. Anything you can do in FDISK you can do more flexibly and easily using a third-party program like Partition Magic. FDISK will not allow you to select or change cluster sizes, resize partitions, move partitions, etc. F-Disk’s primary advantage is, of course, that it is free (well, built-in anyway).

Windows NT uses a program called *Disk Administrator* to handle disk setup tasks. In essence, this is an enhanced version of FDISK that allows you not only to manipulate partitions, but also access some of NT's unique disk management features. For example, you can set up software **RAID** using the Disk Administrator.

11.2 FORMAT

The well-known FORMAT command is used, of course, to format hard disks and floppy disks. Many people don't realize that this command functions quite differently for hard disks and floppy disks. There are **two steps to formatting**: low-level formatting and high-level formatting. For floppy disks, FORMAT does both low-level formatting and high-level formatting. For hard disks, it only does high-level formatting, because modern hard disks are low-level formatted at the factory.

There are many different parameters that the FORMAT command will use; they can be seen by typing "FORMAT /?" at a DOS command line. Most of these commands are used for specifying different formatting options for different types of floppy disks. An important parameter is the "/S" flag, which tells FORMAT to make the volume it is formatting bootable. This is done by creating the proper disk structures for booting, and copying the operating system files to the root directory of the new volume.

11.3 SYS

Normally the operating system files that allow a hard disk to be booted are placed at the front of the disk at the time that the boot volume is high-level formatted, using the **FORMAT command** (with the /S parameter). It is also possible, however, to "convert" an existing disk so that it is bootable, by using the *SYS* command. SYS copies the operating system files from the volume that the system was booted with (hard disk or floppy) to the target volume.

SYS is normally used to create bootable floppy disks from non-bootable disks. It is also sometimes used to upgrade DOS versions on hard disks; you boot the floppy of the new DOS version and then SYS the new operating system files to the hard disk. (Newer operating systems like Windows 95 take care of this sort of work internally, and don't require this sort of manual operation).

11.4 PARTIOTION MAGIC

Partition Magic is a disk manipulation utility from PowerQuest that no serious PC buff should be without. In a nutshell, Partition Magic does all the things that FDISK *should* let you do, but doesn't, plus a lot more. This includes dynamic resizing of partitions without loss of data, changing cluster sizes, copying partitions from one disk to another, and much, much more. There's not much point in my listing all of Partition Magic's features on my site when ***PowerQuest has its own pages describing the product in detail***. To be fair, ***Quarterdeck has a utility that is similar to Partition Magic, called Partition-It***. I haven't used it so I can't say much about it.

The latest version of Partition Magic, 3.x, supports the FAT32 file system and is therefore greatly recommended over the older, 2.x versions. Since FAT32 was only made available in Windows OEM SR2, intended for installation on new machines by OEMs, Microsoft did not include any way to convert existing FAT16 partitions to FAT32. Partition Magic will let you do this if you use version 3.x.

Warning: A final caveat about Partition Magic--some of its operations can take a fair bit of time, such as resizing partitions or changing cluster sizes. During the time that this work is taking place, your hard disk is vulnerable to data loss in the event of a hardware or power failure. It is highly recommended that you back up your data before working on your partitions; when I am doing this sort of work, I try to make sure the PC is plugged into a UPS, just in case. This is also true of using FDISK and in fact any software that works intimately with the hard disk, but since FDISK is normally used when the disk is empty, there isn't the same concern about data loss.

12 DISK COMPRESSION

While it is decreasing in importance due to the very rapid drop in the cost per gigabyte of hard disk storage, there was a time when many PCs used one form or another of disk compression, due to the relatively small disks and their high cost at the time. We're only a few years removed from most PCs having only 40-100 MB hard disks. Compression allows you to store more information in the same amount of disk space, by using special software that reorganizes the way information is stored on the disk. (Hardware compression does exist also but is not generally used for hard disk volumes.)

? **Why Disk Compression Works**
? **Compression Types**

? **Volume Compression Operation**
? **Volume Compression Products**
? **Free Space and the Estimated and Actual Compression Ratios**
? **Slack Reduction Using Volume Compression**
? **Compression Level and Performance Considerations**
? **Memory Issues with Compression Drivers**
? **Compatibility and Reliability Issues in Volume Compression**

12.1 Why Disk compression works?

Disk compression takes advantage of (at least) two characteristics of most files. First is the fact that most files have a large amount of redundant information, with patterns that tend to repeat. By using "placeholders" that are smaller than the pattern they represent, the size of the file can be reduced.

For example, let's take the sentence "In fact, there are many theories to explain the origin of man.". If you look closely, you will see that the string " the" (space plus "the") appears in this sentence three times. Compression software can replace this string with a token, for example "#", and store the phrase as "In fact,#re are many#ories to explain# origin of man.". Then, they reverse-translate the "#" back to " the" when the file is read back. Further, they can replace the string " man" with "$" and reduce the sentence to "In fact,#re are$y#ories to explain# origin of$.". Just replacing those two patterns reduces the size of the sentence by 24%, and this is just a simple example of what full compression algorithms can do, working with a large number of patterns and rules.

The other characteristic of many files that disk compression makes use of is the fact that while each character in a file takes up one byte, most characters don't require the full byte to store them. Each byte can hold one of 256 different values, but if you have a text file, there will be very long sequences containing only letters, numbers, and punctuation. Compression agents use special formulas to pack information like text so that it makes full use of the 256 values that each byte can hold.

The combination of these two effects results in text files often being compressed by a factor of 2 to 1 or even 3 to 1. Data files can often be compressed even more: take a look at some spreadsheet or database files and you will find long sequences of blanks and zeros, sometimes hundreds or thousands in a row. These files can often be compressed 5 to 1, 10 to 1 or even more.

Finally, compression is also useful in battling **slack**. If you have 1,000 files on a hard disk that uses 16,384 byte clusters, and each of these files is 500 bytes in size, you are using 16 MB of disk space to store less than 500 KB of data. The reason is that each file must be allocated a full cluster, and only 500 of the 16,384 bytes actually has any data--the rest is slack (97%!) If you put all of those files into a compressed file like a ZIP file, not only will they probably be reduced in size greatly, but the ZIP file will have a maximum of 16,383 bytes of slack by itself, resulting in a large amount of saved disk space. The advanced features of DriveSpace 3 volume compression will in fact reduce slack even if file compression isn't enabled.

12.2 Compression Types

There are several different ways that files can be compressed on the hard disk. I am referring here to the logical mechanisms for performing the compression and decompression, and not the compression algorithm itself. There are in fact many different compression algorithms, but the details of how the compression is actually done are hidden entirely from the user.

There are three main ways that compression is implemented on PCs:

? **Utility-Based File Compression:** A very popular form of disk compression, used by virtually every PC user whether they realize it or not, is file-by-file compression using a compression utility. With this type of compression, a specific utility is used to compress one or more files into a compressed file (often called an *archive*), and another similar utility is used to decompress the compressed file. The operating system knows nothing about the compression; to it the compressed file is, just another file. In order to use the compressed file at all, it must be decompressed. The most popular compression system of this sort is the wildly popular PKZIP package, and its derivatives such as WinZip. Virtually all software or large files that you download from the Internet for example, uses some form of this compression.

? **Operating System File Compression:** While not supported by the FAT file system used by DOS and most versions of Windows, some operating systems support the compression of files on an individual basis within the operating system itself. For example, Windows NT supports this feature when you use the NTFS file system. This is in many ways the best type of compression, because it is both automatic (decompression is done by the operating system when the file is needed by any program) and it allows full control over which types of files are compressed.

? **Volume Compression:** Distinctly different than compressing individual files, it is also possible with most newer operating systems to create entire disk volumes that are compressed. This can be done either through utilities provided with the operating system, or third-party packages. Volume compression allows you to save disk space without having to individually compress files. Every file that is copied to the compressed volume is automatically compressed, and each file is automatically decompressed when any software program needs it. Volume compression is transparent to the use and generally works well on most PCs.

Of these types of compression, utility-based file compression is common but relatively straight-forward; you use a program to create a compressed file and another to look at it. From the operating system's perspective, the compressed files and the utilities that use it are just like any other files and programs on the disk, no different than say, a word processor and a word processing document file. Volume compression, on the other hand, is commonly used and has more complicating factors involved in its usage. In particular, there are **performance considerations** and **safety and compatibility issues** that need to be carefully weighed before using volume compression.

12.3 Volume Compression Operation

Disk volume compression works by setting up a *virtual volume*. In essence, a software-driven volume is created on the system and special drivers used to make this volume appear to be a physical hard disk. This isn't really that radical a concept, since many devices use software drivers to allow them to appear to the system under a drive letter.

When you decide to create a compressed volume on your disk, here is what the software that is creating it actually does, in approximate terms:

1. The software will ask you which real disk partition you want to use to hold the compressed volume. This is sometimes called the *host volume* or *host partition*. It will also ask you whether you want to compress the existing data on that volume (if any), or instead use the current empty space on the volume to create a new compressed volume from.
2. The target disk volume is prepared for compression by scanning it for logical file structure errors such as lost clusters and also for errors reading the sectors on the disk. If the disk is highly fragmented, it may need to be defragmented as well, since the compressed volume must be in a contiguous block on the disk.
3. A special file on the hard disk is created, called a *compressed volume file* or *CVF*. This file is what contains the compressed volume. If you are creating a compressed volume

from empty space, the CVF is written directly onto the hard disk and prepared with the correct internal structures for operation. If you are creating a compressed volume from an existing disk with files on it, the software may not have enough free space to create the full CVF. It will instead create a smaller one, move some files into it from the disk being compressed, and then use the space that these files were using to increase the size of the CVF to hold more files. This continues until the full disk is compressed. This operation can take a very long time.

4. The CVF is hidden from view using special **file attributes**. Special drivers are installed that will make the CVF appear as a new logical disk volume the next time the system is rebooted. This is sometimes called "mounting" the CVF, in analogy to the physical act of mounting a physical disk.

When you are using compression, then, what you see as a compressed volume is really just a giant file on a real hard disk. In some cases, you will be able to use both disks. For example, when I set up an older system with, say, a 340 MB hard disk, I will often split the disk by creating a compressed drive called D: from say, 150 MB of space from C:. So then C: will have 190 MB free and a 150 MB compressed volume file. The compression drivers will create a logical D: volume from the 150 MB CVF.

Another option that the compression software usually has the ability to provide is to "substitute" the compressed volume in place of the host volume it is made from. This is normally done if you are creating a CVF that takes up an entire disk partition. Let's suppose you have a 540 MB hard disk that is partitioned into a 300 MB C: and a 240 MB D:, and you want to compress the entire D:. What the software will normally do after it creates the CVF taking up all of D: is to "map" the host D: drive to a much higher-up letter like H:, and then make the CVF appear as D: in its place. This allows the seamless compression of a hard disk while retaining its previous letter address.

Warning: I don't recommend doing this with the C: boot partition, even though Microsoft's DOS DriveSpace program sometimes recommends this by default. In my opinion it is better to create a separate compressed volume and leave the boot volume C: alone, so that the system can be booted more easily in the event of a problem.

Warning: If you delete the compressed volume file from the host drive, guess what happens to your compressed volume? Poof. Don't do it.

12.4 Volume Compression Products

There are two main products on the market that are popularly used for volume compression in DOS and Windows. The first is DriveSpace (formerly called DoubleSpace), which is a Microsoft product that comes in various versions and *with* various versions of DOS and Windows. The second is Stacker, which is a product of ***Stac Electronics***. I don't plan to get into a lengthy review of the two products because that is not the purpose of this part of the site. Both do a good job of providing compressed volumes, and each has some advantages and disadvantages.

I personally prefer the use of Microsoft's DriveSpace 3 product, for Windows 95, or the older DriveSpace 2 product for MS-DOS 6.22 (which is what you use if you are running Windows 3.x). This is not to disparage Stacker; in fact I have read that Stacker is a very good product and in many ways better than DriveSpace. I just feel more *comfortable* with DriveSpace because I have used it more, and because due to the **dangers of using compression**, I prefer having the compression software and operating system vendor be the same. I know that I will always be able to rely on the compression being supported fully by the operating system since they are made and tested by the same company (well, that's the theory anyway.)

It should be noted that the older DriveSpace 2 product is limited in its functionality compared to DriveSpace 3 and the latest version of Stacker. It only supports compressed volumes up to 512 MB, and that's the *compressed* volume size. If you want to compress a 1 GB host drive you have to split it into three or four compressed volumes. It also offers far fewer options to let you tailor how the compression is managed on the drive, and suffers from lower performance as well. DriveSpace 3 is a far superior product, but is supported only for Windows 95.

Note: DriveSpace 3 is not supported for use on FAT32 disk volumes.

12.5 Free Space and the Estimated and Actual Compression Ratios

One source of confusion in the use of compressed disk volumes relates to the amount of free space reported on the drive. This can--and will--change in sometimes unexpected ways as the compressed disk fills up, based on how compressible the files placed on the volume are. The reason is simple: the free space on a real disk volume is easily determined simply by examining how many clusters are free in the file allocation table, and multiplying by the cluster size. With a compressed volume, we don't know how much space is free until we know what will be put on the volume, because some files can be compressed a great deal, and some not at all.

In fact, when you see a report of disk space free on a compressed volume, what you are seeing is an *estimate*. Every compressed volume has a number associated with it called its *estimated compression ratio*, which is what tells the compressed volume driver how well you think the files on this volume are going to be compressed. This number can be set on a volume-by-volume basis, and should be estimated, ideally, from the *actual compression ratio* that the volume is using with its current files. Because it makes their tools look impressive, many compression utilities default this estimate to something like 2:1, even though the average disk volume in my experience will not compress at a figure that high. Usually 1.6 to 1.8 is more typical (depending on the settings and of course, to a great deal on what is stored on the drive).

Using the estimated compression ratio, the system will determine an estimated amount of free space by multiplying the uncompressed free space by the estimated ratio. This is what you see as free space on the drive. If you change the estimated compression ratio, the report of free disk space changes as well; in reality, you have not changed at all the capacity of the compressed volume.

As soon as you copy files to the compressed dsk, they are compressed at whatever rate the compression software can manage for the type of file. A huge text file could be compressed at 3:1; a ZIP file--which is already compressed internally--will not compress at all. The amount of space these real files takes up will vary, and so the amount of free space will change, depending on what the files are.

OK, time for an example. Let's suppose we have an empty 500 MB hard disk that we want to compress. We run DriveSpace 3, say, and it sets up a 500 MB CVF on the host disk, and creates a new compressed drive called F: for example. When we look at F:, we see 1000 MB free. Why? Because the default is a 2:1 estimated compression ratio. This can be changed, of course. For now, let's leave it at 2:1. Here's what the disk looks like:

Storage	Used Space	Free Space	Total
Uncompressed Total (Inside the CVF)	0 MB	500 MB	500 MB
Compression Ratio	--	2:1	2:1
Compressed Total	0 MB	1000 MB	1000 MB

So we have 500 MB of "true" space on the host disk, in the CVF, and 1000 MB of space in the compressed disk, assuming our 2:1 ratio. Now suppose we copy to this empty disk a 100 MB file that cannot be compressed very much; let's suppose it can only be compressed at a ratio of 1.25 to 1. This means we will use up not 50 MB of real CVF space as we would expect from a file compressible at 2:1, but rather 80 MB (100 divided by 1.25). Here's what the disk will look like now:

Storage	Used Space	Free Space	Total
Uncompressed Total (Inside the CVF)	80 MB	420 MB	500 MB
Compression Ratio	1.25:1	2:1	1.88:1
Compressed Total	100 MB	840 MB	940 MB

As you can see, the 500 MB total for the CVF always remains the same (unless you resize the volume). But we have "lost" 60 MB from the compressed volume; it now has 840 MB free instead of the 900 MB we would expect after copying a 100 MB file to it. In reality, all that happened is that we weren't able to compress the file at the 2:1 ratio we expected.

Now let's copy another file to the compressed disk, let's say a 180 MB database file that will compress at a ratio of 4 to 1. (This is quite common with large database files, believe it or not.) This file will take only 45 MB of storage in the CVF, instead of the 90 MB that is "par" for a 2:1 volume. Here's what will happen:

Storage	Used Space	Free Space	Total
Uncompressed Total (Inside the CVF)	125 MB	375 MB	500 MB
Compression Ratio	2.24:1	2:1	2.06:1
Compressed Total	280 MB	750 MB	1030 MB

After storing this file our totals have increased, because it used much less space than we "expected". In fact, our compressed volume is now "larger" than when we started! While this is an extreme example, this shows why the free space on a compressed disk tends to move around. The most common situation is when someone sets up a 1000 MB compressed disk and starts copying huge ZIP files to it. Guess what--ZIP files are already compressed and cannot be compressed further. As soon as you copy about 500 MB of ZIP files to a compressed volume of that size, it will be full.

12.6 Slack Reduction Using Volume Compression

Slack is the space wasted due to unused storage at the end of large clusters. When a great number of files are stored on a disk with a large cluster size, a lot of the disk is wasted due to overhead. One way to eliminate this is to use a utility program to create a ZIP file archive, for example, containing all the files. But this results in the files not being readily accessible.

Volume compression tools like DriveSpace 3 can in fact save space even when set not to compress any files at all, due to slack space reduction. Internally, DriveSpace 3 compressed drives allocate files on a sector-by-sector basis. This means that they have an *effective* cluster size of 512 bytes, because they use an internal format that lets them store more than one file in what would be a cluster on a regular drive.

Some people set up DriveSpace 3 volumes just to get this advantage. It can be configured so that no files are actually compressed (for performance reasons, to save on the overhead of decompression) but you still get the slack reduction because of the slack reduction feature. You still have some of the risks associated with using a compressed volume however, and also the loss of some memory for the compressed volume driver, which has to operate even if you set it to no compression (slack reduction only).

12.7 Compression Level and Performance Considerations

Most advanced compression software will give you some control over how much compression you want to use on the volume, and even let you select certain types of files (based on their extension) not to try to compress. In general, the more you try to compress the volume, the more you can cram onto the disk, but the more advanced compression algorithms can result in performance degradation because they work harder to squeeze the files more so they can compress that little bit more. The degree of degradation depends on several factors, but is especially dependent on the speed of the system you are using. A slower processor can cause compression to result in a serious performance hit, while a faster processor can in some cases make compression *improve* performance.

Improve performance through compression? How is that possible? Let's consider two copies of an identical 100 MB file; one copy is uncompressed and the other is compressed 2:1 on a volume on the same hard disk. Suppose we need to scan every byte of each of these files. We can read the uncompressed file at a faster rate *per byte* because we don't have the overhead of decompression. However, we have to read twice as many bytes from the disk, because the file is taking up more space on the disk. The compressed file is using only 50 MB of physical disk sectors, instead of 100 MB.

As you probably know, hard disk access is much slower than the processor, memory, chipset and other system components. For this reason, removing the need to read 50 MB of data from the disk can save more time than the time required for the overhead of decompression! If you have a fast processor that is spending a great deal of time waiting for data from the slow hard disk then you actually get a performance boost with some types of files by using compression.

The important factor is how fast the system is relative to the hard disk. If the processor is fast and the hard disk slow, you will see this effect. If the processor is slow and the hard disk fast, compression will cause a noticeable slowdown. This is why I don't recommend compression on 486-class machines (unless hard disk space is a *severe* problem), while I use it myself for two of my partitions on my Pentium-class machine. I usually notice no slowdown in using these volumes.

There is "middle ground" as well in terms of the compression level. You can usually choose how much compression to use on the volume, ranging from high compression to none at all. Sometimes volumes are used with no compression just to get the benefits of slack reduction that compression can provide.

Finally, there are some types of files that just don't belong on compressed disks. If you have a bunch of large ZIP files, don't put them on a compressed volume, since there is no benefit--they are already compressed and so volume compression will not help them at all. The same thing applies to most multimedia files such as JPEG or GIF images--they have already been compressed internally. Storing these files on compressed volumes will also throw off your **free space estimates** because they will take up much more space than a compressible file of the same size would.

12.8 Memory Issues with Compression Drivers

Compressed volumes only work with a compression driver, which must be loaded before the compressed disk can be used. This is only an issue for compressed volumes running under DOS or Windows 95. The problem is that this driver can be rather large, and can exacerbate conventional memory problems.

Normally, DriveSpace 2 (or DoubleSpace) is used for older DOS versions, and DriveSpace 3 for Windows 95. Under DOS, the DriveSpace 2 driver can be loaded into an upper memory block to reduce conventional memory usage, and this driver is not excessively large. Under Windows 95, the DriveSpace 3 driver *is* large, but the system provides protected mode compression drivers that run in extended memory, so there isn't a problem with conventional memory being used by the driver.

The real problem occurs when Windows 95 drops down to MS-DOS mode; here the real-mode DriveSpace driver must be loaded to conventional memory and it is quite large. This problem is a valid one and cannot be eliminated. Good conventional memory management can reduce the problem, but cannot remove it completely.

12.9 Compatibility and Reliability Issues in Volume Compression

There are several reasons why compression isn't nearly as popular now as it was in the past. One is simply the lack of necessity; with fast hard disks now less than $100 per gigabyte and prices falling fast, there is much less of a need to "stretch" the hard disk using compression. Another is concerns over performance. But probably the biggest one is concerns over the reliability and safety of disk compression.

Some of these concerns are valid, but in my opinion most have been overblown. Mark Twain said that a lie could make it half way around the world while the truth was still putting on its shoes. Truly a wise man, he foresaw the creation of USEnet by a full century ☺. Replace "lie" with "bad news" and "truth" with "good news" and you have the situation in a nutshell. Whoops, I digress. Must have spent too much time writing this hard disk chapter. ☺At any rate, there are some valid concerns regarding compression which are worth looking at, if only to give them some perspective.

First, regarding compatibility, in my experience you will find very few problems. In fact, the biggest problems with programs running on compressed disks have to do either with conventional memory difficulties due to the compression driver, or reduced performance due to decompression overhead, which can affect some software that needs high performance. Virtually all regular programs see a compressed volume as just another disk, and all modern utilities (written in the last few years anyway) will handle compressed volumes just fine. Most software that is not meant for running on compressed disks will tell you this in their instructions.

Some software will not work properly on a compressed volume because it cannot tolerate the potential delay in decompressing the data. This delay can vary depending on how the files are compressed and other factors. Some programs need real-time data streaming from the hard disk without interruption. A CD-R recording utility would be a good example. This sort of application (well, its data anyway) should be placed on an uncompressed volume.

There is a greater chance of a catastrophic data loss when using compressed volumes than uncompressed volumes. The reason is that there is an extra layer of software interpretation, and an extra layer of disk structures that can potentially become damaged. Your entire compressed volume is stored in a single real file, the CVF, and if that file should become damaged or accidentally deleted, you could lose some or all of the files on the compressed volume. In practice, the use of compression today is quite safe. There were in fact some problems with reliability associated with early implementations of the technology, but these have been ironed out quite well for the mostpart.

I will state that I have used compression, from DoubleSpace to DriveSpace to DriveSpace 3, on all of my PCs for over 5 years, and I have set up or maintained several dozen PCs that have used one version or another, and I have never had any problems relating to the use of compression. That said, I recognize the increased possibility of data errors resulting from compression, which is why I follow these general guidelines in how I use it:

? **Never Compress the Boot Drive:** I do not set up PCs that have the C: drive compressed. All of the sensitive disk structures and the operating system are on this drive, and I think it prudent to leave these uncompressed. This also allows the system to be bootable without the compression driver, if this ever becomes necessary, and also makes it easier to deal with virus problems, should they arise.

? **Compress Applications, Not Data:** I generally use compression for items that can be easily recreated in the event of loss, such as installed programs and especially games. I don't generally compress data partitions. My PC at home has the games partition compressed. (I use an uncompressed partition for games requiring faster performance.)

? **Scan For Problems Regularly:** The same utilities that are used to scan for trouble on regular hard disks can be used to great effect on compressed drives, and should be run regularly to minimize long-term problems.

? **Back Up Regularly:** I harp on this a lot, and with good reason. In my opinion you have little to fear from compression overall. If you back up your hard disk every week and keep your sensitive data off your compressed partitions, you have basically nothing to fear from it.

I believe that if used intelligently, compression is safe and has value, under the correct circumstances. Of course, with hard disk sizes getting into the gargantuan range and prices continuing to drop, I expect that compression will soon be a thing of the past (but may become *more* popular in older PCs with small disks, as these strain with trying to upgrade to newer and larger software).

Note: Windows NT is not compatible with traditional compressed volumes. Of course, Windows NT provides far superior file-based compression when you are using its advanced NTFS file system.