from PCGUIDE.COM

HARD-DISK BASICS

2005-Feb-12

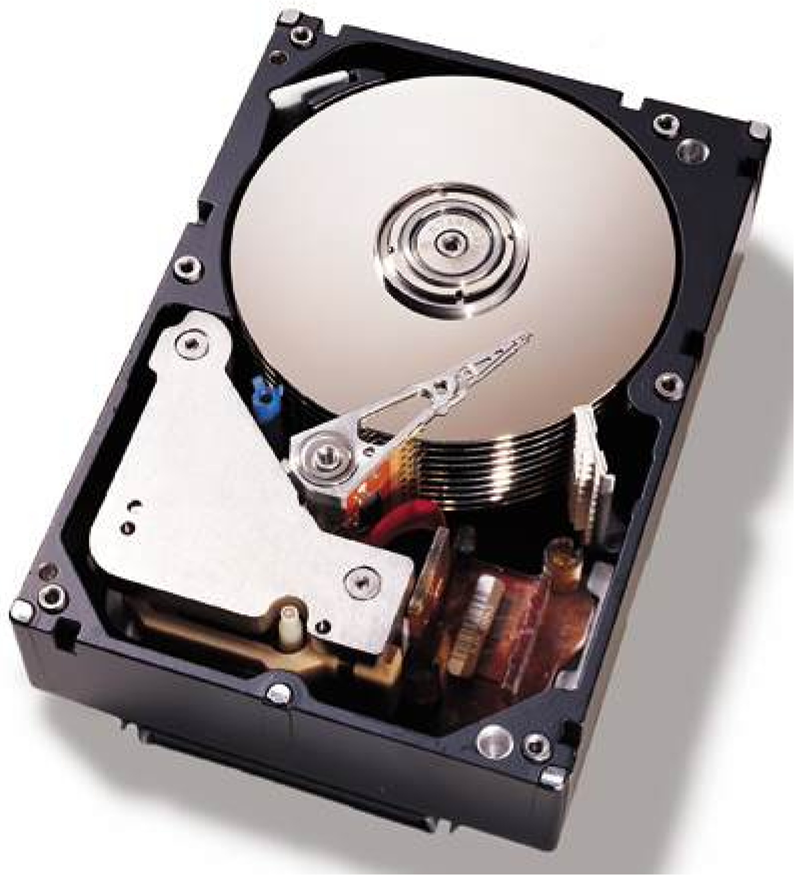
COMPILED By mehedi hasan
mehedi@joinme.com
PABNA , BANGLADESH

[HYPERLINKS HAD BEEN LOST]

Hard Disk Drives

The hard disk drive in your system is the "data center" of the PC. It is here that all of your programs and data are stored between the occasions that you use the computer. Your hard disk (or disks) are the most important of the various types of permanent storage used in PCs (the others being floppy disks and other storage media such as CD-ROMs, tapes, removable drives, etc.) The hard disk differs from the others primarily in three ways: size (usually larger), speed (usually faster) and permanence (usually fixed in the PC and not removable).

Hard disk drives are almost as amazing as microprocessors in terms of the technology they use and how much progress they have made in terms of capacity, speed, and price in the last 20 years. The first PC hard disks had a capacity of 10 megabytes and a cost of over $100 per MB. Modern hard disks have capacities approaching 100 gigabytes and a cost of less than 1 cent per MB! This represents an improvement of 1,000,000% in just under 20 years, or around 67% *cumulative* improvement per year. At the same time, the speed of the hard disk and its interfaces have increased dramatically as well.



Top view of a 36 GB, 10,000 RPM, IBM SCSI server hard disk, with its top cover removed. Note the height of the drive and the 10 stacked platters. (The IBM Ultrastar 36ZX.)

Original image © IBM Corporation
Image used with permission.

Your hard disk plays a significant role in the following important aspects of your computer system:

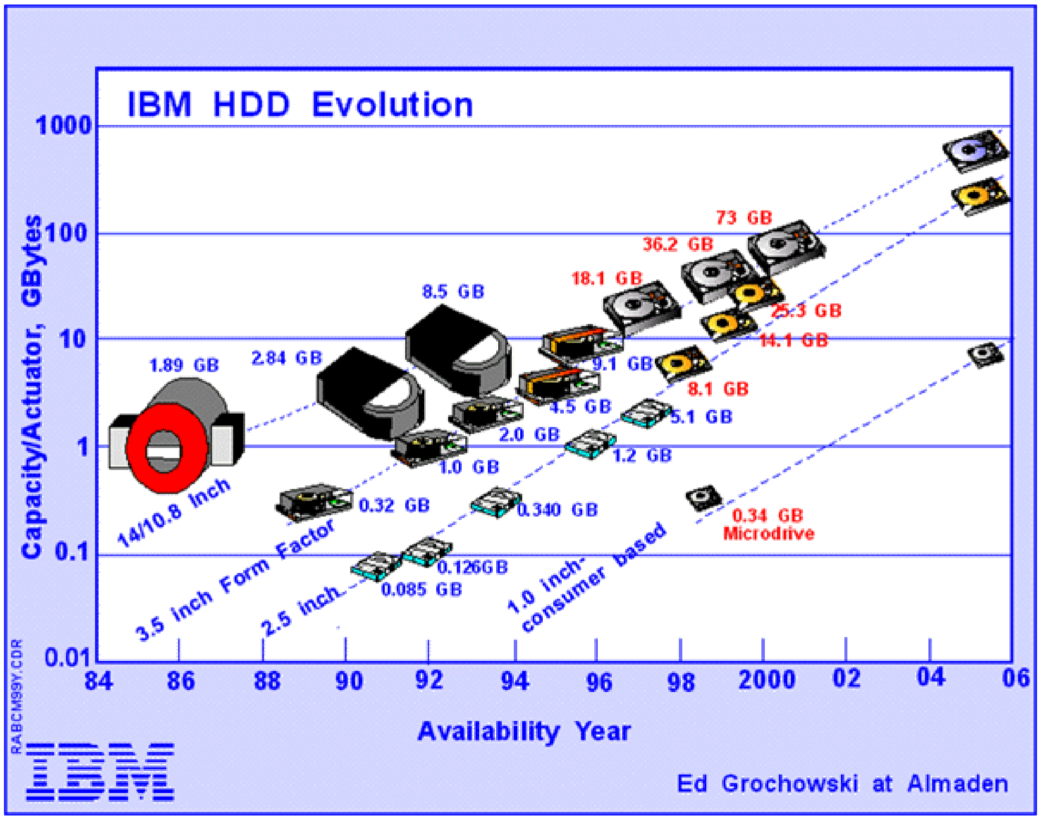
- **Performance:** The hard disk plays a very important role in overall system performance, probably more than most people recognize (though that is changing now as hard drives get more of the attention they deserve). The speed at which the PC boots up and programs load is directly related to hard disk speed. The hard disk's performance is also critical when multitasking is being used or when processing large amounts of data such as graphics work, editing sound and video, or working with databases.
- **Storage Capacity:** This is kind of obvious, but a bigger hard disk lets you store more programs and data.
- **Software Support:** Newer software needs more space and faster hard disks to load it efficiently. It's easy to remember when 1 GB was a lot of disk space; heck, it's even easy to remember when 100 MB was a lot of disk space! Now a PC with even 1 GB is considered by many to be "crippled", since it can barely hold modern (inflated) operating system files and a complement of standard business software.
- **Reliability:** One way to assess the importance of an item of hardware is to consider how much grief is caused if it fails. By this standard, the hard disk is the most important component by a long shot. As I often say, hardware can be replaced, but data cannot. A good quality hard disk, combined with smart maintenance and backup habits, can help ensure that the nightmare of data loss doesn't become part of your life.

This chapter takes a very detailed look at hard disks and how they work. This includes a full dissection of the internal components in the drive, a look at how data is formatted and stored, a discussion of performance issues, and a full analysis of the two main interfaces used to connect hard disks to the rest of the PC. A discussion is also included about the many confusing issues regarding hard disks and BIOS versions, and support for the newer and larger hard disks currently on the market. Finally, a full description is given of logical hard disk structures and the functioning of the FAT and NTFS file systems, by far the most popular currently used by PCs.

Next: A Brief History of the Hard Disk Drive

A Brief History of the Hard Disk Drive

Hard disks are one of the most important and also one of the most interesting components within the PC. They have a long and interesting history dating back to the early 1950s. Perhaps one reason that I find them so fascinating is how well engineers over the last few decades have done at improving them in every respect: reliability, capacity, speed, power usage, and more.



This excellent chart shows the evolution of IBM hard disks over the past 15 years. Several different form factors are illustrated, showing the progress that they have made over the years in terms of capacity, along with projections for the future. 50GB hard disks in laptops in five years? Based on past history, there's a good chance that it will in fact happen! Note that the scale on the left is logarithmic, not linear, and PC hard disks have one actuator.

Image © IBM Corporation
Image used with permission.

This section takes you on a brief "guided tour" of the history of hard disk drives, to help you understand more about how they were developed and what the key technological improvements were that enabled the modern disk drive that we all rely on today.

Next: Life Without Hard Disk Drives

Life Without Hard Disk Drives

It's very hard for modern computer users to even consider what "computer life" would be like with hard disk drives. After all, most of us now have *billions* and *billions* of bytes of information ready at our fingertips (apologies to Carl Sagan... ;^)). What was using a computer like before we had hard disk drives? In a word... *inconvenient*.

Some of the very earliest computers had no storage at all. Each time you wanted to run a program you would have to enter the program manually. Needless to say, this was a major pain in the butt. Even more than that, it made most of what we consider today to be computing impossible, since there was no easy to way to have a computer work with the same data over and over again. It was quickly realized that some sort of permanent storage was necessary if computers were to become truly useful tools.

The first storage medium used on computers was actually *paper*. Programs and data were recorded using holes punched into paper tape or punch cards. A special reader used a beam of light to scan the cards or tape; where a hole was found it read a "1", and where the paper blocked the sensor, a "0" (or vice-versa). This was a pretty simple arrangement. I remember using a punch station, which was like a workstation where you typed characters and the machine punched the holes into the cards. While a great improvement over *nothing*, these cards were still very inconvenient to use. You basically had to write the entire program from scratch on paper, and get it working in your mind before you started trying to put it onto cards, because if you made a mistake you had to re-punch many of the cards. It was very hard to visualize what you were working with. The card readers had a tendency to jam (the old one at my high school was nicknamed the "IBM 1443 card chewer".) And heaven help you if you dropped a stack of cards on the floor... :^) Still however, paper was used as the primary storage medium for many years.

The next big advance over paper was the creation of *magnetic tape*. Almost everyone has at least seen pictures of the large reels of tape used in older computers. Recording information in a way similar to how audio is recorded on a tape, these magnetic tapes were much more flexible, durable and faster than paper tape or punch cards. Of course, tape is still used today on modern computers, but as a form of *offline* or secondary storage. Before hard disks, they were the primary storage for some computers. Their primary disadvantage is that they must be read linearly; it can take minutes to move from one end of the tape to the other, making *random access* impractical.

Warning: Nostalgia mode activated. Be very afraid. :^)

Personal computers developed much later than the early, large mainframes, and were therefore the beneficiaries of advancements in storage technologies fairly early on in their existence. My first computer was purchased for me by my parents in 1980: an Apple][. A great little machine for learning on, using it gave me a profound appreciation for the importance of storage: because it had *none*! No hard disk drive, not even a floppy disk drive. My choices were to type in programs by hand (which I did sometimes) or try to load them from a *cassette tape*. Yes, an audio cassette tape. If you thought modern computer tape drives were unreliable, you should have tried getting *that* to work! :^) (Oh, and I also had to walk barefoot through three feet of snow to get to school... uphill both ways!)

I later purchased a low-density, single-sided floppy disk drive for my Apple. Boy, what a feeling of freedom that was! I could load and save programs and data easily, something I could never do before. That disk drive cost C$700 (back when the Canadian dollar was worth not much less than the U.S. dollar.) The biggest advantages of floppy disks over tapes are the ability to randomly access the data, and much better portability. They don't have nearly as much capacity however.

The first IBM PCs also had no hard disk drive, but rather employed one or two floppy disk drives. While of course far better than nothing, floppy disk drives were slow, small in capacity and relatively unreliable compared to even the earliest hard disks.

Next: Early Disk Drives

Early Disk Drives

The very first disk drives were of course experiments. Researchers, particularly those at IBM, were working with a number of different technologies and concepts to try to develop a disk drive that would be feasible for commercial development. In fact, the very first drives were not "disk drives" at all--they used rotating cylindrical drums, upon which the magnetic patterns of data were stored. The drums were large and hard to work with.

The earliest "true" hard disks had the heads of the hard disk in contact with the surface of the disk. This was done to allow the low-sensitivity electronics of the day to be able to better read the magnetic fields on the surface of the disk. Unfortunately, manufacturing techniques were not nearly as sophisticated as they are now, and it was not possible to get the disk's surface as smooth as would be necessary to allow the head to slide smoothly over the surface of the disk at high speed while in contact with it. Over time the heads would wear out, or wear out the magnetic coating on the surface of the disk.

The key technological breakthrough that enabled the creation of the modern hard disk came in the 1950s. IBM engineers realized that with the proper design the heads could be suspended above the surface of the disk and read the bits as they passed underneath. With this critical discovery that contact with the surface of the disk was not necessary, the basis for the modern hard disk was born.

The very first production hard disk was the IBM 305 RAMAC (Random Access Method of Accounting and Control), introduced on September 13, 1956. This beastie stored 5 million characters (approximately five megabytes, but a "character" in those days was only seven bits, not eight) on a whopping *50* disks, each 24 inches in diameter! Its areal density was about 2,000 bits per square inch; in comparison, today's drives have areal densities measured in billions of bits per square inch. The data transfer rate of this first drive was an impressive 8,800 bytes per second. :^)

Over the succeeding years, the technology improved incrementally; areal density, capacity and performance all increased. In 1962, IBM introduced the model 1301 Advanced Disk File. The key advance of this disk drive was the creation of heads that floated, or flew, above the surface of the disk on an "air bearing", reducing the distance from the heads to the surface of the disks from 800 to 250 microinches.

In 1973, IBM introduced the model 3340 disk drive, which is commonly considered to be the father of the modern hard disk. This unit had two separate spindles, one permanent and the other removable, each with a capacity of 30 MB. For this reason the disk was sometimes referred to as the "30-30". This name led to its being nicknamed the "Winchester" disk drive, after the famous "30-30" Winchester rifle. Using the first sealed internal environment and vastly improved "air bearing" technology, the Winchester disk drive greatly reduced the flying height of the disk: to only 17 microinches above the surface of the disk. Modern hard disks today still use many concepts first introduced in this early drive, and for this reason are sometimes still called "Winchester" drives.

The first hard disk drive designed in the 5.25" form factor used in the first PCs was the Seagate ST-506. It featured four heads and a 5 MB capacity. IBM bypassed the ST-506 and chose the ST-412--a 10 MB disk in the same form factor--for the IBM PC/XT, making it the first hard disk drive widely used in the PC and PC-compatible world.

Next: Key Technological Firsts

Key Technological Firsts

There have been a number of important "firsts" in the world of hard disks over their first 40 or so years. The following is a list, in chronological order, of some of the products developed during the past half-century that introduced key or important technologies in the PC world. Note the dominance of *IBM* in the list; in this author's opinion Big Blue does not get nearly as much credit as it deserves for being the main innovator in the storage world. Note also how many years it took for many of these technologies to make it to the PC world (sometimes as much as a decade, due to the initial high cost of most new technologies). I

- **First Hard Disk (1956):** IBM's RAMAC is introduced. It has a capacity of about 5 MB, stored on 50 24" disks. Its areal density is a mere 2,000 bits per square inch and its data throughput 8,800 bits/s.
- **First Air Bearing Heads (1962):** IBM's model 1301 lowers the flying height of the heads to 250 microinches. It has a 28 MB capacity on half as many heads as the original RAMAC, and increases both areal density and throughput by about 1000%.
- **First Removable Disk Drive (1965):** IBM's model 2310 is the first disk drive with a removable disk pack. While many PC users think of removable hard disks as being a modern invention, in fact they were very popular in the 1960s and 1970s.
- **First Ferrite Heads (1966):** IBM's model 2314 is the first hard disk to use ferrite core heads, the first type later used on PC hard disks.
- **First Modern Hard Disk Design (1973):** IBM's model 3340, nicknamed the "Winchester", is introduced. With a capacity of 60 MB it introduces several key technologies that lead to it being considered by many the ancestor of the modern disk drive.
- **First Thin Film Heads (1979):** IBM's model 3370 is the first with thin film heads, which would for many years be the standard in the PC industry.
- **First Eight-Inch Form Factor Disk (1979):** IBM's model 3310 is the first disk drive with 8" platters, greatly reduced in size from the 14" that had been the standard for over a decade.
- **First 5.25" Form Factor Disk (1980):** Seagate's ST-506 is the first drive in the 5.25" form factor, used in the earliest PCs.
- **First 3.5" Form Factor Disk Drive (1983):** Rodime introduces the RO352, the first disk drive to use the 3.5" form factor, which became one of the most important industry standards.
- **First Expansion Card Disk Drive (1985):** Quantum introduces the *Hardcard*, a 10.5 MB hard disk mounted on an ISA expansion card for PCs that were originally built without a hard disk. This product put Quantum "on the map" so to speak.
- **First Voice Coil Actuator 3.5" Drive (1986):** Conner Peripherals introduces the CP340, the first disk drive to use a voice coil actuator.
- **First "Low-Profile" 3.5" Disk Drive (1988):** Conner Peripherals introduces the CP3022, which was the first 3.5" drive to use the

reduced 1" height now called "low profile" and the standard for modern 3.5" drives.

- **First 2.5" Form Factor Disk Drive (1988):** PrairieTek introduces a drive using 2.5" platters. This size would later become a standard for portable computing.
- **First Drive to use Magnetoresistive Heads and PRML Data Decoding (1990):** IBM's model 681 (Redwing), an 857 MB drive, is the first to use MR heads and PRML.
- **First Thin Film Disks (1991):** IBM's "Pacifica" mainframe drive is the first to replace oxide media with thin film media on the platter surface.
- **First 1.8" Form Factor Disk Drive (1991):** Integral Peripherals' 1820 is the first hard disk with 1.8" platters, later used for PC-Card disk drives.
- **First 1.3" Form Factor Disk Drive (1992):** Hewlett Packard's C3013A is the first 1.3" drive.

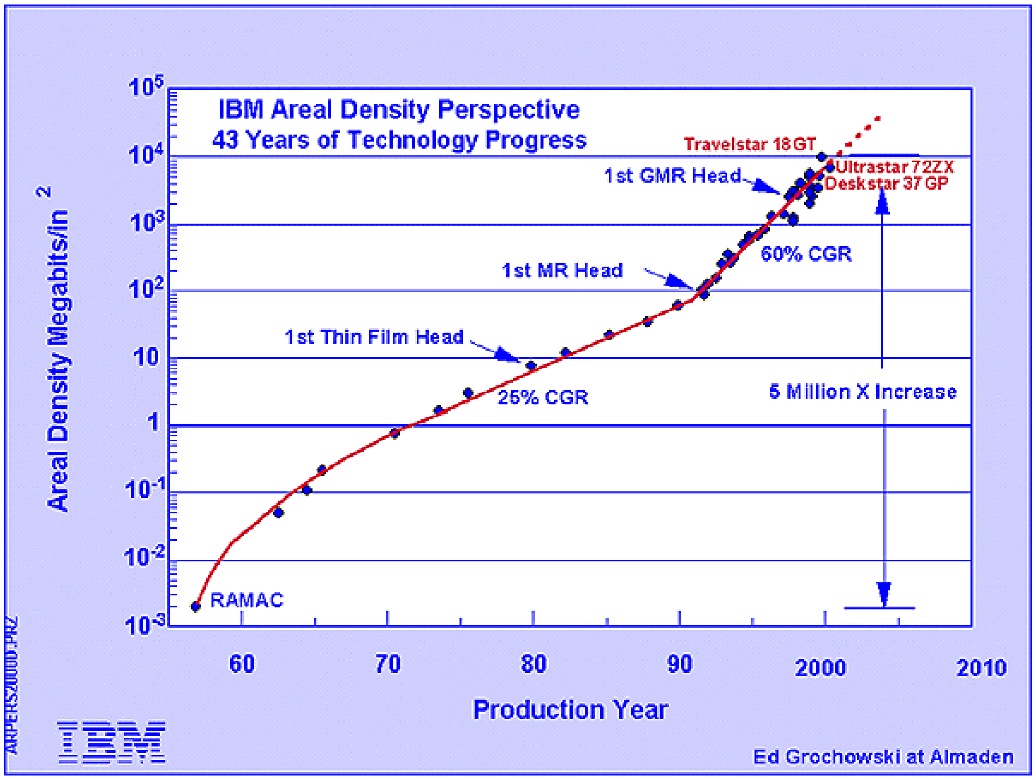
The source for much of this information is *DISK/TREND Inc.* In the 1990s, technological advances in every aspect of hard disks began coming at a fast and furious pace; it would take too long to research and list them all, so I am stopping at 1992. :^)

Next: Hard Disk Trends

Hard Disk Trends

The most amazing thing about hard disks is that they both change and *don't change* more than most other components. In terms of their basic design, today's hard disks aren't a lot different than the 10 MB clunkers installed in the first IBM PC/XTs in the early 1980s. However, in terms of their capacity, storage, reliability and other characteristics, hard drives have probably improved more than any other PC component. Let's take a look at some of the trends in various important hard disk characteristics:

- **Areal Density:** The areal density of hard disk platters continues to increase at an amazing rate even exceeding some of the optimistic predictions of a few years ago. Densities in the lab are now exceeding 35 Gbits/in^2, and modern disks are now packing as much as 20 GB of data onto a single 3.5" platter!



This chart shows the progress of areal density over the last 43 years. The red line is drawn as a best-fit through the blue diamonds which are actual products. Key hard disk head technology developments are indicated. Note that the scale on left is logarithmic, not linear.

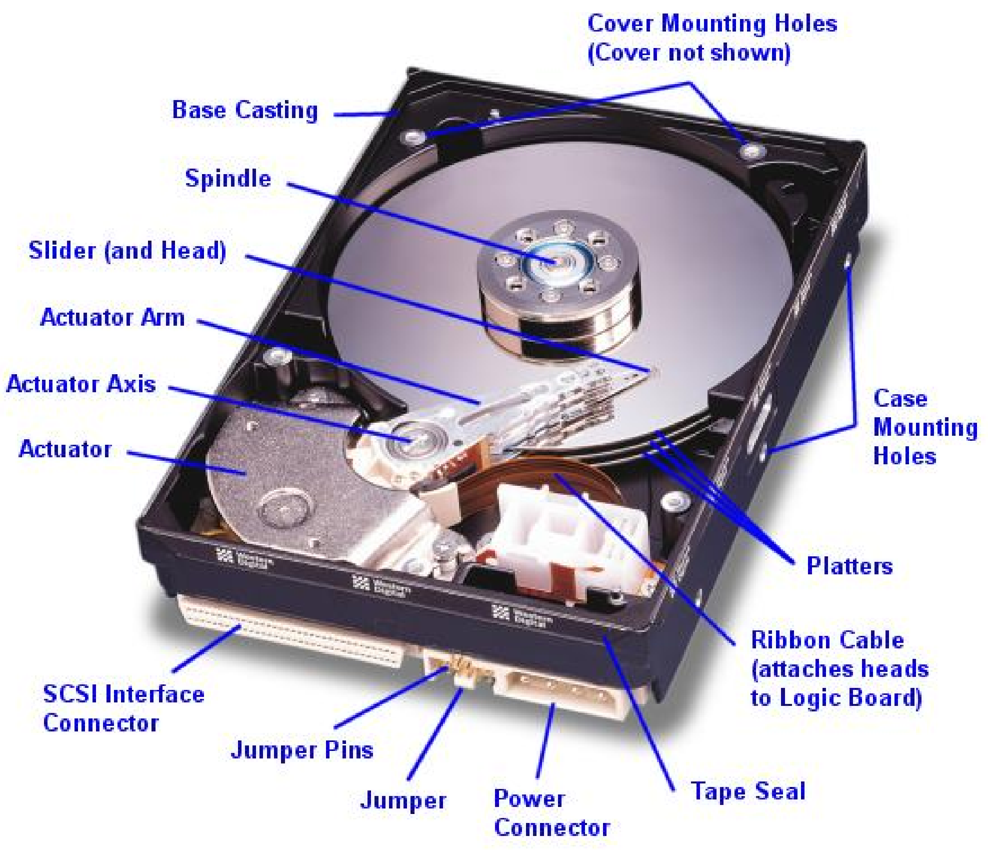
Image © IBM Corporation
Image used with permission.

- **Capacity:** Hard disk capacity continues to not only increase, but increase at an accelerating rate. From 10 MB in 1981, we are now well over 10 *GB* in 2000 and will probably hit 100 GB within a year for consumer drives.
- **Spindle Speed:** The move to faster and faster spindle speeds continues. Since increasing the spindle speed improves both random-access and sequential performance, this is likely to continue. Once the domain of high-end SCSI drives, 7200 RPM spindles are now standard on mainstream IDE/ATA drives. A 15,000 RPM SCSI drive was announced by Seagate in early 2000.
- **Form Factor:** The trend in form factors is downward: to smaller and smaller drives. 5.25" drives have now all but disappeared from the mainstream PC market, with 3.5" drives dominating the desktop and server segment. In the mobile world, 2.5" drives are the standard with smaller sizes becoming more prevalent; IBM in 1999 announced its *Microdrive* which is a tiny 170 MB or 340 MB device only an inch in diameter and less than 0.25" thick! Over the next few years, desktop and server drives are likely to transition to the 2.5" form factor as well. The primary reasons for this "shrinking trend" include the enhanced rigidity of smaller platters, reduction of mass to enable faster spin speeds, and improved reliability due to enhanced ease of manufacturing.
- **Performance:** Both positioning and transfer performance factors are improving. The speed with which data can be pulled from the disk is increasing more rapidly than positioning performance is improving, suggesting that over the next few years addressing seek time and latency will be the areas of greatest value to hard disk engineers.
- **Reliability:** The reliability of hard disks is improving slowly as manufacturers refine their processes and add new reliability-enhancing features, but this characteristic is not changing nearly as rapidly as the others above. One reason is that the technology is constantly changing, and the performance envelope constantly being pushed; it's much harder to improve the reliability of a product when it is changing rapidly.
- **RAID:** Once the province of only high-end servers, the use of multiple disk arrays to improve performance and reliability is becoming increasingly common, and is now even seen in consumer desktop machines. Over the next few years I predict that RAID will become the "next big thing" as the thirst for performance increases, and in five years we may see new PCs commonly shipping with multiple hard disks configured as an array.
- **Interfaces:** Despite the introduction to the PC world of new interfaces such as IEEE-1394 and USB (universal serial bus) the mainstream interfaces in the PC world are the same as they were through the 1990s: IDE/ATA and SCSI. The interfaces themselves continue to create new and improved standards with higher maximum transfer rates, to match the increase in performance of the hard disks themselves.

Next: Construction and Operation of the Hard Disk Drive

Construction and Operation of the Hard Disk

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.

Original image © Western Digital Corporation Image used with permission.

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.

Next: Hard Disk Operational Overview

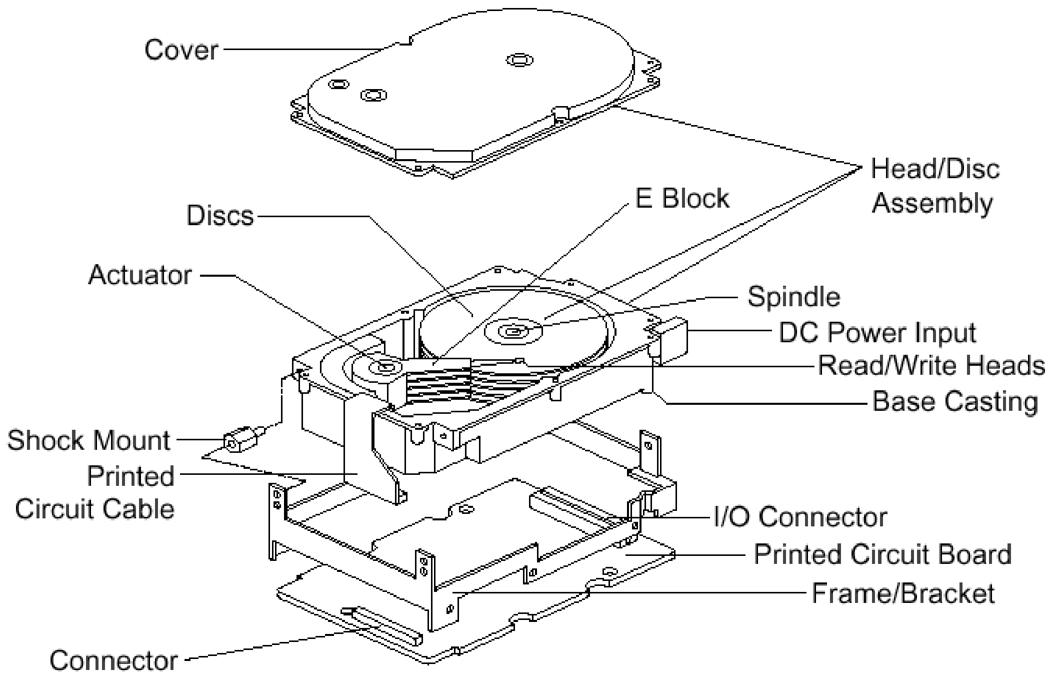
Hard Disk Operational Overview

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.

Original image © Seagate Technology
Image used with permission.

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.

Next: Hard Disk Platters and Media

Hard Disk Platters and Media

Every hard disk contains one or more flat disks that are used to actually hold the data in the drive. These disks are called *platters* (sometimes also "disks" or "discs"). They are composed of two main substances: a *substrate* material that forms the bulk of the platter and gives it structure and rigidity, and a *magnetic media coating* which actually holds the magnetic impulses that represent the data. Hard disks get their name from the rigidity of the platters used, as compared to floppy disks and other media which use flexible "platters" (actually, they aren't usually even called platters when the material is flexible.)

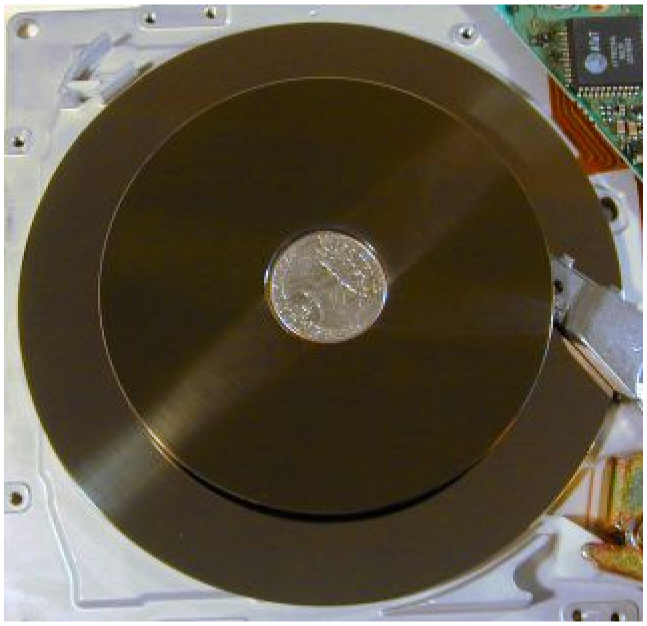
The platters are "where the action is"--this is where the data itself is recorded. For this reason the quality of the platters and particularly, their media coating, is critical. The surfaces of each platter are precision machined and treated to remove any imperfections, and the hard disk itself is assembled in a *clean room* to reduce the chances of any dirt or contamination getting onto the platters.

Next: Platter Size

Platter Size

The size of the platters in the hard disk is the primary determinant of its overall physical dimensions, also generally called the drive's *form factor*; most drives are produced in one of the various standard hard disk form factors. Disks are sometimes referred to by a size specification; for example, someone will talk about having a "3.5-inch hard disk". When this terminology is used it usually refers to the disk's form factor, and normally, the form factor is named based on the platter size. The platter size of the disk is usually the same for all drives of a given form factor, though not always, especially with the newest drives, as we will see below. Every platter in any specific hard disk has the same diameter.

The first PCs used hard disks that had a *nominal* size of 5.25". Today, by far the most common hard disk platter size in the PC world is 3.5". Actually, the platters of a 5.25" drive are 5.12" in diameter, and those of a 3.5" drive are 3.74"; but habits are habits and the "approximate" names are what are commonly used. You will also notice that these numbers correspond to the common sizes for floppy disks because they were designed to be mounted into the same drive bays in the case. Laptop drives are usually smaller, due to laptop manufacturers' never-ending quest for "lighter and smaller". The platters on these drives are usually 2.5" in diameter or less; 2.5" is the standard form factor, but drives with 1.8" and even 1.0" platters are becoming more common in mobile equipment.



A platter from a 5.25" hard disk, with a platter from a 3.5" hard disk placed on top of it for comparison. The quarter is included for scale (and strangely, fits right in the spindle hole for both platters... isn't that a strange coincidence?

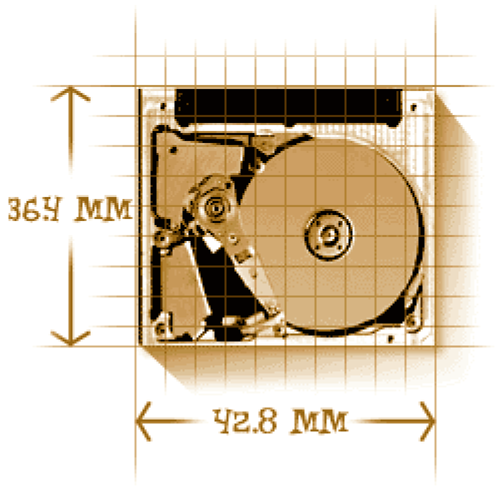
Traditionally, drives extend the platters to as much of the width of the physical drive package as possible, to maximize the amount of storage they can pack into the drive. However, as discussed in the section on hard disk historical trends, the trend overall is towards *smaller* platters. This might seem counter-intuitive; after all, larger platters mean there is more room to store data, so shouldn't it be more cost-effective for manufacturers to make platters as big as possible? There are several reasons why platters are shrinking, and they are primarily related to performance. The areal density of disks is increasing so quickly that the loss of capacity by going to smaller platters is viewed as not much of an issue--few people care when drives are doubling in size every year anyway!--while performance improvements continue to be at the top of nearly everyone's wish list. In fact, several hard disk manufacturers who were continuing to produce 5.25" drives for the "value segment" of the market as recently as 1999 have now discontinued them. (The very first hard disks were 24" in diameter, so you can see how far we have come in 40 or so years.)

Here are the main reasons why companies are going to smaller platters even for desktop units:

- **Enhanced Rigidity:** The *rigidity* of a platter refers to how stiff it is. Stiff platters are more resistant to shock and vibration, and are better-suited for being mated with higher-speed spindles and other high-performance hardware. Reducing the hard disk platter's diameter by a factor of two approximately *quadruples* its rigidity.
- **Manufacturing Ease:** The flatness and uniformity of a platter is critical to its quality; an ideal platter is perfectly flat and consistent. Imperfect platters lead to low manufacturing yield and the potential for data loss due to the heads contacting uneven spots on the surface of a platter. Smaller platters are easier to make than larger ones.
- **Mass Reduction:** For performance reasons, hard disk spindles are increasing in speed. Smaller platters are easier to spin and require less-powerful motors. They are also faster to spin up to speed from a stopped position.
- **Power Conservation:** The amount of power used by PCs is becoming more and more of a concern, especially for portable computing but even on the desktop. Smaller drives generally use less power than larger ones.
- **Noise and Heat Reduction:** These benefits follow directly from the improvements enumerated above.
- **Improved Seek Performance:** Reducing the size of the platters reduces the distance that the head actuator must move the heads side-to-side to perform random seeks; this improves seek time and makes random reads and writes faster. Of course, this is done at the cost of capacity; you could theoretically achieve the same performance improvement on a larger disk by only filling the inner cylinders of each platter. In fact, some demanding customers used to partition hard disks and use only a small portion of the disk, for exactly this reason: so that seeks would be faster. Using a smaller platter size is more efficient, simpler and less wasteful than this sort of "hack".

The trend towards smaller platter sizes in modern desktop and server drives began in earnest when some manufacturers "trimmed" the platters in their 10,000 RPM hard disk drives from 3.74" to 3" (while keeping them as standard 3.5" form factor drives on the outside for compatibility.) Seagate's Cheetah X15 15,000 RPM drive goes even further, dropping the platter size down to 2.5", again trading performance for capacity (it is "only" 18 GB, less than half the size of modern 3.5" platter-size drives.) This drive, despite having 2.5" platters, still uses the common 3.5" form factor for external mounting (to maintain compatibility with standard cases), muddying the "size" waters to some extent (it's a "3.5-inch drive" but it doesn't have 3.5" platters.)

The smallest hard disk platter size available on the market today is a miniscule 1" in diameter! IBM's amazing Microdrive has a single platter and is designed to fit into digital cameras, personal organizers, and other small equipment. The tiny size of the platters enables the Microdrive to run off battery power, spin down and back up again in less than a second, and withstand shock that would destroy a normal hard disk. The downside? It's "only" 340 MB. :^)



Internal view and dimensions of the amazing IBM Microdrive.

Image © IBM Corporation
Image used with permission.

Here's a summary table showing the most common platter sizes used in PCs, in order of decreasing size (which in most cases is also chronological order from their data of introduction, but not always) and also showing the most common form factors used by each technology:

Platter Diameter	**Typical Form Factor**	**Application**
5.12	5.25"	Oldest PCs, used in servers through the

		mid-1990s and some retail drives in the mid-to-late 1990s; now obsolete
3.74	3.5"	Standard platter size for the most common hard disk drives used in PCs
3.0	3.5"	High-end 10,000 RPM drives
2.5	2.5", 3.5"	Laptop drives (2.5" form factor); 15,000 RPM drives (3.5" form factor)
1.8	PC Card (PCMCIA)	PC Card (PCMCIA) drives for laptops
1.3	PC Card (PCMCIA)	Originally used on hand-held PCs (no longer made)
1.0	CompactFlash	Digital cameras, hand-held PCs and other consumer electronic devices

Next: Number of Platters

Number of Platters

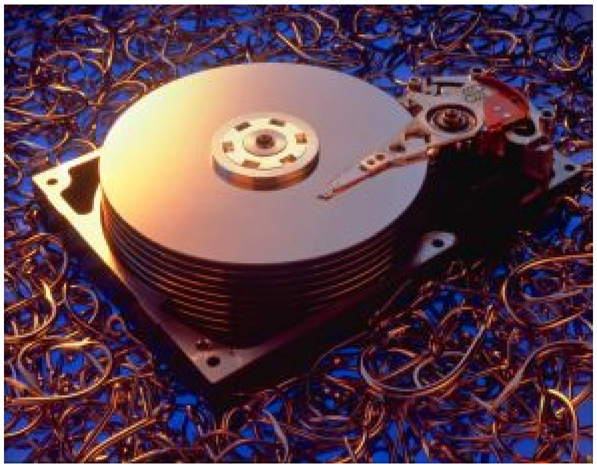
Hard disks can have one platter, or more, depending on the design. Standard consumer hard disks, the type probably in your PC right now, usually have between one and five platters in them. Some high-end drives--usually used in servers--have as many as a dozen platters. Some very old drives had even more. In every drive, all the platters are physically connected together on a common central spindle, to form a single assembly that spins as one unit, driven by the spindle motor. The platters are kept apart using spacer rings that fit over the spindle. The entire assembly is secured from the top using a cap or cover and several screws. (See the spindle motor page for an illustration of these components.)

Each platter has two *surfaces* that are capable of holding data; each surface has a read/write head. Normally both surfaces of each platter are used, but that is not always the case. Some older drives that use dedicated servo positioning reserve one surface for holding servo information. Newer drives don't need to spend a surface on servo information, but sometimes leave a surface unused for *marketing reasons*--to create a drive of a particular capacity in a family of drives. With modern drives packing huge amounts of data on a single platter, using only one surface of a platter allows for increased "granularity". For example, IBM's Deskstar 40GV family sports an impressive 20 GB per platter data capacity. Since IBM wanted to make a 30 version of this drive, they used three surfaces (on two platters) for that drive. Here's a good illustration of how Western Digital created five different capacities using three platters in their Caviar line of hard disk drives:

Model Number	Nominal Size (GB)	Data Sectors Per Drive	Platters	Surfaces
WD64AA	6.4	12,594,960	1	2
WD102AA	10.2	20,044,080	2	3
WD136AA	13.6	26,564,832	2	4
WD172AA	17.2	33,687,360	3	5
WD205AA	20.5	40,079,088	3	6

Note: In theory, using only one surface means manufacturing costs can be saved by making use of platters that have unacceptable defects on one surface, but I don't know if this optimizing is done in practice...

From an engineering standpoint there are several factors that are related to the number of platters used in the disk. Drives with many platters are more difficult to engineer due to the increased mass of the spindle unit, the need to perfectly align all the drives, and the greater difficulty in keeping noise and vibration under control. More platters also means more mass, and therefore slower response to commands to start or stop the drive; this can be compensated for with a stronger spindle motor, but that leads to other tradeoffs. In fact, the trend recently has been towards drives with *fewer* head arms and platters, not more. Areal density continues to increase, allowing the creation of large drives without using a lot of platters. This enables manufacturers to reduce platter count to improve seek time without creating drives too small for the marketplace. See here for more on this trend.



This Barracuda hard disk has 10 platters. (I find the choice of fish hooks as a background for this shot highly amusing. :^))

Original image © Seagate Technology
Image used with permission.

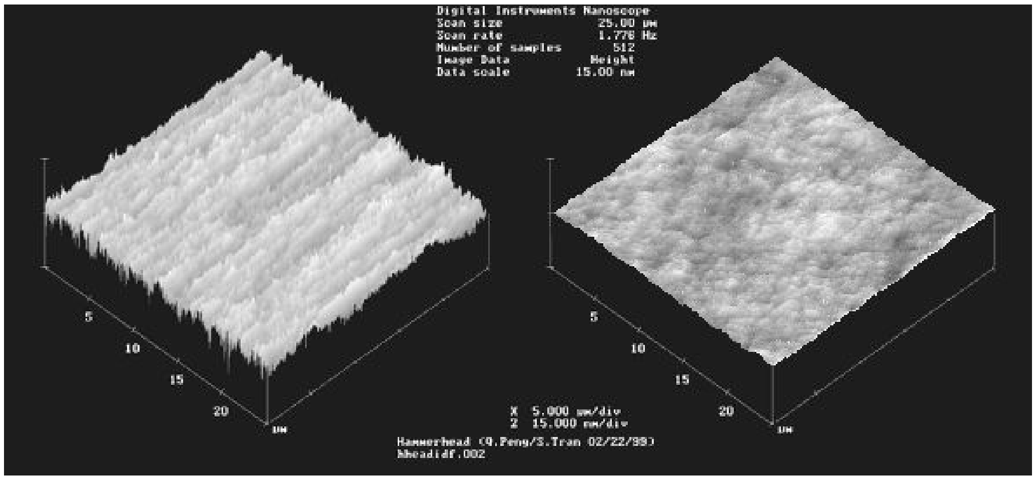
The form factor of the hard disk also has a great influence on the number of platters in a drive. Even if hard disk engineers wanted to put lots of platters in a particular model, the standard PC "slimline" hard disk form factor is limited to 1 inch in height, which limits the number of platters that can be put in a single unit. Larger 1.6-inch "half height" drives are often found in servers and usually have many more platters than desktop PC drives. Of course, engineers are constantly working to reduce the amount of clearance required between platters, so they can increase the number of platters in drives of a given height.

Next: Platter Substrate Materials

Platter Substrate Materials

The magnetic patterns that comprise your data are recorded in a very thin media layer on the surfaces of the hard disk's platters; the bulk of the material of the platter is called the *substrate* and does nothing but support the media layer. To be suitable, a substrate material must be rigid, easy to work with, lightweight, stable, magnetically inert, inexpensive and readily available. The most commonly used material for making platters has traditionally been an aluminum alloy, which meets all of these criteria.

Due to the way the platters spin with the read/write heads floating just above them, the platters must be extremely smooth and flat. With older, slower spindle drives and relatively high fly heights, the uniformity of the platter surface was less of an issue. Now, as technology advances, the gap between the heads and the platter is decreasing, and the speed that the platters spin at is increasing, creating more demands on the platter material itself. Uneven platter surfaces on hard disks running at faster speeds with heads closer to the surface are more apt to lead to head crashes. For this reason many drive makers began several years ago to look at alternatives to aluminum, such as glass, glass composites, and magnesium alloys.



Hard disk platters are very smooth, right? Well, not to a scanning electron microscope! The image on the left is of the surface of an aluminum alloy platter; the one on the right is a glass platter. The images speak for themselves. The scale is in microns.

Composed from two original images © IBM Corporation Images used with permission.

It now is looking increasingly likely that *glass* and composites made with glass will be the next standard for the platter substrate. IBM has been shipping drives with glass platters for several years and in 2000 is introducing them into the IDE/ATA consumer drive market. Compared to aluminum platters, glass platters have several advantages:

- **Better Quality:** The first and most important reason for going to glass is probably that glass platters can be made much smoother and flatter than aluminum, improving the reliability of the hard disk and making low flying heights and faster spindle speeds more feasible.
- **Improved Rigidity:** Another important consideration is that glass is more rigid than aluminum for the same weight of material. Improved rigidity, one of the reasons why platter sizes are also shrinking in size, is important for reducing noise and vibration with drives that spin at high speed.
- **Thinner Platters:** The enhanced rigidity of glass also allows platters to be made thinner than with aluminum, allowing more platters to be packed into the same drive dimensions. Thinner platters also weigh less, reducing spindle motor requirements and reducing start time when the drive is at rest.
- **Thermal Stability:** When heated, glass expands much less than does aluminum. With some hard disk platters now containing 35,000 tracks per inch or more, even a small amount of expansion can causes these tracks to "move around". The drive's servo mechanism compensates for expansion and contraction, but it is still preferable to use materials that move as little as possible because this reduces the amount of adjusting the hard drive has to do, improving performance.

One obvious *disadvantage* of glass compared to aluminum is *fragility*, particularly when made very thin. For this reason some companies are experimenting with glass/ceramic composites. One of these is a Dow Corning product called *MemCor*, which is a glass made with ceramic inserts to reduce the likelihood of cracking. Sometimes these composites are just called "glass", much the way aluminum alloy platters, which usually contain other metals, are just called "aluminum".

Next: Magnetic Media

Magnetic Media

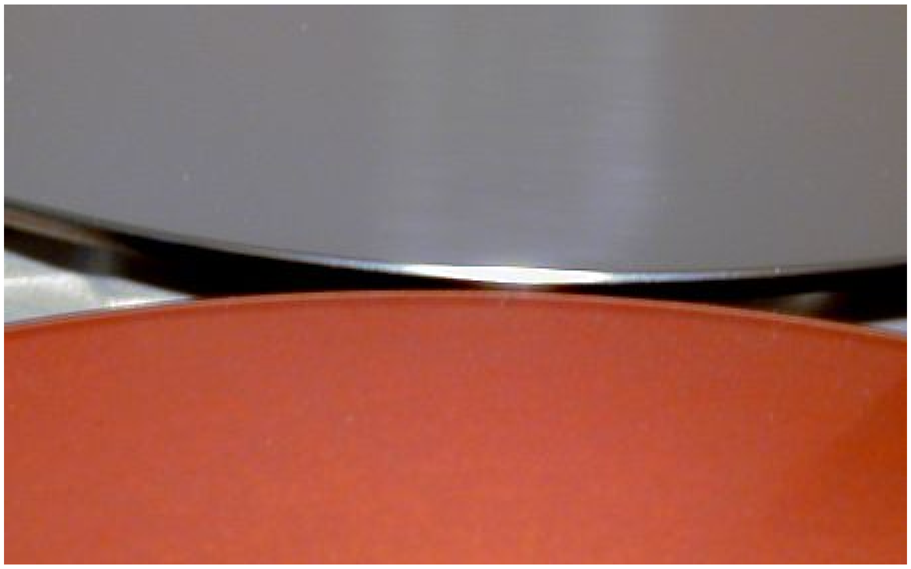
The substrate material of which the platters are made forms the base upon which the actual recording *media* is deposited. The media layer is a very thin coating of magnetic material which is where the actual data is stored; it is typically only a few millionths of an inch in thickness.

Older hard disks used *oxide media*. "Oxide" really means *iron oxide*--rust. Of course no high-tech company wants to say they use *rust* in their products, so they instead say something like "high-performance oxide media layer". :^) But in fact that's basically what oxide media is, particles of rust attached to the surface of the platter substrate using a binding agent. You can actually see this if you look at the surface of an older hard disk platter: it has the characteristic light brown color. This type of media is similar to what is used in audio cassette tape (which has a similar color.)

Oxide media is inexpensive to use, but also has several important shortcomings. The first is that it is a soft material, and easily damaged from contact by a read/write head. The second is that it is only useful for relatively low-density storage. It worked fine for older hard disks with relatively low data density, but as manufacturers sought to pack more and more data into the same space, oxide was not up to the task: the oxide particles became too large for the small magnetic fields of newer designs.

Today's hard disks use *thin film media*. As the name suggests, thin film media consists of a very thin layer of magnetic material applied to the surface of the platters. (While oxide media certainly isn't *thick* by any reasonable use of the word, it was much thicker than this new media material; hence the name "thin film".) Special manufacturing techniques are employed to deposit the media material on the platters. One method is *electroplating*, which deposits the material on the platters using a process similar to that used in electroplating jewelry. Another is *sputtering*, which uses a vapor-deposition process borrowed from the manufacture of semiconductors to deposit an extremely thin layer of magnetic material on the surface. Sputtered platters have the advantage of a more uniform and flat surface than plating. Due to the increased need for high quality on newer drives, sputtering is the primary method used on new disk drives, despite its higher cost.

Compared to oxide media, thin film media is much more uniform and smooth. It also has greatly superior magnetic properties, allowing it to hold much more data in the same amount of space. Finally, it's a much harder and more durable material than oxide, and therefore much less susceptible to damage.



A thin film 5.25" platter (above) next to an oxide 5.25" platter (below). Thin film platters are actually reflective; taking photographs of them is like trying to take a picture of a mirror! This is one reason why companies always display internal hard disk pictures at an angle.

After applying the magnetic media, the surface of each platter is usually covered with a thin, protective, layer made of carbon. On top of this is added a super-thin lubricating layer. These material are used to protect the disk from damage caused by accidental contact from the heads or other foreign matter that might get into the drive.

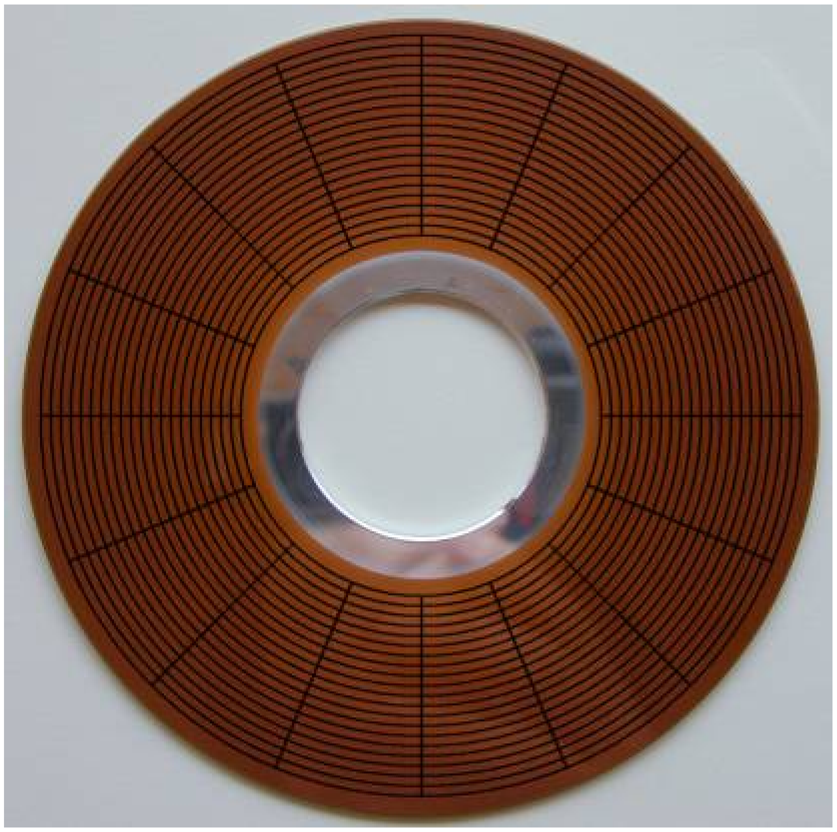
IBM's researchers are now working on a fascinating, experimental new substance that may replace thin film media in the years ahead. Rather than sputtering a metallic film onto the surface, a chemical solution containing organic molecules and particles of iron and platinum is applied to the platters. The solution is spread out and heated. When this is done, the iron and platinum particles arrange themselves naturally into a grid of crystals, with each crystal able to hold a magnetic charge. IBM is calling this structure a "nanocrystal superlattice". This technology has the potential to increase the areal density capability of the recording media of hard disks by as much as 10 or even 100 times! Of course it is years away, and will need to be matched by advances in other areas of the hard disk (particularly read/write head capabilities) but it is still pretty amazing and shows that magnetic storage still has a long way to go before it runs out of room for improvement.

Next: Tracks and Sectors

Tracks and Sectors

Platters are organized into specific structures to enable the organized storage and retrieval of data. Each platter is broken into *tracks*--tens of thousands of them--which are tightly-packed concentric circles. These are similar in structure to the annual rings of a tree (but *not* similar to the grooves in a vinyl record album, which form a connected spiral and not concentric rings).

A track holds too much information to be suitable as the smallest unit of storage on a disk, so each one is further broken down into *sectors*. A sector is normally the smallest individually-addressable unit of information stored on a hard disk, and normally holds 512 bytes of information. The first PC hard disks typically held 17 sectors per track. Today's hard disks can have *thousands* of sectors in a single track, and make use of zoned recording to allow more sectors on the larger outer tracks of the disk.



A platter from a 5.25" hard disk, with 20 concentric tracks drawn over the surface. This is far lower than the density of even the oldest hard disks; even if visible, the tracks on a modern hard disk would require high magnification to resolve. Each track is divided into 16 imaginary sectors. Older hard disks had the same number of sectors per track, but new ones use zoned recording with a different number of sectors per track in different zones of tracks.

A detailed examination of tracks and sectors leads into a larger discussion of disk geometry, encoding methods, formatting and other topics. Full coverage of hard disk tracks and sectors can be found here, with detail on sectors specifically here.

Next: Areal Density

Areal Density

Areal density, also sometimes called *bit density*, refers to the amount of data that can be stored in a given amount of hard disk platter "real estate". Since disk platters surfaces are of course two-dimensional, areal density is a measure of the number of bits that can be stored in a unit of area. It is usually expressed in bits per square inch (BPSI).

Being a two-dimensional measure, areal density is computed as the product of two other one-dimensional density measures:

- **Track Density:** This is a measure of how tightly the concentric tracks on the disk are packed: how many tracks can be placed down in inch of radius on the platters. For example, if we have a platter that is 3.74" in diameter, that's about 1.87 inches. Of course the inner portion of the platter is where the spindle is, and the very outside of the platter can't be used either. Let's say about 1.2 inches of length along the radius is usable for storage. If in that amount of space the hard disk has 22,000 tracks, the track density of the drive would be approximately 18,333 tracks per inch (TPI).
- **Linear or Recording Density:** This is a measure of how tightly the bits are packed within a length of track. If in a given inch of a track we can record 200,000 bits of information, then the linear density for that track is 200,000 bits per inch per track (BPI). Every track on the surface of a platter is a different length (because they are concentric circles), and not every track is written with the same density. Manufacturers usually quote the *maximum* linear density used on each drive.

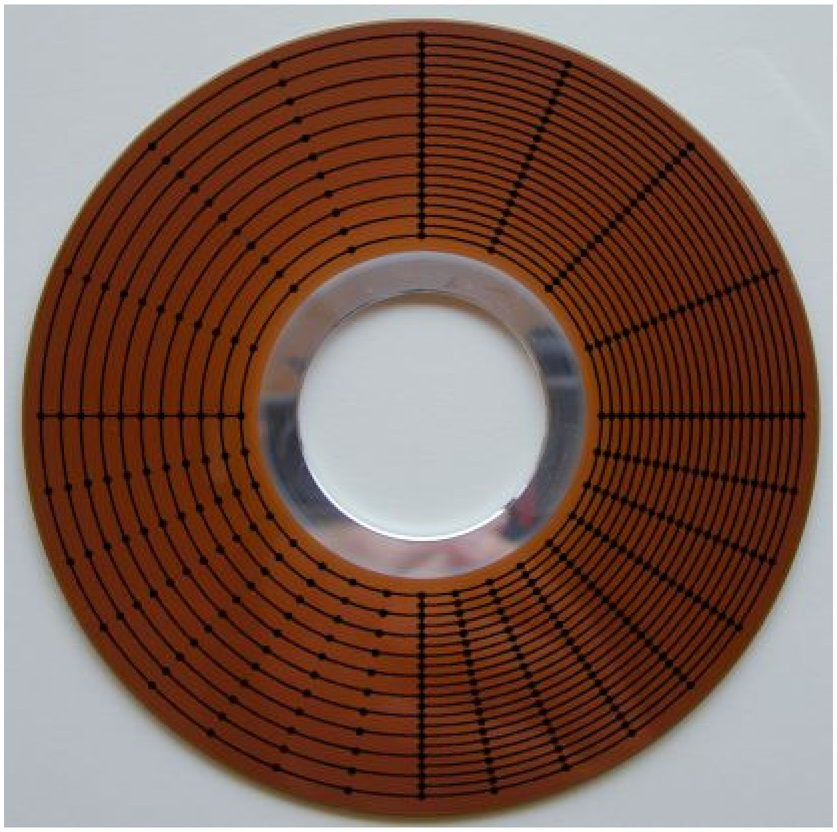
Taking the product of these two values yields the drive's areal density, measured in bits per square inch. If the maximum linear density of the drive above is 300,000 bits per inch of track, its maximum areal density would be 5,500,000,000 bits per square inch, or in more convenient notation, 5.5 Gbits/in$^{2.}$ The newest drives have areal densities exceeding 10 Gbits/in^2, and in the lab IBM in 1999 reached 35.3 Gbits/in^2--524,000 BPI linear density, and 67,300 TPI track density! In contrast, the first PC hard disk had an areal density of about 0.004 Gbits/in^2!

Note: Sometimes you will see areal density expressed in a different way: gigabytes per platter (GB/platter). This unit is often used when comparing drives, and is really a different way of saying the same thing--*as long as you are always clear about what the platter size is*. It's easier conceptually for many people to contrast two units by saying, for example: "Drive A has 10 GB/platter and drive B has 6 GB/platter, so A has higher density". Also, it's generally easier to compute when the true areal density numbers are not easy to find. As long as they both have the same platter size, the comparison is valid. Otherwise you are comparing apples and oranges.

The linear density of a disk is not constant over its entire surface--bear this in mind when reading density specifications, which usually list only the *maximum* density of the disk. The reason that density varies is because the lengths of the tracks increase as you move from the inside tracks to the outside tracks, so outer tracks can hold more data than inner ones. This would mean that if you stored the same amount of data on the outer tracks as the inner ones, the linear density of the outer tracks would be *much* lower than the linear density of the inner tracks. This is in fact how drives used to be, until the creation of zoned bit recording, which packs more data onto the outer tracks to exploit their length. However, the inner tracks still generally have higher density than the outer ones, with density gradually decreasing from the inner tracks to the outer ones.

Tip: Here's how you can figure this out yourself. A typical 3.5" hard disk drive has an innermost track circumference of about 4.75" and an outermost track circumference of about 11". The ratio of these two numbers is about 2.32. What this means is that there is 2.32 times as much "room" on the outer tracks. If you look at the number of sectors per track on the outside zone of the disk, and it is *less than* 2.32 times the number of sectors per track of the inside zone, then you know the outer tracks have lower density than the inner ones. Consider the IBM 40GV drive, whose zones are shown in a table on this page on zoned bit recording. Its outermost tracks have 792 sectors; its innermost tracks 370. This is a ratio of 2.14, so assuming this drive uses tracks with the lengths I mentioned, it has its highest density on the innermost tracks.

There are two ways to increase areal density: increase the linear density by packing the bits on each track closer together so that each track holds more data; or increase the track density so that each platter holds more tracks. Typically new generation drives improve both measures. It's important to realize that increasing areal density leads to drives that are not just bigger, but also *faster*, all else being equal. The reason is that the areal density of the disk impacts both of the key hard disk performance factors: both positioning speed and data transfer rate. See this section for a full discussion of the impact of areal density on hard disk performance.



This illustration shows how areal density works. First, divide the circle in your mind into a left and right half, and then a top and bottom half. The left half shows low track density and the right half high track density. The upper half shows low linear density, and the bottom half high linear density.
Combining them, the upper left quadrant has the lowest areal density; the upper right and lower left have greater density, and the bottom right quadrant of course has the highest density.
(I hope you like this illustration, because you don't want to know what I went through trying to make it... :^))

Increasing the areal density of disks is a difficult task that requires many technological advances and changes to various components of the hard disk. As the data is packed closer and closer together, problems result with interference between bits. This is often dealt with by reducing the strength of the magnetic signals stored on the disk, but then this creates other problems such as ensuring that the signals are stable on the disk and that the read/write heads are sensitive and close enough to the surface to pick them up. In some cases the heads must be made to fly closer to the disk, which causes other engineering challenges, such as ensuring that the disks are flat enough to reduce the chance of a head crash. Changes to the media layer on the platters, actuators, control electronics and other components are made to continually improve areal density. This is especially true of the read/write heads. Every few years a read/write head technology breakthrough enables a significant jump in density, which is why hard disks have been doubling in size so frequently--in some cases it takes less than a year for the leading drives on the market to double in size.

Next: Hard Disk Read/Write Heads

Hard Disk Read/Write Heads

The read/write heads of the hard disk are the interface between the magnetic physical media on which the data is stored and the electronic components that make up the rest of the hard disk (and the PC). The heads do the work of converting bits to magnetic pulses and storing them on the platters, and then reversing the process when the data needs to be read back.

Read/write heads are an extremely critical component in determining the overall performance of the hard disk, since they play such an important role in the storage and retrieval of data. They are usually one of the more expensive parts of the hard disk, and to enable areal densities and disk spin speeds to increase, they have had to evolve from rather humble, clumsy beginnings to being extremely advanced and complicated technology. New head technologies are often the triggering point to increasing the speed and size of modern hard disks.

Next: Read/Write Head Operation

Hard Disk Read/Write Head Operation

In many ways, the read/write heads are the most sophisticated part of the hard disk, which is itself a technological marvel. They don't get very much attention, perhaps in part because most people never see them. This section takes a look at how the heads work and discusses some of their key operating features and issues.

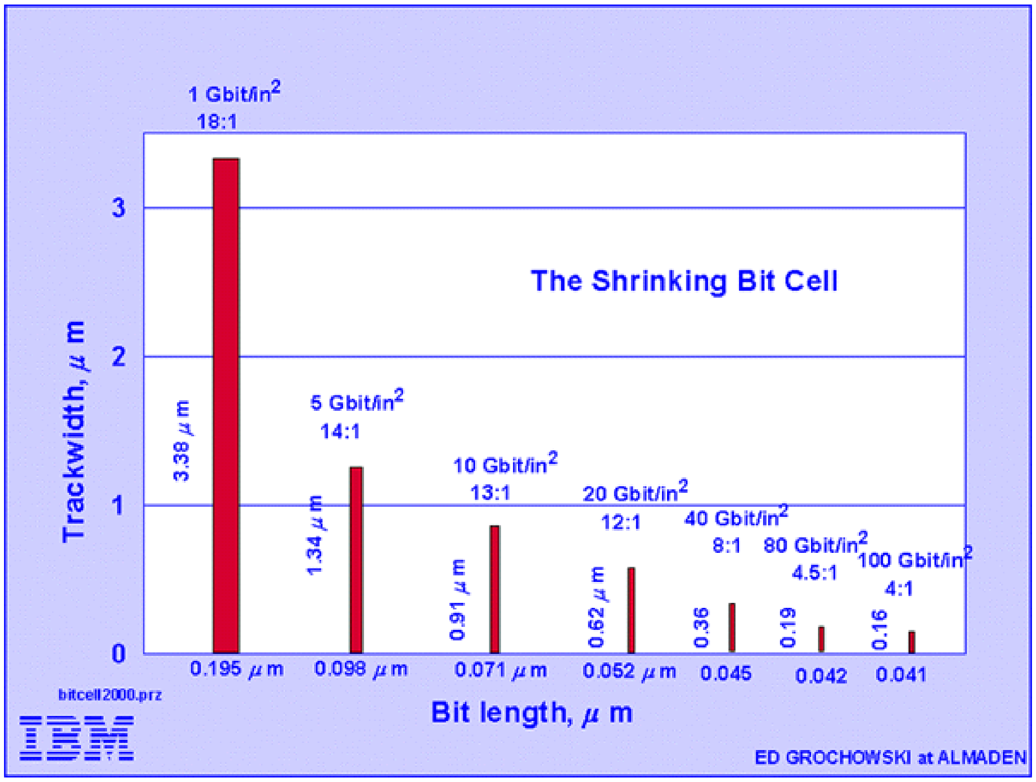
Next: Function of the Read/Write Heads

Function of the Read/Write Heads

In concept, hard disk heads are relatively simple. They are energy converters: they transform electrical signals to magnetic signals, and magnetic signals back to electrical ones again. The heads on your VCR or home stereo tape deck perform a similar function, although using very different technology. The read/write heads are in essence tiny electromagnets that perform this conversion from electrical information to magnetic and back again. Each bit of data to be stored is recorded onto the hard disk using a special encoding method that translates zeros and ones into patterns of magnetic flux reversals.

Older, conventional (ferrite, metal-in-gap and thin film) hard disk heads work by making use of the two main principles of electromagnetic force. The first is that applying an electrical current through a coil produces a magnetic field; this is used when writing to the disk. The direction of the magnetic field produced depends on the direction that the current is flowing through the coil. The second is the opposite, that applying a magnetic field to a coil will cause an electrical current to flow; this is used when reading back the previously written information. (You can see a photograph showing this design on the page on ferrite heads.) Again here, the direction that the current flows depends on the direction of the magnetic field applied to the coil. Newer (MR and GMR) heads don't use the induced current in the coil to read back the information; they function instead by using the principle of *magnetoresistance*, where certain materials change their resistance when subjected to different magnetic fields.

The heads are usually called "read/write heads", and older ones did both writing and reading using the same element. Newer MR and GMR heads however, are in fact composites that include a different element for writing and reading. This design is more complicated to manufacture, but is required because the magnetoresistance effect used in these heads only functions in the read mode. Having separate units for writing and reading also allows each to be tuned to the particular function it does, while a single head must be designed as a compromise between fine-tuning for the write function or the read function. See the discussion of MR read heads for more on this. These dual heads are sometimes called "merged heads".



This graph shows how the bit size of hard disks is shrinking over time: dramatically. The width and length of each bit are shown for hard disks using varies areal densities. Current high-end hard disks have exceeded 10 Gbit/in^2 in areal density, but it has been only a few years since 1 Gbit/in^2 was state of the art. As the bit size drops and the bits are packed closer together, magnetic fields become weaker and more sensitive head electronics are required to properly detect and interpret the data signals.

Original image © IBM Corporation
Image used with permission.

Because of the tight packing of data bits on the hard disk, it is important to make sure that the magnetic fields don't interfere with one another. To ensure that this does not happen, the stored fields are very small, and very weak. Increasing the density of the disk means that the fields must be made still weaker, which means the read/write heads must be faster and more sensitive so they can read the weaker fields and accurately figure out which bits are ones and which bits are zeroes. This is the reason why MR and GMR heads have taken over the market: they are more sensitive and can be made very small so as not read adjacent tracks on the disk. Special amplification circuits are used to convert the weak electrical pulses from the head into proper digital signals that represent the real data read from the hard disk. Error detection and correction circuitry must also be used to compensate for the increased likelihood of errors as the signals get weaker and weaker on the hard disk. In addition, some newer heads employ magnetic "shields" on either side of the read head to ensure that the head is not affected by any stray magnetic energy.

Next: Number of Read/Write Heads

Number of Read / Write Heads

Each hard disk platter has two surfaces, one on each side of the platter. More often than not, both surfaces of the platter are used for data on modern drives, but as described in the section discussing the number of platters in a drive, this is not always the case. There is normally one head for each surface used on the drive. In the case of a drive using dedicated servo, there will be one more head than there are data surfaces. Since most hard disks have one to four platters, most hard disks have between two and eight heads. Some larger drives can have 20 heads or more.

Only one head can read from or write to the hard disk at a given time. Special circuitry is used to control which head is active at any given time.

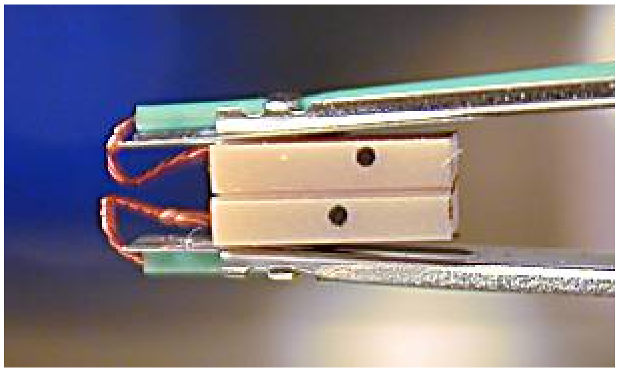
Warning: Most IDE/ATA hard disks come with "setup parameters" intended for use when configuring the disk in the BIOS setup program. Don't be fooled by these numbers, which sometimes bear confusing names like "physical geometry" even though they are not actually the physical geometry at all. For today's drives these numbers have *nothing* to do with what is inside the hard disk itself. Most new IDE/ATA disks these days are set up as having 16 heads, even though all have far fewer, and new drives over 8.4 GB are always specified as having 16 heads per the ATA-4 specification. BIOS translation issues can make this even more confusing. See here for more details.

Next: Floating Height / Flying Height / Head Gap

Floating Height / Flying Height / Head Gap

One distinguishing characteristic of hard disk technology that makes it different from how floppy disks, VCRs and tape decks work, is that the read/write heads do not make contact with the media. The reason for this is that due to the high speed that the hard disk spins, and the need for the heads to frequently scan from side to side to different tracks, allowing the heads to contact the disk would result in unacceptable wear to both the delicate heads and the media. In fact the earliest hard disks did have their heads in contact with the media, and this design was changed due to the wear that contact caused.

Modern drive heads float over the surface of the disk and do all of their work without ever physically touching the platters they are magnetizing. The amount of space between the heads and the platters is called the *floating height* or *flying height*. It is also sometimes called the *head gap*, and some hard disk manufacturers refer to the heads as riding on an "air bearing". The read/write head assemblies are spring-loaded--using the spring steel of the head arms--which causes the sliders to press against the platters when the disk is stationary. (This is done to ensure that the heads don't drift away from the platters; maintaining an exact floating height is essential for correct operation.) When the disk spins up to operating speed, the high speed causes air to flow under the sliders and lift them off the surface of the disk--the same principle of lift that operates on aircraft wings and enables them to fly.



A pair of mated head sliders with their platter removed. You can see that the tension of the head arms has caused them to press against each other.

Due to the very small distance from the heads to the platters--normally measured in millionths of an inch--the hard disk is assembled in a *clean room* containing air specially filtered to remove all but the tiniest particles. Air however is *required* for the heads to function. Whenever someone suggests that the inside of a hard disk is maintained under a vacuum--and it always happens--just ask them how exactly the heads can float on the surface of the disk if there is no air. :^) You will also hear people say that the drive's interior is "sealed" (including, I must admit, myself at one point). This is also generally untrue: while the disk's internal environment is separate from the outside air to keep it clean, air exchange is permitted between the outside and inside of the drive to allow the drive to adjust to changes in air pressure.

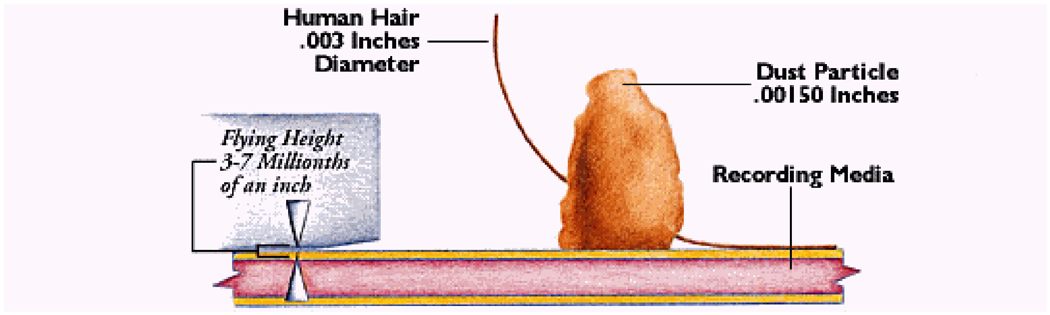
A special "breather" filter is installed to prevent foreign matter from contaminating the drive; see here for more details.

Note: If a drive is used at too high an altitude, the air will become too thin to support the heads at their proper operating height and failure will result; special industrial drives that truly *are* sealed from the outside are made for these special applications.

The distance from the platters to the heads is a specific design parameter that is tightly controlled by the engineers that create the drive. By adjusting the strength of the springs to match the other drive parameters (such as the speed the disks are spinning and the size and shape of the heads) the float height can be precisely maintained. If the height is too great, the heads can't properly read and write the platter. If it is too small, there is increased chance of a head crash (ouch.) As mentioned in the section on operation, increasing areal density means that weaker magnetic fields must be used in storing data on the disks. When this is done the heads must be allowed to ride closer and closer to the platter surface to pick up the weaker signals, which requires other quality improvements to the drive to make sure that there is no chance of a head crash (ouch. :^))

Tip: Some modern drives include sensors that monitor the flying height of the heads and signal a warning if the parameter falls out of the acceptable range.

It's actually quite amazing how close to the surface of the disks the heads fly without touching. To put it into perspective, a modern hard disk has a floating height of an amazing 0.5 microinches. A human hair has a thickness of over 2,000 microinches! You can see why keeping dirt out of the hard disk is so important! In fact, the floating height of a hard disk is smaller than the circuit size of a microprocessor. What's even more amazing is how much abuse these hard disks can take when they are placed in laptop PCs, for example, given these facts, and how many people take this technology for granted every day...



This illustration gives you some idea of just how small the flying height of a modern hard disk is (and today's hard disks have flying heights significantly lower than 3-7 millionths of an inch!

Image © Quantum Corporation

Image used with permission.

When the areal density of a drive is increased to improve capacity and performance, the magnetic fields are made smaller and weaker. To compensate, either the heads must be made more sensitive, or the floating height must be decreased. Each time the floating height is decreased, the mechanical aspects of the disk must be adjusted to make sure that the platters are flatter, the alignment of the platter assembly and the read/write heads is perfect, and there is no dust or dirt on the surface of the platters. Vibration and shock also become more of a concern, and must be compensated for. This is one reason why manufacturers are turning to smaller platters, as well as the use of glass platter substrates. Newer heads such as GMR are preferred because they allow a higher flying height than older, less sensitive heads, all else being equal.

As the flying height of drives continues to decrease, hard disk engineers are recognizing that we may soon reach the point where it cannot be made any smaller without touching the surfaces of the platters. There is actually talk about the possibility of going back to the concept of *contact disks*, where the head gap is intentionally made zero. This would allow even smaller magnetic fields than is possible in today's drives. Of course, this brings us full circle to the first hard disk experiments in the 1950s! The difference of course is almost 50 years of advances in technology. For example, thin film media is much tougher than the oxide media used on contact disks half a century ago, and lubricating agents are much more advanced as well. Even so, it will probably be several years before we know if this technology will be feasible from both an engineering and manufacturing standpoint.

Next: Head Crashes

Head Crashes

Since the read/write heads of a hard disk are floating on a microscopic layer of air above the disk platters themselves, it is possible that the heads can make contact with the media on the hard disk under certain circumstances. Normally, the heads only contact the surface when the drive is either starting up or stopping. Considering that a modern hard disk is turning over 100 times a second, this is not a good thing. :^)

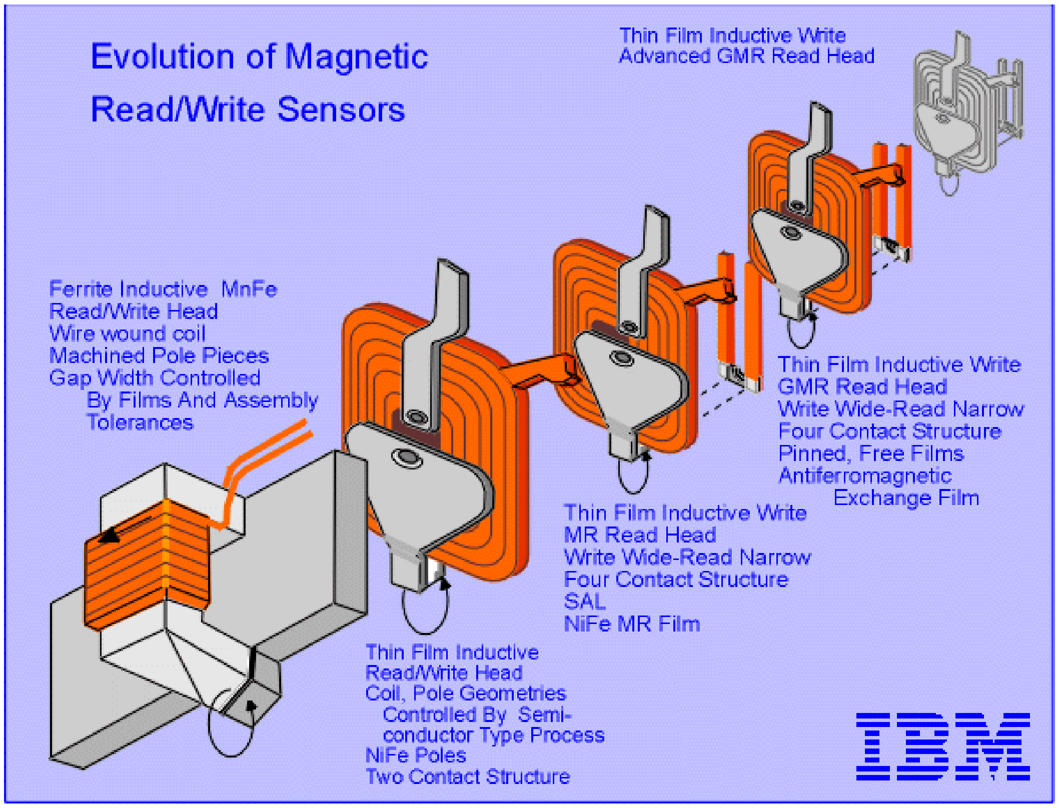
If the heads contact the surface of the disk while it is at operational speed, the result can be loss of data, damage to the heads, damage to the surface of the disk, or all three. This is usually called a *head crash*, two of the most frightening words to any computer user. :^) The most common causes of head crashes are contamination getting stuck in the thin gap between the head and the disk, and shock applied to the hard disk while it is in operation.

Despite the lower floating height of modern hard disks, they are in many ways less susceptible to head crashes than older devices. The reason is the superior design of hard disk enclosures to eliminate contamination, more rigid internal structures and special mounting techniques designed to eliminate vibration and shock. The platters themselves usually have a protective layer on their surface that can tolerate a certain amount of abuse before it becomes a problem. Taking precautions to avoid head crashes, especially not abusing the drive physically, is obviously still common sense. Be especially careful with portable computers; I try to never move the unit while the hard disk is active.

Next: Read/Write Head Technologies

Hard Disk Read/Write Head Technologies

There are several different technology families that have been employed to make hard disk read/write heads. Usually, to enable hard disk speed and capacity to progress to higher levels, adjustments must be made to the way the critical read/write head operation works. In some cases this amounts to minor tweaking of existing technologies, but major leaps forward usually require a breakthrough of some sort, once the existing technologies have been pushed to their limits.



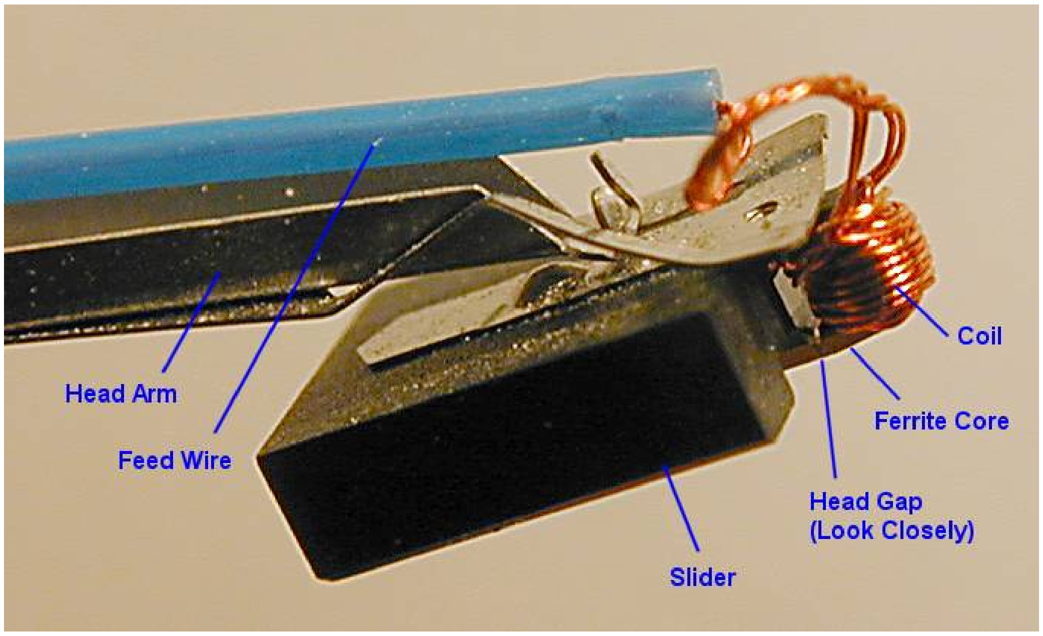
Summary chart showing the basic design characteristics of most of the read/write head designs used in PC hard disks.

Original image © IBM Corporation
Image used with permission.

Next: Ferrite Heads

Ferrite Heads

The oldest head design is also the simplest conceptually. A *ferrite head* is a U-shaped iron core wrapped with electrical windings to create the read/write head--almost a classical electromagnet, but very small. (The name "ferrite" comes from the iron of the core.) The result of this design is much like a child's U-shaped magnet, with each end representing one of the poles, north and south. When writing, the current in the coil creates a polarized magnetic field in the gap between the poles of the core, which magnetizes the surface of the platter where the head is located. When the direction of the current is reversed, the opposite polarity magnetic field is created. For reading, the process is reversed: the head is passed over the magnetic fields and a current of one direction or another is induced in the windings, depending on the polarity of the magnetic field. See here for more general details on how the read/write heads work.



Extreme closeup view of a ferrite read/write head from a mid-1980s Seagate ST-251, one of the most popular drives of its era. The big black object is not actually the head, but the slider. The head is at the end of the slider, wrapped with the coil that magnetizes it for writing, or is magnetized during a read. If you look closely you can actually see the gap in the core, though it is very small. The blue feed wire runs back to the circuits that control the head.

Ferrite heads suffer from being large and cumbersome, which means they must ride at a relatively great distance from the platter and must have reasonably large and strong magnetic fields. Their design prevents their use with modern, very-high-density hard disk media, and they are now obsolete and no longer used. They are typically encountered in PC hard disks under 50

MB in size.

Next: Metal-In-Gap (MIG) Heads

Metal-In-Gap (MIG) Heads

An evolutionary improvement to the standard ferrite head design was the invention of *Metal-In-Gap* heads. These heads are essentially of the same design as ferrite core heads, but add a special metallic alloy in the head. This change greatly increases its magnetization capabilities, allowing MIG heads to be used with higher density media, increasing capacity. While an improvement over ferrite designs, MIG heads themselves have been supplanted by thin film heads and magnetoresistive technologies. They are usually found in PC hard disks of about 50 MB to 100 MB.

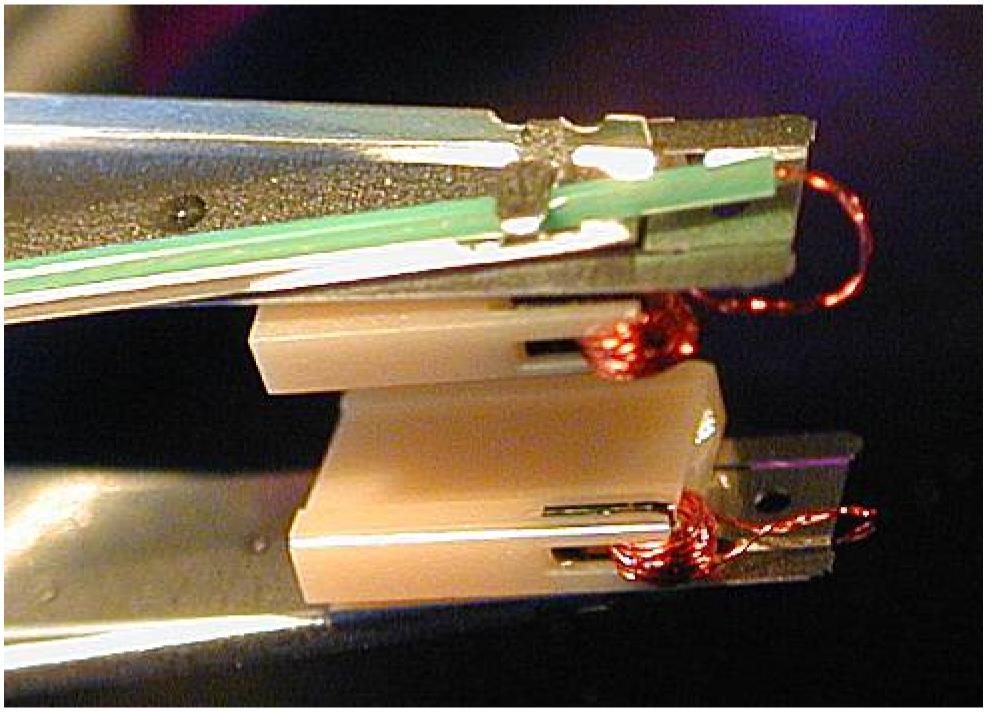
Note: The word "gap" in the name of this technology refers to the gap between the poles of the magnet used in the core of the read/write head, not the gap between the head and the platter.

Next: Thin Film (TF) Heads

Thin Film (TF) Heads

Thin Film (TF) heads--also called *thin film inductive* (TFI)--are a totally different design from ferrite or MIG heads. They are so named because of how they are manufactured. TF heads are made using a photolithographic process similar to how processors are made. This is the same technique used to make modern thin film platter media, which bears the same name; see here for more details on this technology.

In this design, developed during the 1960s but not deployed until 1979, the iron core of earlier heads, large, bulky and imprecise, is done away entirely. A substrate wafer is coated with a very thin layer of alloy material in specific patterns. This produces a very small, precise head whose characteristics can be carefully controlled, and allows the bulky ferrite head design to be completely eliminated. Thin film heads are capable of being used on much higher-density drives and with much smaller floating heights than the older technologies. They were used in many PC hard disk drives in the late 1980s to mid 1990s, usually in the 100 to 1000 MB capacity range.



A pair of mated thin film head assemblies, greatly magnified. The heads are gray slivers with coils wrapped around them, embedded at the end of each slider (large beige objects). One feed line (with green insulation) is visible.

As hard disk areal densities increased, however, thin film heads soon reached their design limits. They were eventually replaced by magnetoresistive (MR) heads. Next: (Anisotropic) Magnetoresistive (MR/AMR) Heads

(Anisotropic) Magnetoresistive (MR/AMR) Heads

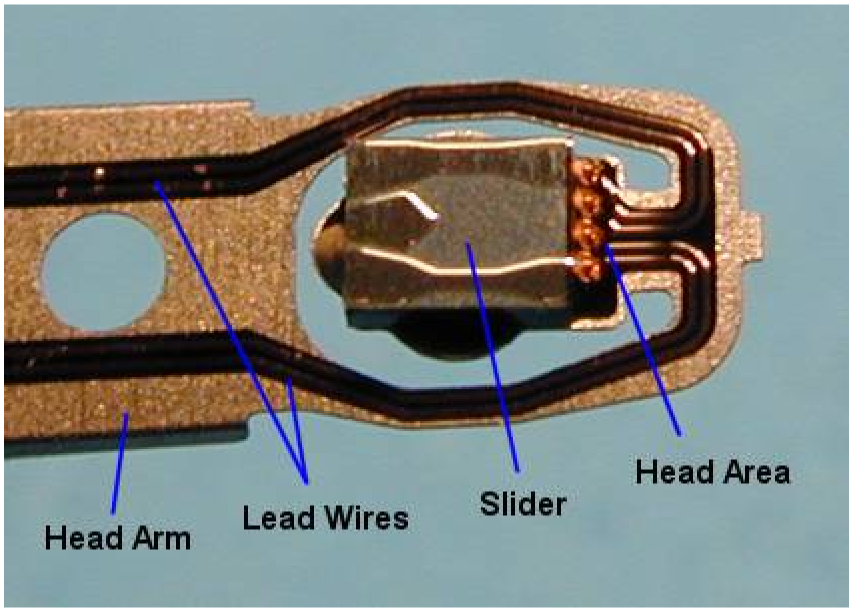
The newest type of technology commonly used in read/write heads is much more of a radical change to the way the read/write head works internally than the earlier advances, which were much more evolutionary and more related to how the head was made than how it worked. While conventional ferrite or thin film heads work on the basis of inducing a current in the wire of the read head in the presence of a magnetic field, *magnetoresistive* (*MR*) heads use a different principle entirely to read the disk.

Note: The correct technical name for first-generation MR heads is *anisotropic magnetoresistive* (AMR), but traditionally they have just been called "magnetoresistive" (MR). With GMR heads now on the market, there is the potential for confusion between the terms "magnetoresistive" and "giant magnetoresistive". Therefore, some companies have now gone back to calling the older MR heads "AMR" heads to distinguish them from GMR ones. Normally though, if you are told a drive has "MR heads", this means the older technology described here.

An MR head employs a special conductive material that changes its resistance in the presence of a magnetic field. As the head passes over the surface of the disk, this material changes resistance as the magnetic fields change corresponding to the stored patterns on the disk. A sensor is used to detect these changes in resistance, which allows the bits on the platter to be read.

The use of MR heads allows much higher areal densities to be used on the platters than is possible with older designs, greatly increasing the storage capacity and (to a lesser extent) the speed of the drive. Because the MR head is not generating a current directly the way standard heads do, it is several times more sensitive to magnetic flux changes in the media. This allows the use of weaker written signals, which lets the bits be spaced closer together without interfering with each other, improving capacity by a large amount.

MR technology is used for reading the disk only. For writing, a separate standard thin-film head is used. This splitting of chores into one head for reading and another for writing has additional advantages. Traditional heads that do both reading and writing are an exercise in tradeoffs, because many of the improvements that would make the head read more efficiently would make it write less efficiently, and vice-versa. For example, if you increase the number of windings of wire around the core of a standard read/write head, you increase the sensitivity of the head when reading, but you make it much more difficult to write at high speed. Also, for best results we want to write a wider data track (to ensure the media is properly magnetized) but read a narrower one (to make sure we don't accidentally pick up signals from adjacent bits). In an MR design the MR head does the reading, so the thin film write head can be optimized solely for writing without worrying about these sorts of compromises.



Closeup view of an MR head assembly. Note that the separate copper lead wire of older head designs is gone, replaced by thin circuit-board-like traces. The slider is smaller and has a distinctive shape. The actual head is too small to be seen without a microscope. If you look at the page discussing GMR heads you will see a diagram showing in detail where on the assembly the head is found.

First introduced in 1991 by IBM--who else--but not used widely until several years later, MR heads were one of the key inventions that led to the creation of hard disks over 1 GB in size, and the subsequent explosive growth in size since then. Despite the increased cost of MR heads, they have now totally replaced thin film heads, which just are not up to the challenge of hard disks in the tens of gigabytes. MR heads are commonly found in hard disks from about 1 GB to about 30 GB in size.

Even MR heads however have a limit in terms of how much areal density they can handle. Successive generations of MR heads were reduced in size to allow still greater areal density. Sometimes these more advanced designs were dubbed *MRX* for *Extended Magnetoresistive* heads. The successor to MR now appears to be GMR heads, named for the *giant magnetoresistive effect*. They are similar in basic concept to MR heads but are more advanced; GMR heads are discussed here.

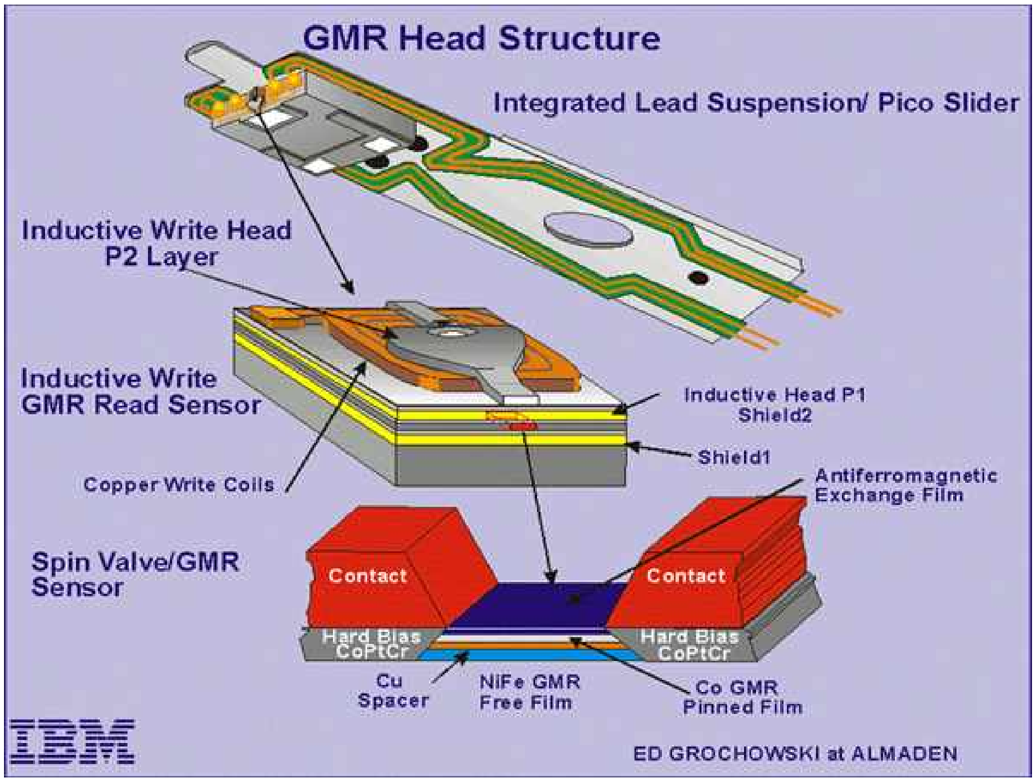
Next: Giant Magnetoresistive (GMR) Heads

Giant Magnetoresistive (GMR) Heads

From the beginning in the 1950s and its first hard disk drive, IBM has been at the forefront of hard disk technological advances. IBM's groundbreaking research and development efforts continue, nearly 50 years later, with the introduction of *giant magnetoresistive* (GMR) hard disk read heads. These heads work on the same general principles as the original (anisotropic) magnetoresistive heads, but use a somewhat different design that makes them superior in several ways.

GMR heads are not named "giant" because of their size; they are actually smaller than the regular (A)MR heads developed by IBM many years earlier. Rather, they are named after the *giant magnetoresistive effect*, first discovered in the late 1980s by two European researchers, Peter Gruenberg and Albert Fert, who were working independently. Working with large magnetic fields and thin layers of various magnetic materials, they noticed very large resistance changes when these materials were subjected to magnetic fields. These early experiments used techniques and materials that were not suitable to manufacturing, but they formed the basis for the technology.

From there, engineers and scientists at IBM's Almaden Research Center took over. IBM developed GMR into a commercial product by experimenting with thousands of different materials and methods. A key advance was the discovery that the GMR effect would work on multilayers of materials deposited by sputtering, the same technique used to make thin film media and thin film read/write heads. By December 1997, IBM had introduced its first hard disk product using GMR heads, and areal density records have been looking over their shoulders nervously ever since.



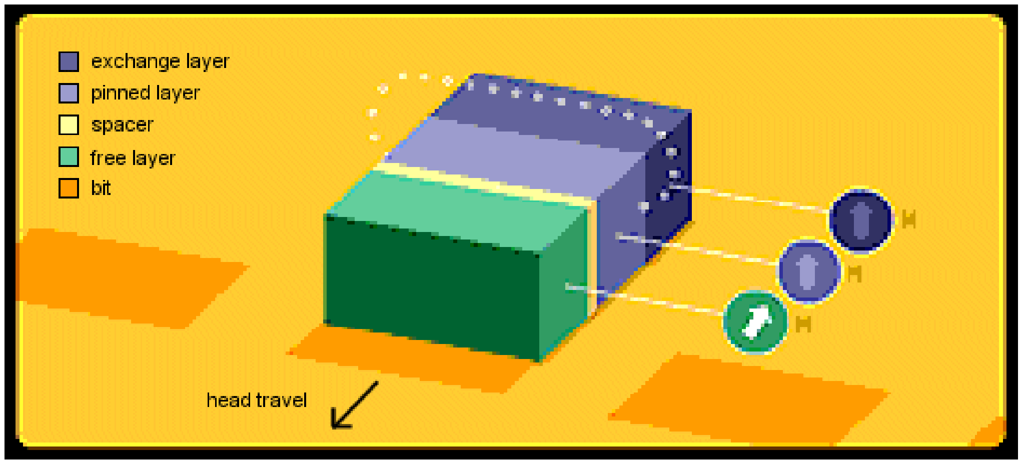
Detailed structure diagram of a GMR head assembly. Notice how small these devices are: the scale is magnified twice to get to the detail level of the GMR sensor itself, and the arm/slider/head structure at top is itself only about a quarter of an inch long (see the illustration on the page covering MR heads to see what one of those assemblies looks like in real life.)

Original image © IBM Corporation
Image used with permission.

GMR heads are comprised of four layers of thin material sandwiched together into a single structure:

1. **Free Layer:** This is the sensing layer, made of a nickel-iron alloy, and is passed over the surface of the data bits to be read. As its name implies, it is free to rotate in response to the magnetic patterns on the disk.
2. **Spacer:** This layer is nonmagnetic, typically made from copper, and is placed between the free and pinned layers to separate them magnetically.
3. **Pinned Layer:** This layer of cobalt material is held in a fixed magnetic orientation by virtue of its adjacency to the exchange layer.
4. **Exchange Layer:** This layer is made of an "antiferromagnetic" material, typically constructed from iron and manganese, and fixes the pinned layer's magnetic orientation.

Here's how the GMR head works in a nutshell (without getting into quantum physics that would make your brain melt--and mine. :^)) When the head passes over a magnetic field of one polarity (say, a "0" on the disk), the free layer has its electrons turn to be aligned with those of the pinned layer; this creates a lower resistance in the entire head structure. When the head passes over a magnetic field of the opposite polarity ("1"), the electrons in the free layer rotate so that they are not aligned with those of the pinned layer. This causes an increase in the resistance of the overall structure. The resistance changes are caused by changes to the spin characteristics of electrons in the free layer, and for this reason, IBM has named these structures *spin valves*. If you imagine a plumbing pipe with a rotatable shut-off valve, that's the general concept behind the name.



Conceptual operation of a GMR head, showing the four layers. As the head moves across a bit, electrons in the free layer rotate, increasing the resistance of the overall structure. See the end of the page for a URL where you can view this illustration animated.

Original image © IBM Corporation
Image used with permission.

GMR heads are superior to conventional MR heads because they are more sensitive. While older MR heads typically exhibit a resistance change when passing from one magnetic polarity to another of about 2%, for GMR heads this is anywhere from 5% to 8%. This means GMR heads can detect much weaker and smaller signals, which is the key to increasing areal density, and thus capacity and performance. They are also much less subject to noise and interference because of their increased sensitivity, and they can be made smaller and lighter than MR heads. GMR heads are also typically fitted with "shields" that prevent them from being affected by stray magnetic fields, i.e., anything but the bit directly below the head.

GMR heads are used in the latest technology drives, which currently have capacities of up to 75 GB and areal densities of approximately 10 to 15 Gbits/in^2. As of early 2000, IBM has already produced GMR heads in the lab capable of 35 Gbits/in^2, suggesting that standard form factor hard disks of 200 GB and up are just around the corner.

Tip: 99% of the Java and Shockwave stuff on the Web is fluffy, superfluous nonsense, but *this page of IBM Research GMR head animated demos* proves that animation on the WWW can be educational, and not just annoying.

Next: Colossal Magnetoresistive (CMR) Heads

Colossal Magnetoresistive (CMR) Heads

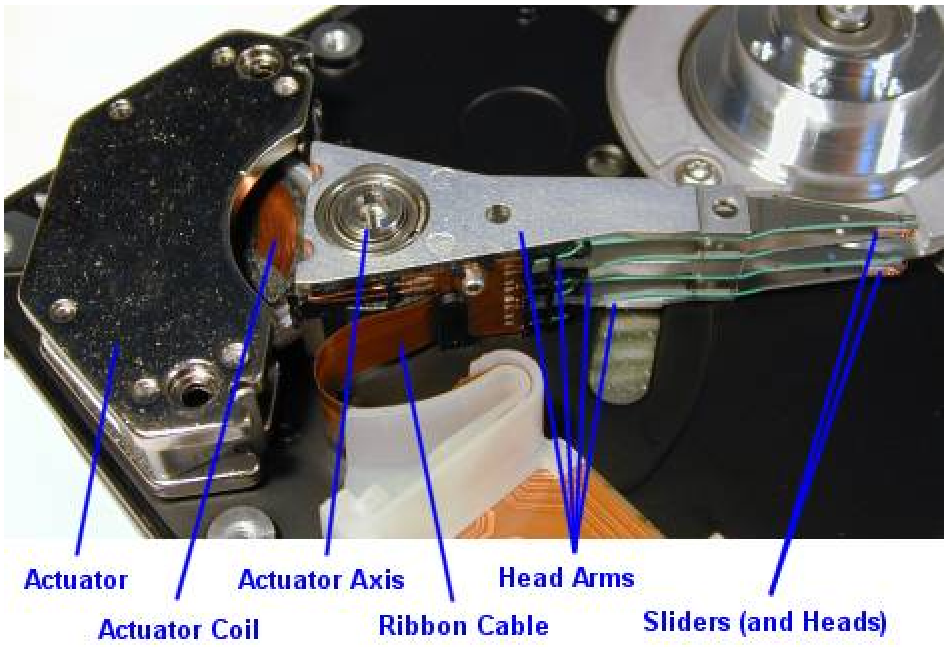
Now that new giant magnetoresistive (GMR) heads are taking over the market, engineers have turned their attentions to the next breakthrough that will take hard disk read heads to the next level of performance. One of the designs being researched is an evolutionary advance based on GMR that has been dubbed by some as *colossal magnetoresistive* or *CMR*. At this stage this technology is in the early stages of investigation, and I have not been able to find much information on it. It would be logical to assume that it involves using slightly different materials than GMR in an effort to further increase sensitivity. When I find more information on CMR I'll add it here.

Next: Hard Disk Head Sliders, Arms and Actuator

Hard Disk Head Sliders, Arms and Actuator

The hard disk platters are accessed for read and write operations using the read/write heads mounted on the top and bottom surfaces of each platter. Obviously, the read/write heads don't just float in space; they must be held in an exact position relative to the surfaces they are reading, and furthermore, they must be moved from track to track to allow access to the entire surface of the disk. The heads are mounted onto a structure that facilitates this process. Often called the *head assembly* or *actuator assembly* (or even the *head-actuator assembly*), it is comprised of several different parts.

The heads themselves are mounted on *head sliders*. The sliders are suspended over the surface of the disk at the ends of the *head arms*. The head arms are all mechanically fused into a single structure that is moved around the surface of the disk by the *actuator*. (Sort of like "the leg bone's connected to the knee bone", isn't it? :^)) It would be an understatement to say that these components are neglected; heck, most people don't even know they *exist*! Yet they play an important role in the function and performance of the drive. In particular, advances in slider, arm and actuator design are critical to improving the seek time of a hard disk; the impact that the actuator has on performance is discussed in this section.



Annotated illustration of a typical PC actuator assembly, showing the major components. The platters have been removed from the drive to provide a better
view of the actuator arms and heads. There are four sliders but only one of each pair is visible. The spindle motor is visible at the top right.

This section discusses the sliders, arms and actuator of the modern disk drive, as well as explaining the operation of disk drive *servo mechanisms* and related technologies.

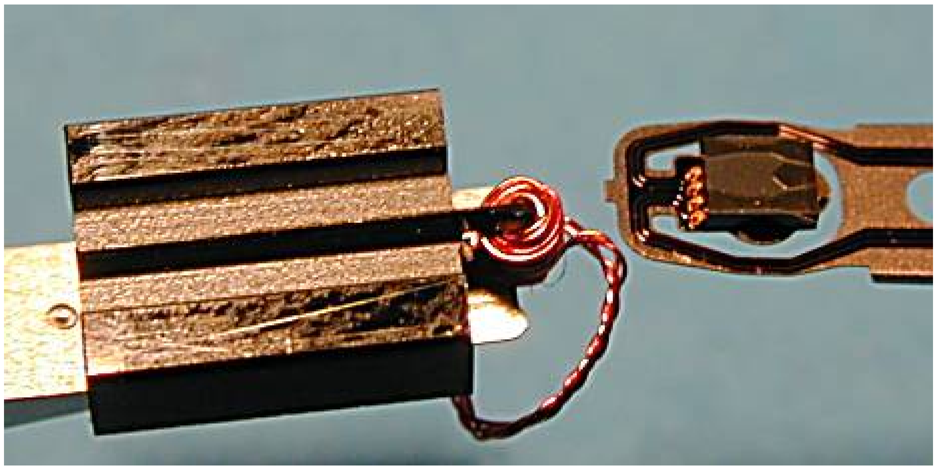
Next: Head Sliders

Head Sliders

Hard disk read/write heads are too small to be used without attaching them to a larger unit. This is especially true of modern hard disk heads. Each hard disk head is therefore mounted to a special device called a *head slider* or just *slider* for short. The function of the slider is to physically support the head and hold it in the correct position relative to the platter as the head floats over its surface.

Sliders are given a special shape to allow them to ride precisely over the platter. Usually they are shaped somewhat like a sled; there are two rails or runners on the outside that support the slider at the correct flying height over the surface of the disk, and in the middle the read/write head itself is mounted, possibly on another rail.

As hard disk read/write heads have been shrinking in size, so have the sliders that carry them. The main advantage of using small sliders is that it reduces the weight that must be yanked around the surface of the platters, improving both positioning speed and accuracy. Smaller sliders also have less surface area to potentially contact the surface of the disk.



A graphic illustration of what approximately 15 years' worth of technological evolution has done to hard disk head sliders. At left, a slider from a 40 MB 5.25" ferrite-head drive; at right, the slider from a 3.2 GB, 3.5" MR-head drive.

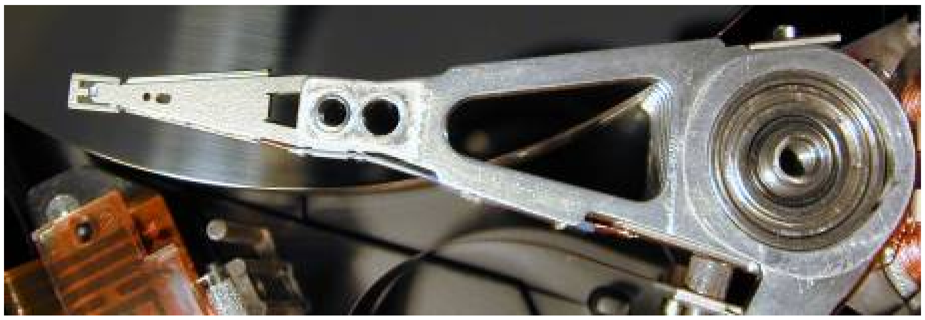
The hard disk industry has given names to various generations of slider technology. When the design was first reduced to 50% of the size of the first hard disk sliders, someone decided to call it the *nano-slider*, where "nano" is the prefix denoting "one billionth". Of course the name is silly, since the sliders have not shrunk by even a factor of 10. The newest sliders have been shrunk from the "nano" size by about another 50% and are being called *pico-sliders*, which in this author's opinion is an equally silly name, and for the same reason. :^)

Each slider is mounted onto a head arm to allow it to be moved over the surface of the platter to which it is mated.

Next: Head Arms

Head Arms

The head arms are thin pieces of metal, usually triangular in shape onto which the head sliders (carrying the read/write heads) are mounted. In a way, the idea here is similar to how the arm of a phonograph is used to move the stylus from the outside of a record to the inside (although of course the similarity ends there). There is one arm per read/write head, and all of them are lined up and mounted to the head actuator to form a single unit. This means that when the actuator moves, all of the heads move together in a synchronized fashion. Heads cannot be individually sent to different track numbers.


The top head arm of a typical recent-design hard disk drive. Note that the arm is not solid, but rather has a structural triangular shape. This is done to reduce weight while maintaining rigidity.

The arms themselves are made of a lightweight, thin material, to allow them to be moved rapidly from the inner to outer parts of the drive. Newer designs have replaced solid arms with structural shapes in order to reduce weight and improve performance. This is the same technique used to reduce weight in the construction of airplane wings, for example. Newer drives achieve faster seek times in part by using faster and smarter actuators and lighter, more rigid head arms, allowing the time to switch between tracks to be reduced.

A recent trend in the hard disk industry has been the reduction in the number of platters in various drive families. Even some "flagship" drives in various families now only have three or even two platters, where four or five was commonplace a year or so ago. One reason for this trend is that having a large number of head arms makes it difficult to make the drive with high enough precision to permit very fast positioning (on random seeks). This is due to increased weight in the actuator assembly from the extra arms, and also problems aligning all the heads. So in essence, this is a tradeoff that some drive manufacturers are making to improve performance at the expense of capacity. With drive densities now at 20 GB per platter and bound to increase, this is an acceptable design decision for most buyers.

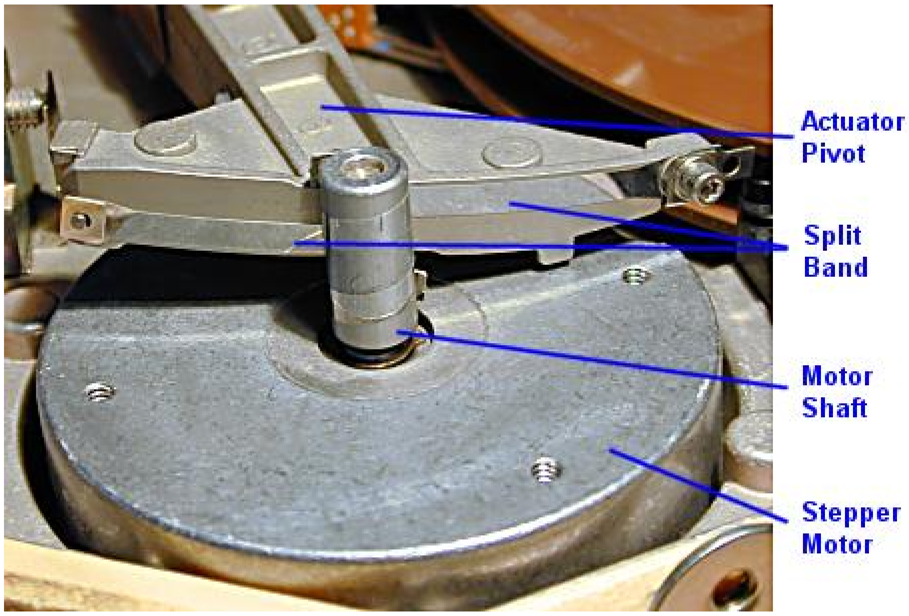
Next: Head Actuator

Head Actuator

The actuator is the device used to position the head arms to different tracks on the surface of the platter (actually, to different cylinders, since all head arms are moved as a synchronous unit, so each arm moves to the same track number of its respective surface). The actuator is a very important part of the hard disk, because changing from track to track is the only operation on the hard disk that requires active movement: changing heads is an electronic function, and changing sectors involves waiting for the right sector number to spin around and come under the head (passive movement). Changing tracks means the heads must be shifted, and so making sure this movement can be done quickly and accurately is of paramount importance. This is especially so because physical motion is so *slow* compared to anything electronic--typically a factor of 1,000 times slower or more.

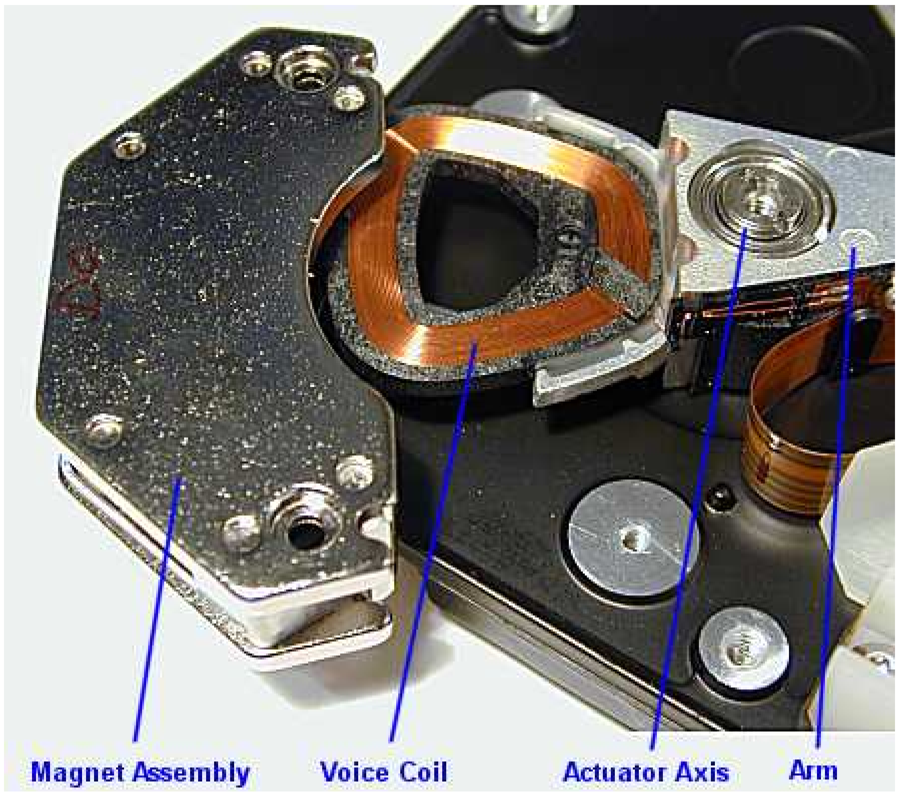
Head actuators come in two general varieties:

- **Stepper Motors:** Originally, hard disk drives used a *stepper motor* to control the movement of the heads over the surface of the platters. A regular motor turns in a rotary fashion continuously; it can stop at any point in its rotation as it spins around, kind of like the second hand on a wind-up wristwatch. A stepper motor can only stop at predefined "steps" as it turns around, much the way the second hand turns on an electronic, quartz wristwatch. A hard drive using a stepper motor for an actuator attaches the arms to the motor, and each time the motor steps one position clockwise or counterclockwise, the arms move in or out one position. Each position defines a track on the surface of the disk. Stepper motors are also commonly used for both turning the spindle and positioning the head on floppy disk drives. If you have a floppy drive, find one of its motors and turn it slowly with your hand; you will feel the discrete step-wise nature of its motion.



A stepper motor actuator. The motor moves in steps, which you can feel if you move the motor shaft by hand. The shaft has two thin strips of metal wrapped around it, which are connected to a pivot that is rigidly attached to the actuator arms. As the motor shaft turns, one half of this "split band" coils onto the shaft and the other half uncoils. When the motor turns in the opposite direction the process reverses. As this occurs the pivot moves and in doing so, moves the actuator arms and the hard disk heads.

- **Voice Coils:** The actuator in a modern hard disk uses a device called a *voice coil* to move the head arms in and out over the surface of the platters, and a closed-loop feedback system called a *servo system* to dynamically position the heads directly over the data tracks. The voice coil works using electromagnetic attraction and repulsion. A coil is wrapped around a metal protrusion on the end of the set of head arms. This is mounted within an assembly containing a strong permanent magnet. When current is fed to the coil, an electromagnetic field is generated that causes the heads to move in one direction or the other based on attraction or repulsion relative to the permanent magnet. By controlling the current, the heads can be told to move in or out much more precisely than using a stepper motor. The name "voice coil" comes from the resemblance of this technology to that used to drive audio speakers, which are also basically electromagnets. All PC hard disk voice coil actuators are *rotary*, meaning that the actuator changes position by rotating on an axis.



A partially-disassembled voice coil actuator. The magnet assembly has been unscrewed from its mounting and pulled to the left to expose the coil. The magnet assembly consists of two metal plates (top one easily visible above, and part of the bottom one visible.) The magnet itself is mounted on the underside of the top plate, and spacers used between the plates to create the gap for the coil assembly. Being non-ferrous the coil moves freely between the plates, rotating the actuator on its axis as its magnetic polarity is changed. (Incidentally, the magnet is strong enough that after removing the spacers between the plates, the bottom plate got "stuck" on the magnet and required considerable effort to remove!)

The primary distinction between the two designs is that the stepper motor is an *absolute* positioning system, while the voice coil is a *relative* positioning system. Commands given to a stepper motor actuator are generally of the form "Go in this direction to position A, where you'll find item B". Commands to a voice coil actuator are of the form "Go in this direction until you find item B". Consider this analogy. In your backyard you have buried a "secret treasure" and want to tell a friend where to find it. When you buried it, you walked down a path 139 paces to the fourth oak tree, and buried it at the edge of the path. The stepper motor analog would be to tell your friend to walk 139 paces down the path, and start digging. The voice coil analog would be to tell him to look for the fourth oak tree and dig there. Obviously, using the "139 paces" method, your friend has a problem: his paces aren't likely to be the same length as yours. In fact, even if you yourself walked out 139 paces twice, you'd probably end up in very different spots, since a "pace" isn't an accurate or repeatable measure. On the other hand, the fourth oak tree will always be the fourth oak tree (barring disastrous chain-saw activity :^)).

Now hard disks of course don't have to use inaccurate measures like "paces", and it's always the same stepper motor accessing the disk, not a "friend", so why is saying "track #139" a big problem? For starters, motors change their characteristics over time, and after a year or two position #139 might not be where it was when the drive was first formatted. However, they have an even more serious problem: disk components (the platters and the head arms themselves especially) expand and contract with heat. Even if a stepper motor was perfect, it could not properly account for the fact that the disks are changing in size, and therefore, the tracks are literally moving around. If you consider our backyard analogy and think about what it would be like if the oak tree moved a few feet closer to or further from the house based on the day's temperature, you start to realize how inadequate absolute positioning of this form can be.

A stepper motor has no way to compensate for expansion or contraction of the disk: all it can do is go to where "track #139" is supposed to be, and hope it finds it there! If it doesn't find it because the motor and the disk have become out of sync, errors and data loss result. This is why older disks were so sensitive to temperature, and normally had to be low-level formatted periodically to make sure the tracks lined up with the heads properly. This is also why many drives would fail when first powered up after a weekend, but would work properly after the drive had been allowed to warm up.

The shortcomings of stepper motors were unfortunate but acceptable with old hard disks, because of their relatively low track density. To compensate, tracks could be written fairly wide so that the head would find them even if it was a bit misaligned. The first PC hard disks in 1982 had a track density of only two or three hundred tracks per inch (TPI). Even In 1986, the year Conner Peripherals introduced the first voice coil PC hard disk, density had increased to only about 1,000 TPI. Stepper motors are still used to drive floppy disks, for example, because the accuracy demands for floppies are much lower: a 1.44 MB floppy disk has a track density of 135 tracks per inch. In contrast, today's hard disks have densities as high as *30,000* tracks per inch. With data packed this densely, tracks are extremely thin, and a stepper motor lacks the accuracy and stability necessary for proper operation.

All modern hard disks use voice coil actuators. The voice coil actuator is not only far more adaptable and insensitive to thermal issues, it is much faster and more reliable than a stepper motor. The actuator's positioning is dynamic and is based on feedback from examining the actual position of the tracks. This closed-loop feedback system is also sometimes called a *servo motor* or *servo positioning system* and is commonly used in thousands of different applications where precise positioning is important. There are several different ways that the servo positioning system is implemented in PCs; the servo's operation is discussed in its own section.

Next: Servo Techniques and Operation

Servo Techniques and Operation

Modern hard disks use voice coil actuators to position the heads on the surface of the hard disk's platters. This actuator is one instance of what is commonly called a *servo system*, which is a type of *closed-loop feedback system*. In this sort of positioning system, a device is controlled by doing something, measuring the result, seeing how far off the device is from its target, making an adjustment, and repeating. This enables the device to reach its target intelligently instead of just taking a guess and hoping it is correct.

One example of a closed-loop feedback system is a modern heating system that uses a thermostat: when the temperature gets too low relative to the "target", the heat turns on and stays on until the target temperature is reached. Another example is a driver steering through a curve in the road: when the car starts to veer off course, the driver turns the wheel and looks to see if the turn was enough to keep the card on the road. If not, he or she turns the wheel more. When the curve straightens out, the steering wheel is returned to the normal position. The *feedback* is what makes the control system "closed-loop". In contrast a "send it and hope it finds what is supposed to be there" system such as a stepper motor actuator is called an *open-loop system*.

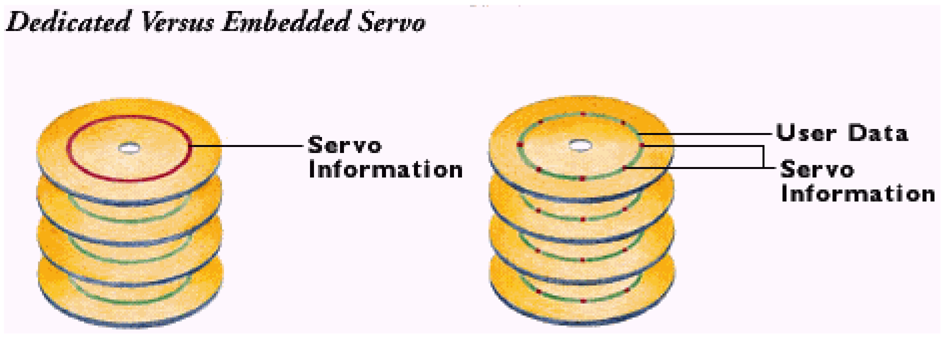
A key element of any closed-loop feedback system is a measuring device, to provide the feedback. In the case of the thermostat it is a thermometer, and for the driver, it is his or her eyes viewing the road. For the hard disk, the feedback device is the read/write head itself, and special codes written on the disk that let the hard disk know where the heads are when the actuator moves. These codes are, unsurprisingly, typically called *servo codes*. They are read by the heads and fed back to the actuator control logic (at very high speed of course) to guide the actuator to the correct track. By putting different codes on each track of the disk, the actuator can always figure out which track it is looking at.

There are three different ways that the hard disk servo mechanism has been implemented. Each uses a different way of recording and reading the servo information from the disk:

- **Wedge Servo:** In this implementation used in older drives, the servo information is recorded in a "wedge" of each platter; sort of like a "slice" out of a pie. The remainder of the "pie" contains data. This design has an important flaw: the servo information is only in one location on the hard disk, which means that to position the heads a lot of waiting must be done for the servo wedge to rotate around to where the heads are. All this waiting makes the positioning performance of drives that use this method painfully slow. Obsolete, this technique is no longer used.
- **Dedicated Servo:** In this technique, an entire surface of one disk platter is "dedicated" *just* for servo information, and no servo information is recorded on the other surfaces. One head is constantly reading servo information, allowing very fast servo feedback, and eliminating the delays associated with wedge servo designs.

Unfortunately, an entire surface of the disk is "wasted" because it can contain no data. Also, there is another problem: the heads where data is recorded may not always line up exactly with the head that is reading the servo information, so adjustments must be made to compensate, and since the servo platter may be warmer or cooler than the data platters, these drives are notorious for needing frequent thermal recalibration. Because one platter surface is used for servo information and not data, dedicated servo drives usually have an odd number of heads (though there are also marketing reasons why this can happen.) They were found in many drives through the mid-1990s.

- **Embedded Servo:** The newest servo technique intersperses servo information with data across the entire surface of all of the hard disk platter surfaces. The servo information and data are read by the same heads, and the heads never have to wait for the disk to rotate the servo information into place as with wedge servo. This method doesn't provide the constant access to positioning information that is available with dedicated servo, but it also doesn't require an entire surface to be expended on overhead. Also, the need for constant thermal recalibration is greatly reduced since the the servo information and data are the same distance from the center of the disk and will expand or contract together. All modern hard disks use embedded servo.



A simple illustration of the difference between dedicated servo and embedded servo. On the left, dedicated servo: one platter surface contains nothing but servo information, and the others nothing but data. On the right, embedded servo, with data and servo information together. (Note that for clarity only one track on each platter (one cylinder) is shown in this illustration; in fact every track of the servo surface has servo information in the dedicated servo design, and every track of *every* surface has interspersed servo information in the embedded design.

Image © Quantum Corporation
Image used with permission.

The servo codes are written to the disk surfaces at the time the hard disk is manufactured. Special, complex and expensive equipment is employed to record this information, which as you can imagine must be placed very precisely on each surface. The machines that do this are called ... wait for it... *servowriters*. :^) The servo codes are put in place for the life of the drive and cannot be rewritten without returning the drive to the factory (which never happens because it would be *way* too expensive). The hard disk heads themselves are locked out at the hardware level by the drive's controller from writing to the areas where servo information is written. The creation of this precise pre-written information is part of the low-level formatting of a modern drive, and the need for the fancy machine is one reason why modern disks cannot be low-level-formatted outside the factory. There is nothing a user can do with a drive to touch the servo information (well, short of using a screwdriver, which is *not* recommended... :^))

Next: Thermal Recalibration

Thermal Recalibration

All modern drives use voice coil actuators instead of the older stepper motors, which makes them far less sensitive to thermal effects than older hard disks were. The built in servo mechanism automatically adjusts for shifts in the position of the media due to temperature variations. However, even the newest drives have issues related to thermal stability. Since engineers continue to decrease the width of each track of data, and increase the number of tracks per inch on the surface of the disk, even the highest-quality electronic servo motors can have, well, "issues" when the various pieces of metal in the hard disk start expanding and contracting at different rates. This is especially the case with drives using *dedicated servo*, because the servo information and data are on different physical surfaces. Since the surfaces can easily be at different temperatures and can therefore expand or contract at different rates, there is the potential that the servo data and user data might become misaligned.

To combat this problem, most drives manufactured in the mid-1990s include a feature called *thermal recalibration*. Every few minutes, the heads are moved and the distance between tracks measured. This information is recorded in the drive's memory and used to aid in positioning the heads when reading or writing needs to be done. When the recalibration occurs you can hear the disk operate as if you were reading or writing to it, even if you are not.

Thermal recalibration produces one unfortunate side-effect: if you attempt to access the disk while a recalibration cycle is taking effect, there is a slight pause until it completes. This does not cause any read or write requests to be lost, but it can cause a problem if you are performing a read or write task that is operating in "real time" and might be sensitive to the delay. Common examples include real-time video playback, audio file editing, or burning a recordable CD-ROM disk. For users working with these applications, thermal recalibration represents an unacceptable problem, so the hard disk manufacturers created special drives that work around the recalibration "feature". Typically by using special buffering techniques or by intelligently "scheduling" recalibration activity to avoid platters being accessed by the user, these drives essentially "hide" recalibration from the PC operator. They were frequently marketed as being "audio/visual" or "A/V" drives during the 1990s.

Today, recalibration has become largely a moot point. The need for constant recalibration was greatly diminished with the creation of embedded servo technology, which is now the standard for hard disk drives. Recalibration had become a real "black eye" of sorts to hard drive manufacturers over the years, so they were happy to announce to the world that they had "eliminated" thermal recalibration in their newest units. This is sort of true and not true; it would be more accurate to say that it has been greatly reduced. Some degree of recalibration is still required even with the newest drives, but it does not cause the potential impact on performance that users of the old style drives had to deal with.

Tip: You may still find companies trying to sell "special A/V drives" at higher prices than regular drives. These days, this is more likely to be an attempt to get you to part with a greater percentage of your wallet than anything else. It's not necessary to look for such designations on modern drives.

Next: Head Parking and the Landing Zone

Head Parking and the Landing Zone

When the platters are not spinning, the heads rest on the surface of the disk. When the platters spin up, the heads rub along the surface of the platters until sufficient speed is gained for them to "lift off" and float on their cushion of air. When the drive is spun down, the process is repeated in reverse. In each case, for a period of time the heads make contact with the surface of the disk--while in motion, in fact.

While the platters and heads are designed with the knowledge in mind that this contact will occur, it still makes sense to avoid having this happen over an area of disk where there is data! For this reason, most disks set aside a special track that is designated to be where the heads will be placed for takeoffs and landings. Appropriately, this area is called the *landing zone*, and no data is placed there. The process of moving the heads to this designated area is called *head parking*.

Most early hard drives that used stepper motors did not automatically park the heads of the drive. As a safety precaution, small utilities were written that the user would run before shutting down the PC. The utility would instruct the disk to move the heads to the landing zone, and then the PC could be shut off safely. A parameter in the BIOS setup for the hard disk told the system which track was the landing zone for the particular model of hard disk. Usually, it was the next consecutive-numbered track above the largest-numbered one actually used for data.

Modern voice-coil actuated hard disk drives are all auto-parking. On some disks, a weak spring is attached to the head assembly that tries to pull the heads to the landing zone. When power is applied the actuator is able to overpower the spring and position the heads normally. When the power is shut off, the electromagnetic force from the voice coil abates, and the spring yanks the heads to the landing zone before the platters can spin down; this can sometimes be heard on older drives as an audible *clunk* when you turn the power off. Other disks use a different mechanical or electronic scheme to achieve the same goal. Some even make use of the rotational energy remaining in the spindle motor to move the heads off the data surface when the power is cut off! This means that modern hard disks will automatically park their heads--even in the event of a power failure--and no utilities are required. The BIOS landing zone parameter for modern drives is ignored.

Some people still think that it is necessary to manually park the heads of modern hard disks, but this is not true. I sometimes think of head parking utilities as the disk drive's equivalent of a screen saver. In both cases, the software was invented as a preventative measure, and one that made sense for use with the technology that prevailed at the time it was thought up. And in both cases, the technology has evolved to the point where utility is no longer necessary, yet many people still think it is. :^)

IBM has developed an alternative to conventional head parking that I think is really a great idea. Instead of letting the heads fall down to the surface of the disk when the disk's motor is stopped, the heads are lifted completely off the surface of the disk while the drive is still spinning, using a special ramp. Only then are the disks allowed to spin down. When the power is reapplied to the spindle motor, the process is reversed: the disks spin up, and once they are going fast enough to let the heads fly without contacting the disk surface, the heads are moved off the "ramp" and back onto the surface of the platters. IBM calls this *load/unload technology*. In theory it should improve the reliability of the hard disk as a whole, by eliminating most contact between the heads and platters entirely. I am unaware of any other drive manufacturers using it at this time. You can read more about it here.

Another feature related to reducing damage to the hard disks caused by wear from the heads is *wear leveling*, which moves the heads over the surface of the drive to avoid "wearing out" one section of the drive. It is discussed in this quality and reliability section.

Next: Single vs. Multiple Actuators

Single vs. Multiple Actuators

One common question that folks studying hard drives often have is something like the following: "If the mechanical motion of the heads across the surface of the hard disk platters is so slow--relative to other components in the hard drive and the rest of the computer--why don't the hard disk manufacturers just make hard disks with more than one actuator?" It's a very good question. Putting a second set of heads in the drive would allow two areas of the disk to be accessed simultaneously, greatly improving performance, particularly on random seeks. Sounds great... in fact, why not put four actuators and sets of heads on the drive, while we're at it? :^)

In fact, such hard disks *have* been built. Conner Peripherals, which was an innovator in the hard disk field in the late 1980s and early 1990s (they later went bankrupt and their product line and technology were purchased by Seagate) had a drive model called the *Chinook* that had two complete head-actuator assemblies: two sets of heads, sliders and arms and two actuators. They also duplicated the control circuitry to allow them to run independently. For its time, this drive was a great performer. But the drive never gained wide acceptance, and the design was dropped. Nobody to my knowledge has tried to repeat the experiment in the last several years.

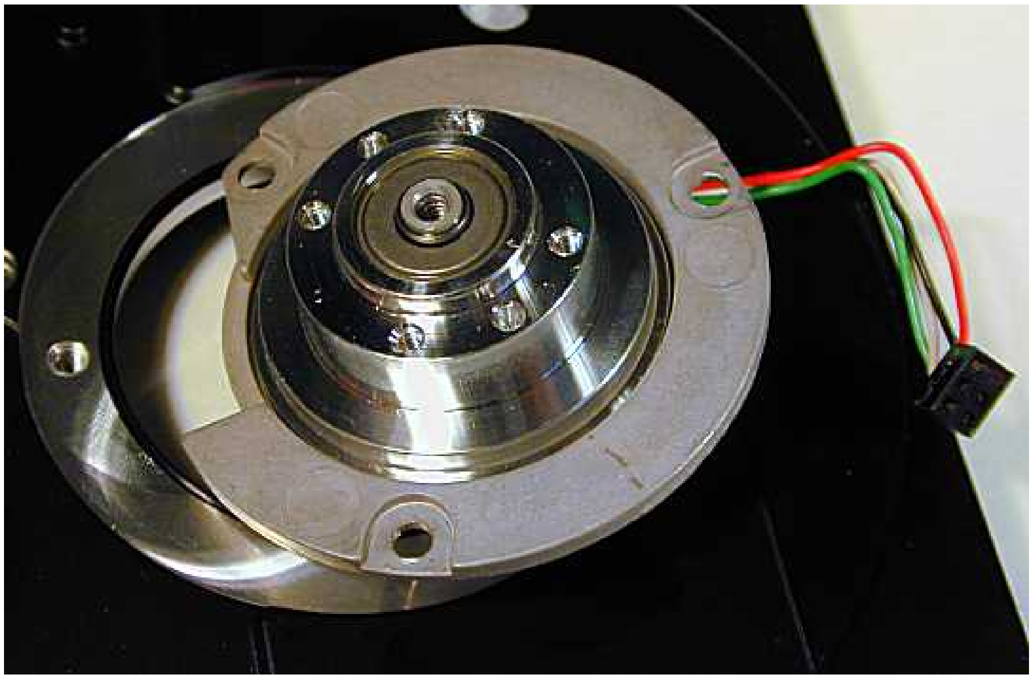
There are several reasons why it is not practical to make a drive with more than one actuator. Some are technical; for starters, it is very difficult to engineer. Having multiple arms moving around on a platter makes the design complex, especially in small form factors. There are more issues related to thermal expansion and contraction. The heat generated inside the hard drive is increased. The logic required to coordinate and optimize the seeks going on with the two sets of heads requires a great deal of work. And with hard disk designs and materials changing so quickly, this work would have to be re-done fairly often.

However, the biggest reasons why multiple actuators designs aren't practical are related to marketing. The added expense in writing specialty electronics and duplicating most of the internal control components in the drive would make it very expensive, and most people just don't care enough about performance to pay the difference. Hard disks are complex technology that can only be manufactured economically if they are mass-produced, and the market for those who would appreciate the extra actuators isn't large enough to amortize the development costs inherent in these fancy designs. It makes more sense instead to standardize on mass-produced drives with a single actuator stack, and build RAID arrays from these for those who need the added performance. Compare a single 36 GB drive to an array of four 9 GB drives: in effect, the array *is* a 36 GB drive with four sets of *everything*. It would in most cases yield performance and reliability superior to a single 36 GB drive with four actuators, and can be made from standard components without special engineering.

Next: Hard Disk Spindle Motor

Hard Disk Spindle Motor

The spindle motor, also sometimes called the spindle shaft, is responsible for turning the hard disk platters, allowing the hard drive to operate. The spindle motor is sort of a "work horse" of the hard disk. It's not flashy, but it must provide stable, reliable and consistent turning power for thousands of hours of often continuous use, to allow the hard disk to function properly. In fact, many drive failures are actually failures with the spindle motor, not the data storage systems.



A hard disk spindle motor, stripped of its platters and other components, and detached from the drive's base casting. You can see that it attaches with three screws around its perimeter. The shiny metal is the shaft, which rotates; the dull metal is the base of the motor. The six small screw holes on the top of the shaft are for securing the platters. You can also see a large screw hole in the center top of the shaft, which is used to attach the top cover to the spindle shaft for added stability. the four wire connector attaches to the hard disk logic board.

For many years hard disks all spun at the same speed. In the interests of performance, manufacturers have been steadily ratcheting up their products' spin speeds over the last few years. These higher-speed spindles often have issues related to the amount of heat and vibration they generate. The increased performance and also the new potential issues related to the spindle motor have given it renewed attention in the last few years.

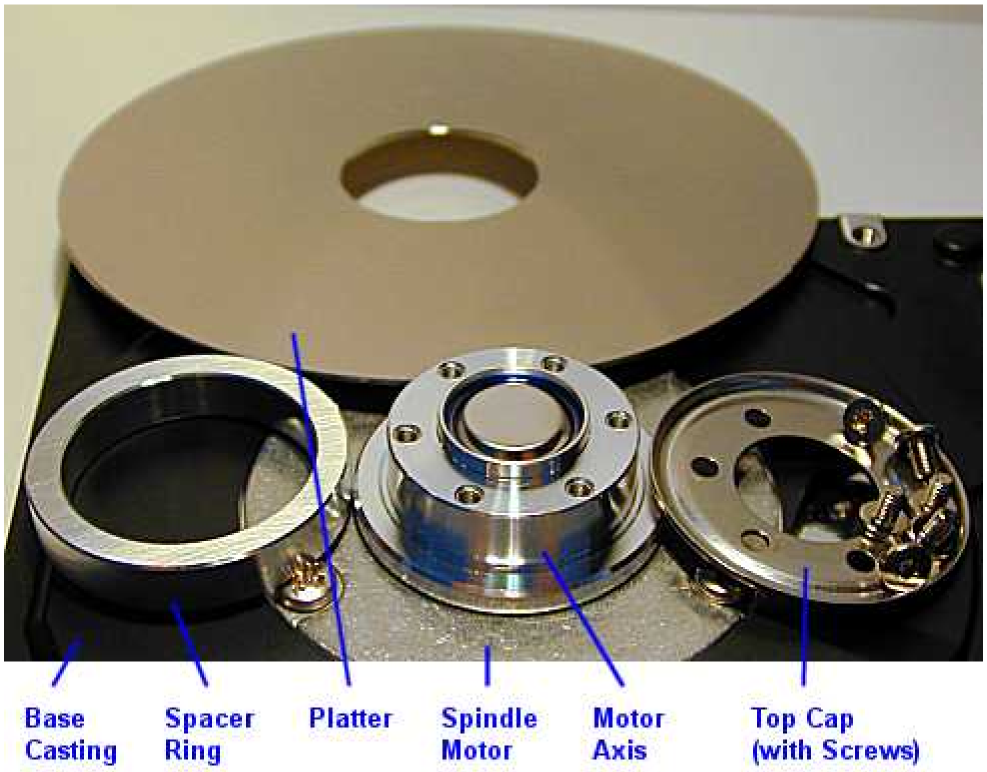
Next: Spindle Motor Operation

Spindle Motor Operation

It will not surprise you, given the precision involved in every facet of the construction of the hard disk drive, that the spindle motor has several important demands placed upon it. First, the motor must be of high quality, so it can run for thousands of hours, and tolerate thousands of start and stop cycles, without failing. Second, it must be run smoothly and with a minimum of vibration, due to the tight tolerances of the platters and heads inside the drive. Third, it must not generate excessive amounts of heat or noise. Fourth, it should not draw too much power. And finally, it must have its speed managed so that it turns at the proper speed.

To meet these demands, all PC hard disks use servo-controlled DC spindle motors. A servo system is a closed-loop feedback system; this is the exact same technology as is used in modern voice coil actuators, and I discuss how servo systems work in detail in that section. In the case of the spindle motor, the feedback for the closed-loop system comes in the form of a speed sensor. This provides the feedback information to the motor that allows it to spin at exactly the right speed.

All hard disk spindle motors are configured for *direct connection*; there are no belts or gears that are used to connect them to the hard disk platter spindle. The spindle onto which the platters are mounted is attached directly to the shaft of the motor. The platters are machined with a hole the exact size of the spindle, and are placed onto the spindle with separator rings (spacers) between them to maintain the correct distance and provide room for the head arms. The entire assembly is secured with a head cap and usually, lots of small Torx screws.



Components of the spindle motor assembly. Platters and spacer rings have the
same inside diameter and alternated over the spindle motor axis to build the platter stack. The top cap goes, well, on top, and is secured using those annoying
teeny weeny Torx screws. :^) Note that this particular drive does not have a screw hole in the center of the spindle motor shaft to secure it to the drive cover.

The amount of work that the spindle motor has to do is dependent on a number of factors. The first is the size and number of platters that it must turn. Larger platters and more platters in a drive mean more mass for the motor to turn, so more powerful motors are required. The same is true of higher-speed drives. Finally, with power management becoming more of a concern today, users increasingly want hard disks that will spin up from a stopped position to operating speed quickly, which also requires faster or more powerful motors.

One important quality issue that has become a focus of attention with newer hard disks is the amount of noise, heat and vibration they generate. The reason for this becoming more of an issue is the increase in spindle speed in most drives. On older hard disks that typically spun at 3600 RPM, this was much less of a problem. Some newer drives, especially 7200 and 10,000 RPM models, can make a lot of noise when they are running. If possible, it's a good idea to check out a hard disk in operation before you buy it, to assess its noise level and see if it bothers you; this varies greatly from individual to individual. The noise produced also varies to some extent depending on the individual drive even in the same family. Heat created by the spindle motor can eventually cause damage to the hard disk, which is why newer drives need more attention paid to their cooling.

Tip: Newer high-speed drives almost always run cooler and quieter than the first generation of drives at any new spindle speed. It can be painful to be a pioneer. :^)

A critical component of the hard disk's spindle motor that has received much attention recently due to concerns over noise, vibration and reliability is the set of *spindle motor bearings*. Bearing are precision components that are placed around the shaft of the motor to support them and ensure that the spindle turns smoothly with no wobbling or vibration. As hard disk speeds increase, the demands placed on the bearings increase dramatically. Many of the noise and heat issues created by fast motors are related to the bearings, so engineers are constantly trying to improve them.

Most hard disk motors use *ball bearings*. These are small metal balls that are placed in a ring around the spindle motor shaft; you have no doubt seen them used in many different applications outside the PC. They are also used elsewhere inside the PC, such as higher-quality power supply fans. Some hard disks use special *fluid-dynamic bearings* instead of ball bearings. Here, the metal balls are replaced with a thick oil, which reduces noise significantly because the metal-to-metal of ball bearings is removed. It also theoretically greatly increases bearing life, though ball bearings should have a life exceeding the hard disk's normal service life anyway.

Next: Spindle Speed

Spindle Speed

As hard disks become more advanced, virtually every component in them is required to do more and work harder, and the spindle motor is no exception. As discussed in detail here, increasing the speed at which the platters spin improves both positioning and transfer performance: the data can be read off the disk faster during sequential operations, and rotational latency--the time that the heads must wait for the correct sector number to come under the head--is also reduced, improving random operations. For this reason, there has been a push to increase the speed of the spindle motor, and more than at any other time in the past, hard disk spin speeds are changing rapidly.

At one time all PC hard disks spun at 3,600 RPM; in fact, for the first 10 years of the PC's existence, that was all there was. One reason for this is that their designs were based on the old designs of large, pre-PC hard disks that used AC motors, and standard North American AC power is 60 Hz per second: 3,600 RPM. In the early 1990s manufacturers began to realize how much performance could be improved by increasing spindle speeds. The next step up from 3,600 RPM was 4,500 RPM; 5,400 RPM soon followed and became a standard for many years. From there speeds have steadily marched upwards. Usually, faster PC hard disk speeds "debut" on SCSI drives that are used in higher-performance applications, and then filter down to IDE/ATA a few years later. At one time 7,200 RPM spindles were only found on top-of-the-line SCSI drives; they are now being used in consumer IDE/ATA disks sold at retail while SCSI has moved on to loftier heights. This table shows the most common PC spindle speeds, their associated average rotational latency, and their typical applications as of early 2000:

Spindle Speed (RPM)	Average Latency (Half Rotation) (ms)	Typical Current Applications
3,600	8.3	Former standard, now obsolete
4,200	7.1	Laptops
4,500	6.7	IBM Microdrive, laptops
4,900	6.1	Laptops
5,200	5.8	Obsolete
5,400	5.6	Low-end IDE/ATA, laptops
7,200	4.2	High-end IDE/ATA, Low-end SCSI
10,000	3.0	High-end SCSI
12,000	2.5	High-end SCSI
15,000	2.0	Top-of-the-line SCSI

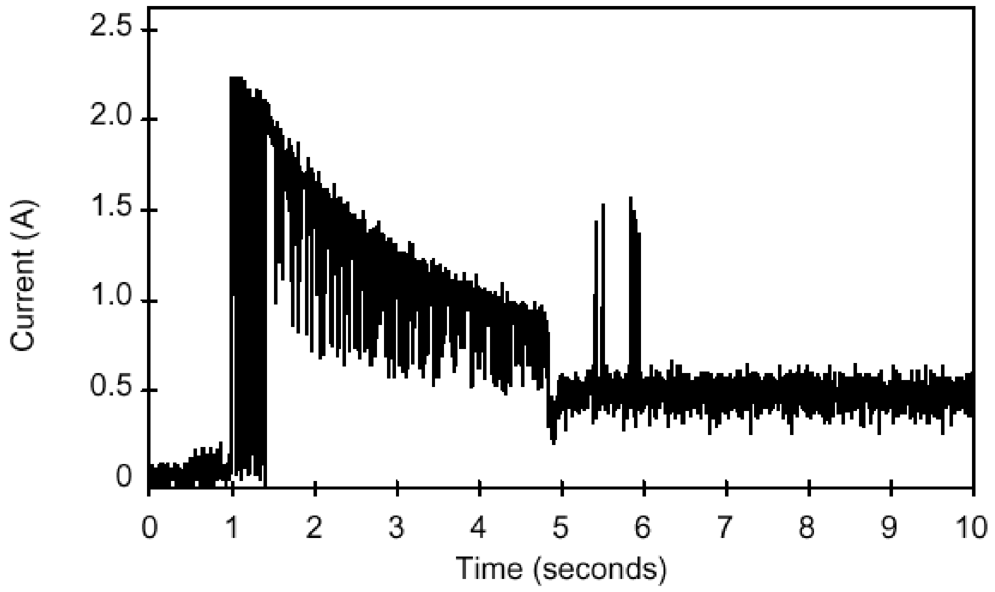
Note: Hard disks for laptops and specialty applications come in a wide variety of spindle speeds, even beyond the several speeds listed above. I have not exhaustively researched and listed these here.

Increasing spindle motor speed creates many design challenges, particularly aimed at keeping vibration and heat under control. As discussed here, when the motor spins faster these become more of an issue; some high-end drives have very serious heat, vibration and noise problems that require special mounting and cooling work to allow them to run without problems. To some extent, there is a tradeoff between spindle speed, and the heat and noise issue. Engineers generally focus on keeping these matters under control, and usually improve them significantly after the first generation of drives at any given spindle speed. However, in some applications, using a slower and quieter drive can make sense.

Next: Continuous Power vs. Peak Power at Spin-Up

Continuous Power vs. Peak Power at Spin-Up

Most of the power used by a modern hard disk drive is taken up by the spindle motor. Due to their smaller, more efficient designs, newer drives require relatively little power to keep their platters spinning continuously. Even drives with faster spindle speeds take less power than the large, inefficient motors of the drives of a decade ago. However, when the hard disk is first started up, the motor can draw a peak level of power that is more than two times what it takes to keep the disks spinning. (Most things require more energy to start them in motion than to keep them in motion, but in the case of hard disk motors this is especially so because of their electrical characteristics.) While in most cases even the peak start-up power usage isn't all that much, there can be an issue when you are using multiple hard disks that all try to spin up at once when you turn on the machine. The ratings of your power supply, particularly for the +12 V level, must be sufficient to take this initial demand into account; these matters are discussed in this section of the power supply reference.



12V power profile (current vs. time) of an IDE/ATA hard disk at startup. You can see that the peak power draw is over quadruple the steady-state operating requirement. The graph appears "noisy" due to frequent oscillations in current requirements

Original image © Seagate Technology
Image used with permission.

Hard drive manufacturers generally program their drives so that when two are used on the same IDE channel as master and slave, the slave drive delays its spin up by several seconds to offset its start from that of the master drive, and reduce the peak demand on the power supply. Similarly, many SCSI drives can be programmed to delay their startups, using the "Remote Start" command or a special jumper. These features offset the time that the drives spin up to reduce the peak demand on the power supply.

Another issue with spin-up relates to boot speed. In order for the system to boot up properly when it is turned on, the platters must be brought from a standstill to operating speed in a relatively short period of time. Modern PCs often have "expedited" BIOSes that go from power on to attempting to boot the hard disk in only a few seconds. If the disk isn't ready, often the system will hang, even though it will boot if you press the reset button and start the boot process over again. Some system BIOSes include a delay feature that can be used to slow down the boot process to avoid this problem. Otherwise, slowing down the boot process by turning on the memory test feature, for example, can sometimes cure this problem.

Next: Power Management

Power Management

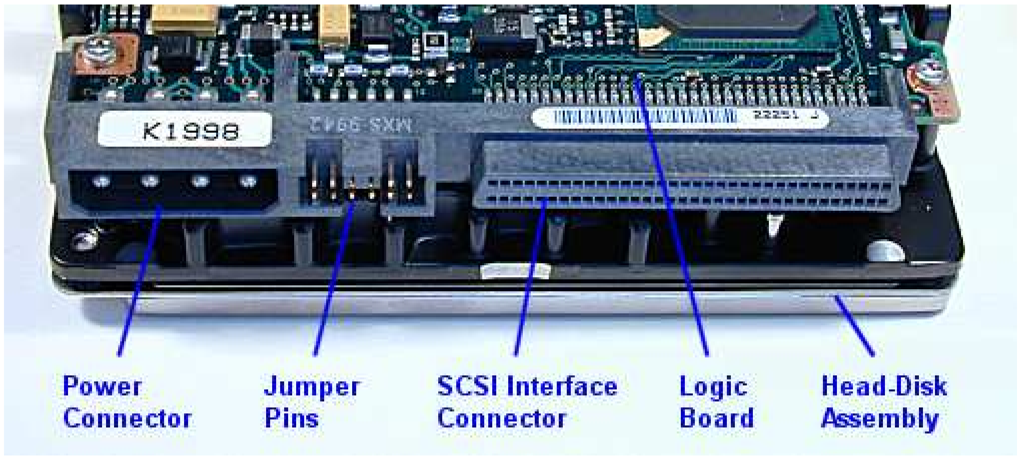
Power management is a set of protocols used to allow PCs to reduce the amount of power that they consume, especially when they lie idle. Hard disk drives are always spinning during operation, and the spindle motor is what is using the bulk of the power consumed by the drive as a whole. Therefore, many power management schemes include a mechanism by which the hard disk's spindle motor can be "spun down" after a certain amount of inactivity, and then can be "spun up" (reactivated) when the system needs them again.

There is some controversy surrounding this feature. First, there is some debate as to whether or not dramatically increasing the number of times the drive spins up and down (which power management does) is detrimental to the drive or could cause it to fail prematurely. Second, there is some skepticism about how much energy is really saved, given that the hard disk doesn't use that much power under normal operation to begin with. See here for a discussion of pros and cons of power management. Certainly for laptop users power management is an important feature; all laptop hard disks support power management via BIOS settings and/or operating system controls.

Next: Hard Disk Connectors and Jumpers

Hard Disk Connectors and Jumpers

Several different connectors and jumpers are used to configure the hard disk and connect it to the rest of the system. The number and types of connectors on the hard disk depend on the data interface it uses to connect to the system, the manufacturer of the drive, and any special features that the drive may possess. Instructions for setting common jumpers are usually printed right on the drive; full instructions for all jumpers will be in the product's manual, or on the manufacturer's web site.



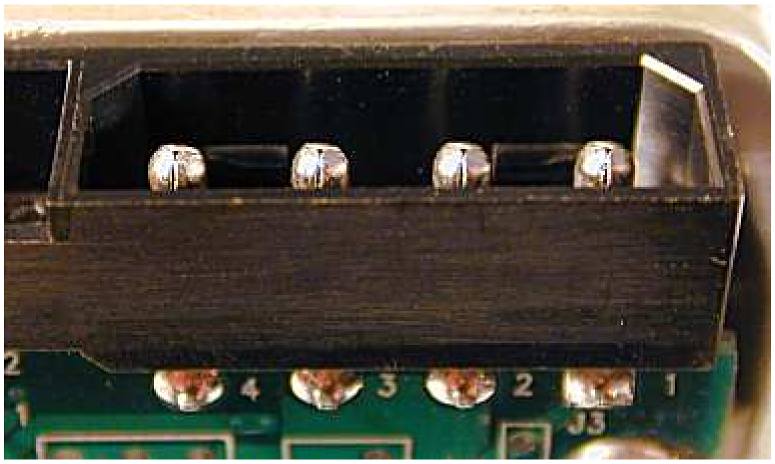
Some of the connectors and jumper pins on a 3.5", 36 GB, 10,000 RPM SCSI Cheetah drive.

If you are unfamiliar with the concepts behind jumpers, you can read a description of them in this PC fundamentals page.

Next: Power Connector

Power Connector

Hard disk drives use a standard, 4-pin male connector plug, that takes one of the power connectors coming from the power supply. This keyed, 4-wire plastic connector provides +5 and +12 voltage to the hard disk. See this discussion of drive power plugs in the power supply reference for more information.



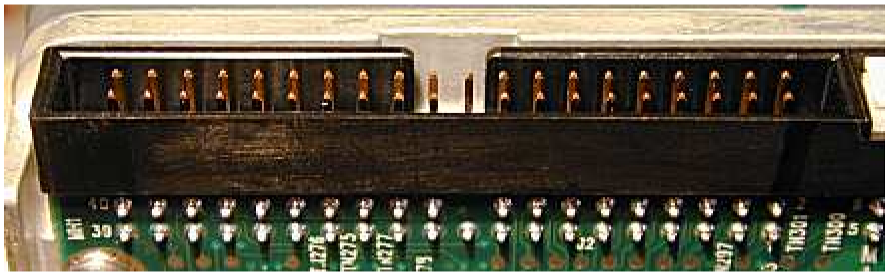
A standard hard disk power connector. Note the "4, 3, 2, 1" pin markings on the logic board, and the square shape of solder pad for pin #1.

Next: Data Interface Connector

Data Interface Connector

Modern hard disk drives use one of two interfaces: IDE (ATA) and its variants, or SCSI. You can tell immediately by looking at the back of the hard disk which interface is being used by the drive:

- **IDE/ATA:** A 40-pin rectangular connector. See here for more information on IDE/ATA cables and connectors.
- **SCSI:** A 50-pin, 68-pin, or 80-pin D-shaped connector (the same shape used for serial and parallel port connectors). A 50-pin connector means the device is narrow SCSI; 68 pins means wide SCSI; 80 pins means wide SCSI using single connector attachment (SCA). See here for more on SCSI cables and connectors.



A standard hard disk IDE/ATA data interface connector. If you look closely you can see markings for pins #1, #2, #39 and #40. A 50-pin SCSI interface connector looks identical except that it has 25 columns of pins instead of 20 (they look so much alike that getting the two mixed up is common). A 68-pin SCSI interface connector is pictured on the parent page of this page.

The connectors on hard disk drives are generally in the form of a 2xN rectangular grid of pins (where N is 20, 25, 34 or 40 depending on the interface). Older ST-506 (also called MFM, RLL) and ESDI hard disks used two data connectors, one 34 pins and the other 20 pins. These connectors were often not in the form of pins but rather card edge connectors, such as those used by ISA expansion cards. Some SCSI connectors may have different shapes, especially older ones.

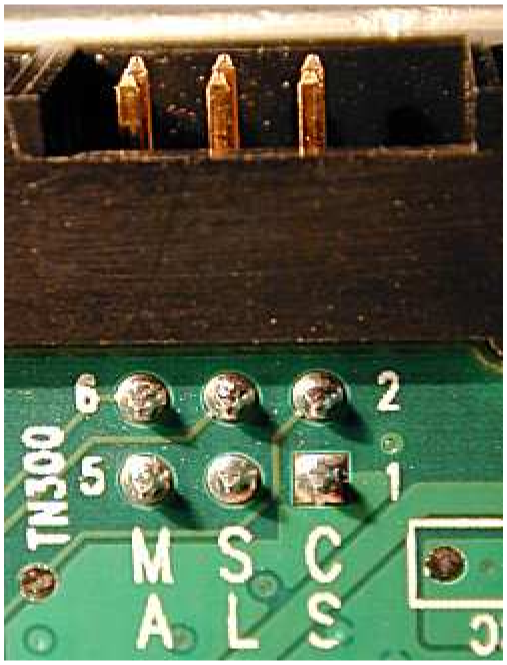
While most current SCSI interface connectors are keyed to prevent incorrect insertion (because they are D-shaped), this is not always the case for other interfaces. For this reason, it is important to make sure that the cable is oriented the correct way before plugging it in. The cable has a red stripe to indicate wire #1 and the hard disk uses markers of one form or another to indicate the matching pin #1.

Next: IDE/ATA Configuration Jumpers

IDE/ATA Configuration Jumpers

IDE/ATA hard disks are fairly standard in terms of jumpers. There are usually only a few and they don't vary greatly from drive to drive. Here are the jumpers you will normally find:

- **Drive Select:** Since there can be two drives (master and slave) on the same IDE channel, a jumpers is normally used to tell each drive if it should function as a master or slave on the IDE channel. For a single drive on a channel, most manufacturers instruct that the drive be jumpered as master, while some manufacturers (notably Western Digital) have a separate setting for a single drive as opposed to a master on a channel with a slave. The terms "master" and "slave" are misleading since the drives really have no operational relationship. See this section on IDE/ATA jumpering for more.
- **Slave Present:** Some drives have an additional jumper that is used to tell a drive configured as master that there is also a slave drive on the ATA channel. This is only required for some older drives that don't support standard master/slave IDE channel signaling.
- **Cable Select:** Some configurations use a special cable to determine which drive is master and which is slave, and when this system is used a cable select jumper is normally enabled.
- **Size Restriction Jumper:** Some larger hard disk drives don't work properly in older PCs that don't have a BIOS program modern enough to recognize them. To get around this, some drives have special jumpers that, when set, will cause them to appear as a smaller size than they really are to the BIOS, for compatibility. For example, some 2.5 GB hard disks have a jumper that will cause them to appear as a 2.1 GB hard disk to a system that won't support anything over 2.1 GB. These are also sometimes called *capacity limitation jumpers* and vary from manufacturer to manufacturer. See here for more on this.



Jumper block for an IDE hard disk. The jumpers are labeled "MA" (master), "SL" (slave) and "CS" (cable select). Other IDE drives will have slightly different jumper configuration or placement.

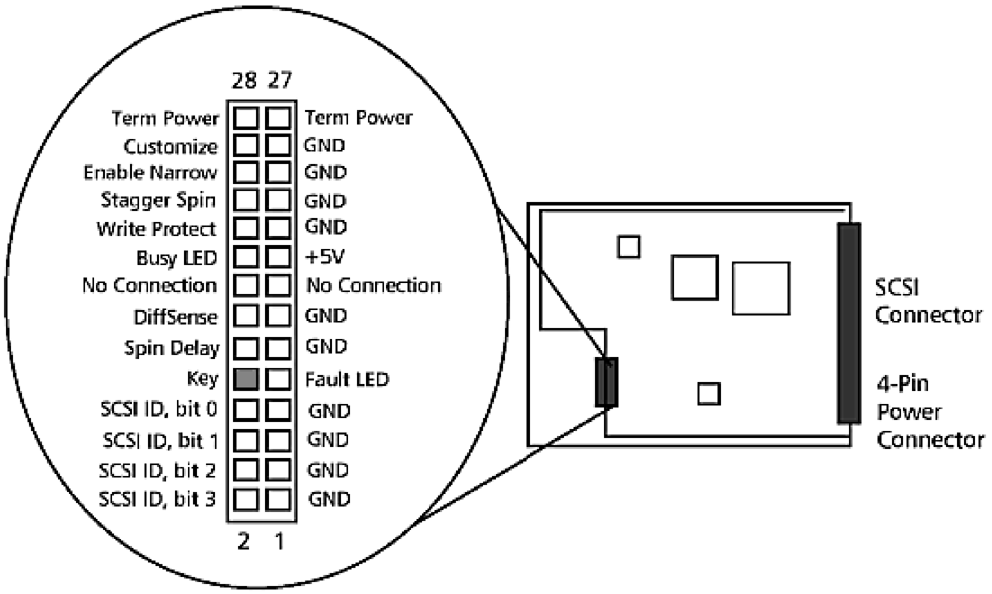
See here for full details on IDE/ATA hard disk setup.

Next: SCSI Configuration Jumpers

SCSI Configuration Jumpers

SCSI hard disks have more sophisticated controllers than their IDE/ATA cousins, and as a result typically have many more jumpers that can be set to control their operation. They also tend to vary much more from manufacturer to manufacturer, and from model to model, in the number and types of jumpers they have. Typically the following are the most common and important SCSI drive jumpers:

- **SCSI Device ID:** Every device on a SCSI bus must be uniquely identified for addressing purposes. Narrows SCSI drives will have a set of three jumpers that can be used to assign the disk an ID number from 0 to 7. Wide SCSI drives will have four jumpers to enable ID numbers from 0 to 15. Some systems don't use jumpers to configure SCSI device IDs.
- **Termination Activate:** The devices on the ends of the SCSI bus must terminate the bus for it to function properly. If the hard disk is at the end of the bus, setting this jumper will cause it to terminate the bus for proper operation. Not all drives support termination.
- **Disable Auto Start:** When present, this jumper will tell the drive not to automatically spin up when the power is applied, but instead wait for a start command over the SCSI bus. This is usually done to prevent excessive startup load on the power supply. Some manufacturers invert the sense of this jumper; they disable startup by default and provide an "Enable Auto Start" jumper.
- **Delay Auto Start:** This jumper tells the drive to start automatically, but wait a predefined number of seconds from when power is applied. It is also used to offset motor startup load on systems with many drives.
- **Stagger Spin:** An "enhanced version" of "Delay Auto Start". When a system with many hard drives has this option set for each unit, the drives stagger their startup time by multiplying a user-defined constant times their SCSI device ID. This ensures no two drives on the same SCSI channel will start up simultaneously.
- **Narrow/Wide:** Some drives have a jumper to control whether they will function in narrow or wide mode.
- **Force SE:** Allows newer Ultra2, Wide Ultra2, Ultra160, Ultra160+ or other LVD SCSI drives to be forced to use single-ended (SE) operation instead of low voltage differential (LVD).
- **Disable Parity:** Turns off parity checking on the SCSI bus, for compatibility with host adapters that don't support the featurer.



Option header block signals and functions for the Quantum Atlas 10K SCSI drive.

Original image © Quantum Corporation
Image used with permission.

This is list is not intended to be exhaustive; many SCSI drives have additional special features that are enabled through more jumpers. Some drives have replaced some of their jumpers with software commands sent over the SCSI interface. SCSI jumpers are often clustered together into what is called an *option block*. See here for details on SCSI hard disk setup.

Next: LED Connector

LED Connector

Hard disks use a light-emitting diode or LED to indicate drive activity (if you are unfamiliar with LEDs, see the page on basic electrical components for some background information). The hard disk activity LED is a very useful indicator that generally tells the PC user at a glance when the system is active. The first PC hard disks shipped with a faceplate (or bezel) on the front. The hard disk was mounted into an external hard drive bay (in place of a floppy disk drive) and its integral LED was visible from the front of the PC, because the drive's front was actually protruding from the case, much as floppy disk drives still do.

It was quickly realized that having the disks mounted internally to the case made more sense than using external drive bays, but the LED was still desirable. So a remote LED was mounted to the case and a wire run to a two-pin connector on the hard disk itself. This system worked fine when there was just one hard disk, but became a problem in systems that had two or three hard disks. Eventually, the case LED was made to connect to the hard disk *controller* instead, to show activity on any of the hard disks that were managed by the controller.

Modern PCs have integrated IDE/ATA controllers built into the chipset on the motherboard, so the LED is usually connected to special pins on the motherboard itself. For systems that use add-in controllers, the LED is connected to the controller, as it was in the days before integrated controllers. Over time, as connecting the LED to the controller has become the standard, most manufacturers have dropped entirely the LED connector on the disk itself on IDE/ATA drives.

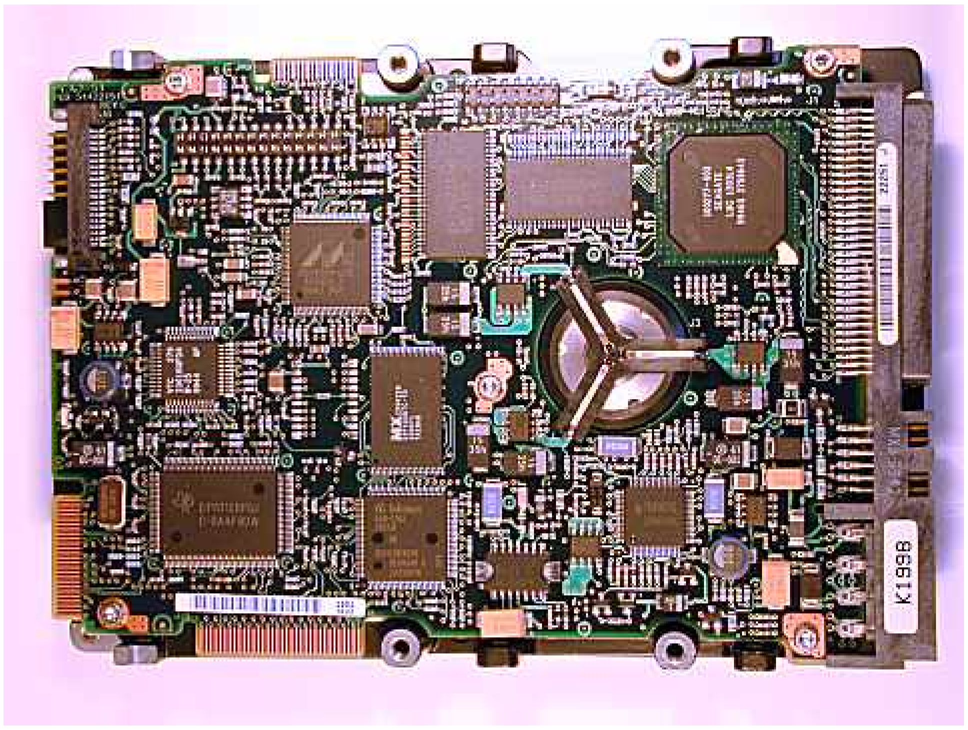
Since support for SCSI drives is not present in the vast majority of PC motherboards, they often do still come with an external LED connector.

Next: Hard Disk Logic Board

Hard Disk Logic Board

All modern hard disks are made with an intelligent circuit board integrated into the hard disk unit. Early hard disks were "dumb", meaning that virtually all of the control logic for controlling the hard disk itself was placed into the controller plugged into the PC; there were very little smarts on the drive itself, which had to be told specifically how to perform every action. This design meant that it was necessary for controllers to be very *generalized*; they could not be customized to the particulars of a given disk drive since they had to be able to work with any type. Older drives were similar enough, and sufficiently undemanding in terms of performance that this arrangement was acceptable. As newer drives were introduced with more features and faster speed, this approach became quite impractical, and once electronics miniaturization progressed far enough, it made sense to move most of the control functions to the drive itself.

The most common interface for PC hard disks is called *IDE*, which in fact stands for *Integrated Drive Electronics*. This name is something of a misnomer today. When it was introduced, IDE was distinguished from the other interfaces of the day by having the integrated electronics on the drive, instead of on the controller card plugged into the system bus like older interfaces. However, the term really refers to where the control logic is and not the interface itself, and since all hard disks today use integrated electronics the name doesn't mean anything any more, despite the fact that everyone continues to use it. The other popular PC hard disk interface today, SCSI, also uses drives that have integrated controllers. The more correct name for the IDE interface is *AT Attachment* or *ATA*; see here for more.



The logic board of a Cheetah 10,000 RPM 36 GB hard disk drive.

The main interface and power connectors are on the right-hand side; auxiliary connectors on the bottom and left side. The bottom of the spindle motor protrudes through a round hole made for it in the circuit board.

Today's hard disks contain logic boards that are in most ways more sophisticated than an entire early PC! In fact, most of them contain more memory and faster internal processors than an entire PC of even the mid-1980s. The logic board performs several important functions, and as hard disks become faster and more sophisticated, more functions are added to the logic board. This means the logic circuits need to be more powerful, to handle changes like geometry translation, advanced reliability features, more complicated head technologies, faster interfaces, and higher bandwidth data streaming from the disk itself.

Next: Control Circuitry

Control Circuitry

The drive's internal logic board contains a microprocessor and internal memory, and other structures and circuits that control what happens inside the drive. In many ways, this is like a small embedded PC within the hard disk itself. The control circuitry of the drive performs the following functions (among others):

- Controlling the spindle motor, including making sure the spindle runs at the correct speed.
- Controlling the actuator's movement to various tracks.
- Managing all read/write operations.
- Implementing power management features.
- Handling geometry translation.
- Managing the internal cache and optimization features such as pre-fetch.
- Coordinating and integrating the other functions mentioned in this section, such as the flow of information over the hard disk interface, optimizing multiple requests, converting data to and from the form the read/write heads require it, etc.
- Implementing all advanced performance and reliability features.

The control circuitry of the drive is underrated and misunderstood, even by those interested in hard disk performance issues. The reason is that the quality or optimization level of the control circuitry doesn't manifest itself as a single, simple specification. You can't easily compare the circuitry of five different drive families. Most hard disk manufacturers provide very little information about the "guts" of the logic board, and even if they did, most people wouldn't know what to do with the information (including myself).

In fact, differences in control circuitry account for part of the differences in some specifications. This is probably most true of seek performance, as discussed here. Beyond this, you can't really tell much about what's inside the circuitry. However, if you use two different drives that have very similar specifications and run on the same interface on the same PC, but one just "feels faster" than the other, differences in their internal circuitry may be part of the answer.

Next: Sense, Amplification and Conversion Circuits

Sense, Amplification and Conversion Circuits

Since the signals read from the hard disk on a modern drive are very weak, special circuits are required to read the low-voltage signals coming from the drive heads, amplify them and interpret them, to decide if each signal read is a one or a zero. As hard disks have their data packed more tightly the signals get weaker, and this job becomes more complex. The new PRML read methods used in modern hard disks involve using complicated digital signal processing techniques to allow the use of even weaker signals, making this job more complex.

During the write process the opposite conversion is necessary. The logic board must translate the "1"s and "0"s of the data stream into appropriate commands and signals for the read/write heads, using the drive's encoding method.

Next: Interface Hardware

Interface Hardware

While the drive electronics on a modern disk are moved to the disk itself, there is still a controller card or integrated controller on the motherboard, that the disk must talk to. The difference between older drives and newer ones is that older controllers actually ran the internals of the disk itself, while the newer ones just run the data interface between the disk and the rest of the system. Every hard disk's logic contains interface hardware to manage this flow of information between itself and the controller it is talking to.

An integrated IDE/ATA controller chip from a Fujitsu hard disk logic board. While at one time interface control functions required many different chips, the advantages of integration have spurred the development of single-chip interface solutions, even by third parties (Cirrus Logic doesn't make hard disks.) In many ways it makes more sense for Fujitsu to use this chip rather than reinvent the wheel.

The interface hardware itself ranges from relatively simple (slower, older IDE/ATA interfaces) to relatively complex (newer, faster IDE/ATA and SCSI interfaces). In particular, SCSI interface hard disks have considerable "smarts" on them to handle the increased sophistication and the wide variety of commands available on that interface.

Next: Firmware

Firmware

Since modern hard disks have internal microprocessors, they also have internal "software" that runs them. (Wouldn't it be neat if they had internal hard disks? :^)) These routines are what run the control logic and make the drive work. Of course this isn't really software in the conventional sense, because these instructions are embedded into read-only memory. This code is analogous to the system BIOS: low-level, hardware-based control routines, embedded in ROM. It is usually called *firmware*, with the word "firm" intending to connote something in between "hard" and "soft". The functions that run the logic board's circuitry could be implemented strictly with hardware devices, as was done with early drives. However, this would be expensive and inflexible for today's sophisticated controllers, since it would make it difficult to update or adapt the logic to match changes in hard disks or the devices they interface with.

Much the way the system BIOS benefits from being in a changeable ROM chip that can be modified relatively easily, the hard disk's firmware does as well. In fact, in many drives the firmware can be updated under software control, very much the same way that a flash BIOS works. Unlike the system BIOS, this is only very rarely done, when a particular sort of problem exists with the firmware logic that can be fixed without requiring a physical hardware change. If you suspect that your drive needs new firmware, check the drive manufacturer's web site. There you will find the instructions that tell you if you need an update, and if so, how to accomplish it.

Next: Multiple Command Control and Reordering

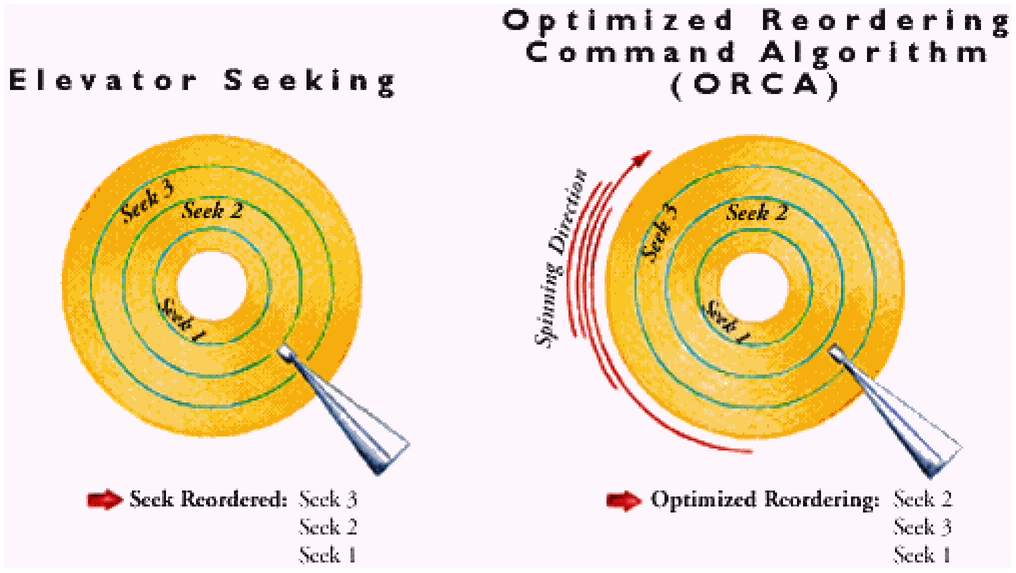
Multiple Command Control and Reordering

The IDE/ATA interface used on most PCs is generally limited to a single transaction outstanding on the interface at any given time. This means the hard disk can not only do just one thing at a time, it can't even keep track of what it needs to do next. It must be fed commands one at a time by the controller. In contrast, newer drives using the SCSI interface generally include the ability to handle multiple requests, up to a certain number. This advanced feature, sometimes called *command queuing and reordering* or *multiple command queuing*, is very useful for servers and other systems being used by multiple people (while its absence is generally not a problem for most single-user PCs equipped with IDE/ATA hardware.) You can read more about this feature here.

If the hard drive's logic circuitry receive multiple commands to read or write from the disk, it must process them and figure out where on the disk the data is for each request. Some requests may be filled from the internal cache; these would generally be filled immediately. For the remainder, the controller must decide in which order to perform them. Since random reads or writes on even the fastest hard disks take thousands of times longer than computing operations, this determination is very important to overall performance. There are probably dozens of different specific algorithms that could be used to decide which commands get fulfilled first. However, they generally fall into these three categories:

- **First In, First Out:** The drive processes the requests in the order that the requests arrived. This is the "supermarket deli" algorithm: simple in both concept and implementation; but in the case of hard disks, poor in performance. In the deli, all the customers are standing in the same place; on the hard disk, the sectors to be read or written might be in very different areas, and processing them just based on which came first will result in no optimization whatsoever. Imagine a 100-story building with the elevator on the ground floor. A person on floor #77 presses the "Up" button, followed immediately by someone on floor #31, then floor #94, and then floor #20. Filling these requests in the order received would result in a lot of needless waiting around (especially by the poor guy on floor #20!)
- **Seek Time Optimization:** The drive analyzes all outstanding requests, and reorders them based on their cylinder number, compared to the cylinder where the heads currently are located. This is *seek time optimization*, sometimes also called *elevator seeking* because it is designed to prevent needless swings of the head actuator across the surface of the disk, much the way a real elevator would sort the requests given in the example above to avoid wasteful up-and-down activity.
- **Access Time (Seek and Latency) Optimization:** The problem with seek time optimization is that it doesn't take into account rotational latency. Two requests might be on cylinders that are very close to each other, but in very different places within the track; meanwhile, there might be a third sector that is a few cylinders further away but much closer overall to where the first request is. The most advanced logic boards in the newest drives will carefully analyze their requests

and determine which to do first based on both how much time will be required to seek to the various sectors' cylinders, and how much time will elapse waiting for the data to rotate under the heads. This is often called *multiple command reordering* or *multiple command optimization*. Needless to say, this is much more complicated than regular elevator seeking, because the drive must know many more details about exactly where each sector is, and must understand its own performance attributes just as well. If the logic is too "aggressive" and the head takes longer to seek to a location than anticipated, it might miss the correct sector as the drive rotates, and have to wait for nearly a full rotation of the disk until that sector comes around again.


Elevator Seeking
Optimized Reordering Command Algorithm (ORCA)
Seek 3
Seek 2
Seek 1
Spinning Direction
Seek 3
Seek 2
Seek 1
Seek Reordered: Seek 3 Seek 2 Seek 1
Optimized Reordering: Seek 2 Seek 3 Seek 1

Quantum's version of command optimization is called "ORCA", as you can see above. The diagram shows graphically what the idea is with this technology. Note when comparing the two figures above, that on the one at right, "Seek 3" has been "rotated" counter-clockwise a bit. That's significant: if you applied "ORCA" to the figure at left, with "Seek 2" and "Seek 3" so close to each other, it would probably not be possible to do them in the order "2, 3, 1" because there wouldn't be enough time to change tracks before "Seek 3" rotated
past the head. If you tried, you'd end up doing worse than if you just did "2, 1, 3".
This is all part of what the algorithm has to figure out as part of its job.

Image © Quantum Corporation
Image used with permission.

You may recall that I mentioned in the section on control circuitry that the importance of the drive's internal circuitry is typically under-appreciated. I said that this is generally because it is hard to "boil down" the differences between drives into a single metric, and because it is nearly impossible to get detailed information about the circuitry's internal logic. This section on multiple command handling provides a good illustration of this phenomenon. Two drives could have nearly-identical seek times, spindle speeds and transfer rates, but one could handle heavy access by multiple users much more gracefully if its algorithms were more efficient.

Next: Hard Disk Cache and Cache Circuitry

Hard Disk Cache and Cache Circuitry

All modern hard disks contain an integrated *cache*, also often called a *buffer*. The purpose of this cache is not dissimilar to other caches used in the PC, even though it is not normally thought of as part of the regular PC cache hierarchy. The function of cache is to act as a buffer between a relatively fast device and a relatively slow one. For hard disks, the cache is used to hold the results of recent reads from the disk, and also to "pre-fetch" information that is likely to be requested in the near future, for example, the sector or sectors immediately after the one just requested.

The use of cache improves performance of any hard disk, by reducing the number of physical accesses to the disk on repeated reads and allowing data to stream from the disk uninterrupted when the bus is busy. Most modern hard disks have between 512 KiB and 2 MiB of internal cache memory, although some high-performance SCSI drives have as much as 16 MiB, more than many whole PCs have!

Note: When someone speaks generically about a "disk cache", they are usually *not* referring to this small memory area inside the hard disk, but rather to a cache of *system memory* set aside to buffer accesses to the disk system.

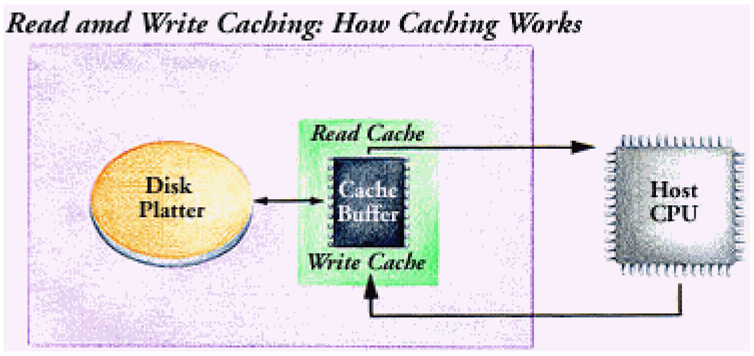
Next: Cache Circuitry and Operation

Cache Circuitry and Operation

The reason that the hard disk's cache is important is due to the sheer difference in the speeds of the hard disk and the hard disk interface. Finding a piece of data on the hard disk involves random positioning, and incurs a penalty of *milliseconds* as the hard disk actuator is moved and the disk rotates around on the spindle. In today's PCs, a millisecond is an *eternity*. On a typical IDE/ATA hard disk, transferring a 4,096-byte block of data from the disk's internal cache is over 100 times faster than actually finding it and reading it from the platters. *That* is why hard disks have internal buffers. :^) If a seek isn't required (say, for reading a long string of consecutive sectors from the disk) the difference in speed isn't nearly as great, but the buffer is still much faster.

Tip: While different in operation and technology from the system cache, the hard disk buffer is similar in concept and role. You may find the section discussing the system cache helpful if you want to understand more about caching in general.

Note: This section discusses caching in general, and especially as it applies to data reads from the hard disk. *Writes* to the disk have more issues involved with caching; most of what is in this section applies to writes but in addition there are other issues regarding caching writes.



General concepts behind the operation of an internal hard disk cache.

Image © Quantum Corporation
Image used with permission.

The basic principle behind the operation of a simple cache is straightforward. Reading data from the hard disk is generally done in blocks of various sizes, not just one 512-byte sector at a time. The cache is broken into "segments", or pieces, each of which can contain one block of data. When a request is made for data from the hard disk, the cache circuitry is first queried to see if the data is present in any of the segments of the cache. If it is present, it is supplied to the logic board without access to the hard disk's platters being necessary. If the data is not in the cache, it is read from the hard disk, supplied to the controller, and then placed into the cache in the event that it gets asked for again. Since the cache is limited in size, there are only so many pieces of data that can be held before the segments must be recycled. Typically the oldest piece of data is replaced with the newest one. This is called *circular*, *first-in, first-out* (*FIFO*) or *wrap-around* caching.

In an effort to improve performance, most hard disk manufacturers today have implemented enhancements to their cache management circuitry, particularly on high-end SCSI drives:

- **Adaptive Segmentation:** Conventional caches are chopped into a number of equal-sized segments. Since requests can be made for data blocks of different sizes, this can lead to some of the cache's storage in some segments being "left over" and hence wasted (in exactly the same way that slack results in waste in the FAT file system). Many newer drives dynamically resize the segments based on how much space is required for each access, to ensure greater utilization. It can also change the number of segments. This is more complex to handle than fixed-size segments, and it can result in waste itself if the space isn't managed properly.
- **Pre-Fetch:** The drive's cache logic, based on analyzing access and usage patterns of the drive, attempts to load into part of the cache data that has not been requested yet but that it *anticipates* will be requested soon. Usually, this means loading additional data beyond that which was just read from the disk, since it is statistically more likely to be requested next. When done correctly, this will improve performance to some degree.
- **User Control:** High-end drives have implemented a set of commands that allows the user detailed control of the drive cache's operation. This includes letting the user enable or disable caching, set the size of segments, turn on or off adaptive segmentation and pre-fetch, and so on.

While obviously improving performance, the limitations of the internal buffer should be fairly obvious. For starters, it helps very little if you are doing a lot of random accesses to data in different parts of the disk, because if the disk has not loaded a piece of data recently in the past, it won't be in the cache. The buffer is also of little help if you are reading a large amount of data from the disk, because normally it is pretty small: if copying a 10 MiB file for example, on a typical disk with a 512 kiB buffer, at *most* 5% of the file could be in the buffer: the rest must be read from the disk itself.

Due to these limitations, the cache doesn't have as much of an impact on overall system performance as you might think. How much it helps depends on its size to some extent, but at least as much on the intelligence of its circuitry; just like the hard disk's logic overall. And just like the logic overall, it's hard to determine in many cases exactly what the cache logic on a given drive is like.

Next: Cache Size

Cache Size

In the last couple of years, hard disk manufacturers have dramatically increased the size of the hard disk buffers in their products. Even as recently as the late 1990s, 256 to 512 kiB was common on consumer drives, and it was not unusual to find only 512 kiB buffers on even some SCSI units (though many had from 1 MiB to 4 MiB). Today, 2 MiB buffers are common on retail IDE/ATA drives, and some SCSI drives are now available with a whopping 16 MiB!

I believe there are two main reasons for this dramatic increase in buffer sizes. The first is that memory prices have dropped precipitously over the last few years. With the cost of memory only about $1 per MiB today, it doesn't cost much to increase the amount the manufacturers put into their drives. The second is related to marketing: hard disk purchasers have a perception that doubling or quadrupling the size of the buffer will have a great impact on the performance of the hardware.

The cache chip from a Seagate Barracuda hard disk logic board. This chip is the entire cache: it's a 4 Mib chip, which is 512 kiB, the size of the cache on this drive. Some caches use more than one chip, especially the larger ones.

The size of the disk's cache *is* important to its overall impact in improving the performance of the system, for the same reason that adding system memory will improve system performance, and why increasing the system cache will improve performance as well. However, the attention that the size of the hard disk buffer is getting today is largely unwarranted. It has become yet another "magic number" of the hardware world that is tossed around too loosely and overemphasized by salespeople. In fact, *a benchmarking comparison done by StorageReview.com* showed very little performance difference between 512 kiB and 1 MiB buffer versions of the same Maxtor hard drive. See this section for more on this performance metric.

So, where does this leave us? Basically, with the realization that the size of the buffer is important only to an extent, and that only large differences (4 MiB vs. 512 kiB) are likely to have a significant impact on performance. Also remember that the size of the drive's internal buffer will be small on most systems compared to the amount of system memory set aside by the operating system for its disk cache. These two caches, the one inside the drive and the one the operating system uses to avoid having to deal with the drive at all, perform a similar function, and really work together to improve performance.

Next: Write Caching

Write Caching

Caching reads from the hard disk and caching writes to the hard disk are similar in some ways, but very different in others. They are the same in their overall objective: to decouple the fast PC from the slow mechanics of the hard disk. The key difference is that a write involves a *change* to the hard disk, while a read does not.

With no write caching, every write to the hard disk involves a performance hit while the system waits for the hard disk to access the correct location on the hard disk and write the data. As mentioned in the general discussion of the cache circuitry and operation, this takes at least 10 milliseconds on most drives, which is a long time in the computer world and really slows down performance as the system waits for the hard disk. This mode of operation is called *write-through* caching. (The contents of the area written actually are put into the cache in case it needs to be *read* again later, but the write to the disk always occurs at the same time.)

When write caching is enabled, when the system sends a write to the hard disk, the logic circuit records the write in its much faster cache, and then immediately sends back an acknowledgement to the operating system saying, in essence, "all done!" The rest of the system can then proceed on its merry way without having to sit around waiting for the actuator to position and the disk to spin, and so on. This is called *write-back* caching, because the data is stored in the cache and only "written back" to the platters later on.

Write-back functionality of course improves performance. There's a catch however. The drive sends back saying "all done" when it really isn't done--the data isn't on the disk at all, it's only in the cache. The hard disk's logic circuits begin to write the data to the disk, but of course this takes some time. The hard disk is using a variant of that old "the check is in the mail" trick you might hear when you call someone to remind them of that loan they were supposed to pay back three weeks ago. :^)

Now, this isn't really a problem most of the time, as long as *the power stays on*. Since cache memory is volatile, if the power goes out, its contents are lost. If there were any pending writes in the cache that were not written to the disk yet, they are gone forever. Worse, the rest of the system has no way to know this, because when it is told by the hard disk "all done", it can't really know what that means. So not only is some data lost, the system doesn't even know which data, or even that it happened. The end result can be file consistency problems, operating system corruption, and so on. (Of course, this problem doesn't affect cached reads at all. They can be discarded at any time.)

Due to this risk, in some situations write caching is not used at all. This is especially true for applications where high data integrity is critical. Due to the improvement in performance that write caching offers, however, it is increasingly being used despite the risk, and the risk is being mitigated through the use of additional technology. The most common technique is simply ensuring that the power does not go off! In high-end server environments, with their uninterruptible power supplies and even redundant power supplies, having unfilled cached writes is much less of a concern. For added peace of mind, better drives that employ write caching have a "write flush" feature that tells the drive to immediately write to disk any pending writes in its cache. This is a command that would commonly be sent before the UPS batteries ran out if a power interruption was detected by the system, or just before the system was to be shut down for any other reason.

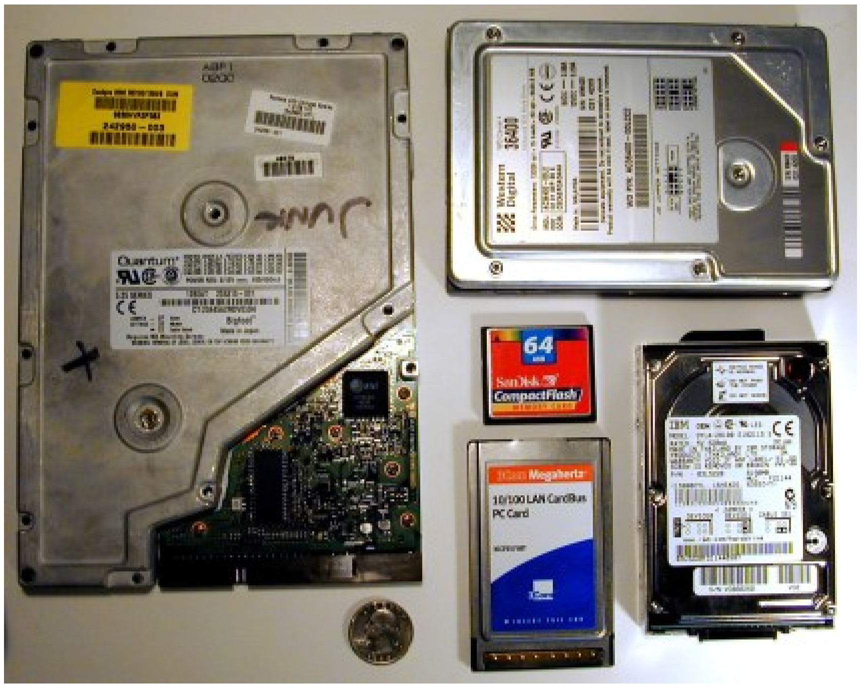
Next: Hard Disk Form Factors

Hard Disk Form Factors

Most hard disks are designed to be installed on the inside of the PC, and are produced in one of a dozen or so standard sizes and shapes. These standards are called hard disk *form factor*s and refer primarily to its external dimensions. The reason for standardizing on form factors is compatibility. Without these standards, hard disks would have to be custom-made to fit different PCs. By agreeing on standards shapes and sizes for hard disks--as well as standard interfaces of course--it is possible for any of the thousands of PC makers to purchase units from any hard disk manufacturer and know that there won't be problems with fit or form during installation.

Over the life of the PC there have only been a few different hard disk form factors. Since changing a form factor standard requires coordination from the makers of other components (such as the makers of system cases) there is resistance in the industry to change the standard sizes unless there is a compelling reason to do so. (For example, when laptop PCs became popular new, smaller drives were created to save space and power, important goals in the world of mobile computing.)

Form factors are generally described by a single metric. For example, the most common form factors today are "3.5-inch" and "2.5-inch". These numbers generally refer to the *width* of the drive, but they can be both vague and misleading (nice, huh? :^)) They usually were chosen for historical reasons and in typically were based on either the platter size of drives that use the form factor, or the width of drives using that form factor. Obviously a single number cannot represent both, and in some cases, it represents neither! For example, 3.5" hard disks are generally 4" wide and use 3.74" platters. :^) (The name in this case comes from the fact that the drives fit in the same space as a 3.5" floppy disk drive!) Much more about the relationship between form factors and platters can be found in the discussion of platter size in the media section. You will also find there a detailed description of the trend towards smaller platters in modern hard disks.



The five most popular internal form factors for PC hard disks. Clockwise from the left: 5.25", 3.5", 2.5", PC Card and CompactFlash.

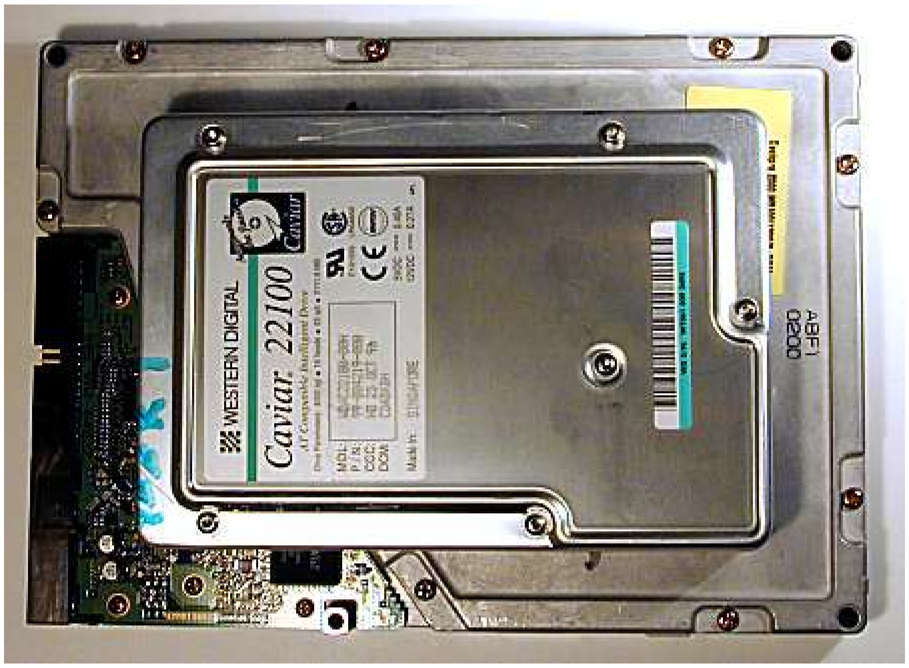
In this section I examine the major form factors that have been used for internal hard drives in PCs. This includes details on the dimensions of the form factor, especially the different heights associated with each. (Most form factors are actually a *family* of form factors, with different drives varying in the height dimension). In addition to the standard internal drive form factors, I briefly discuss external drives and also removable drive trays, which are sort of a "hybrid" of internal and external designs.

Note: There may be drives available in form factors other than those listed here; I believe I have them all but as always, could be wrong. :^)

Next: 5.25" Form Factor

5.25" Form Factor

The 5.25" form factor is the oldest in the PC world. Used for the first hard disks on the original IBM PC/XT back in the early 1980s, this form factor has been used for most of the PC's life span, but is now obsolete. The basis of the form factor is the 5.25" drive bay used in the first PCs for 5.25" floppy disk drives (themselves obsolete today). These bays still exist today in modern PCs, but are now used primarily for CD-ROM/DVD drives and similar devices, not hard disks. The 5.25" form factor was replaced by the 3.5" form factor for two main reason: first, 5.25" drives are *big* and take up a lot of space; second, 3.5" drives offer better performance; see the discussion in the section on platter sizes for an explanation. The use of 5.25" drives continued as late as the mid-1990s for high-end drives used in servers and other applications where the large size of the platters in these drive was needed to allow drives with high capacities to be created. They mostly disappeared from consumer PC many years prior to that.



A 3.5" form factor hard disk piggybacked on a 5.25" form factor hard disk to contrast their dimensions. The 5.25" drive here is a Quantum Bigfoot and is the same height as a regular 3.5" low profile drive.

5.25" drives generally use 5.12" platters and have a width of 5.75" and depth of 8.0". For many years they were found in only two different height profiles: *full-height*, meaning the same height as the floppy drive on the original PC and the bay it used (3.25"); and *half-height*, which is of course half that number. In the 1990s, Quantum launched a new line of 5.25" drives named the *Bigfoot* family, which reintroduced 5.25" drives to the consumer marketplace. These were sold as "economy" drives and due to the larger platter size, offered a lot of capacity--but due to slower spindle speeds and a "value line" design, not much performance. They were popular with many PC manufacturers but eventually were phased out. These drives used what had up to that point been non-standard heights for 5.25" drives, typically 1" high or less. Quantum calls these drives *low-profile* or *ultra-low-profile*. Here are the statistics and applications of the different profiles used in the 5.25" form factor:

Form Factor	Width (in)	Depth (in)	Height (in)	Application
5.25" Full-Height	5.75	8.0	3.25	All drives in early 1980s; Large capacity drives with many platters as late as the mid-1990s
5.25" Half-Height	5.75	8.0	1.63	Early 1980s through early 1990s
5.25" Low-Profile	5.75	8.0	1.0	Quantum Bigfoot, mid-to-late 1990s
5.25" Ultra-Low-Profile	5.75	8.0	0.75 - 0.80	Quantum Bigfoot, mid-to-late 1990s

Interestingly, despite the general trend to smaller drives, the drives that continued to use the 5.25" form factor through the late 1980s and early 1990s were more often found as full-height devices than half-height ones. This may be due to the fact that their niche became applications where a lot of storage was needed, so the ability to fit many more platters in that nice, roomy 3.25" high package was attractive.

Next: 3.5" Form Factor

3.5" Form Factor

The 3.5" form factor is *the* standard in the PC world today, and has been for about the last decade. Drives of this size are found almost exclusively now in modern desktop PCs, and even in servers and larger machines. The only major market where other form factors hold sway over 3.5" is that for laptops and other portable devices, where the reduced size of 2.5" and smaller form factors is important.

Like the 5.25" form factor before it, the 3.5" form factor is named not for any dimension of the drives themselves, but rather for the fact that they were designed to fit into the same drive bay as 3.5" floppy disk drives. 3.5" form factor drives traditionally have used 3.74" platters with an overall drive width of 4.0" and depth of about 5.75". In recent years, 3.5" form factor drives with platters smaller than 3.74"--in some cases much smaller--have appeared on the market. Most 10,000 RPM spindle speed drives reduce the size of the platters to 3", and the new 15,000 RPM Seagate drive has platters just 2.5" in diameter. The shrinking media size is done for performance reasons, but the 3.5" form factor is maintained for compatibility (these high-end drives are designed to go into expensive servers, not laptops!) For this reason, it is no longer the case that you can tell the size of a drive's platters by its form factor. See the discussion of platter size for more details.

3.5" form factor drives come in two general profiles: the larger is the so-called *half-height* drive, which is 1.63" in height. This name is kind of funny, since it is "half" of a height that never existed for 3.5" form factor drives. The name was derived from the fact that these drives are the same height as half-height 5.25" form factor drives, which are half the height of full-height 3.25" high drives in that form factor. Half-height 3.5" form factor drives are still used today, but only in servers and other high-end platforms. The standard for 3.5" is 1" height, which is commonly called *slimline* or *low-profile*, but just as commonly given no name at all and assumed as the default. The reason for the smaller size being the standard is that 1" is the height of a standard 3.5" floppy disk drive and 3.5" drive bay. In addition, there are some drives that are reduced in size from the 1" standard, using for example 0.75" height. Here are the standard profiles for 3.5" form factor drives:

Form Factor	**Width (in)**	**Depth (in)**	**Height (in)**	**Application**
3.5" Half-Height	4.0	5.75	1.63	High-end, high-capacity drives
3.5" Low-Profile	4.0	5.75	1.0	Industry standard, most common form factor for PC hard disks

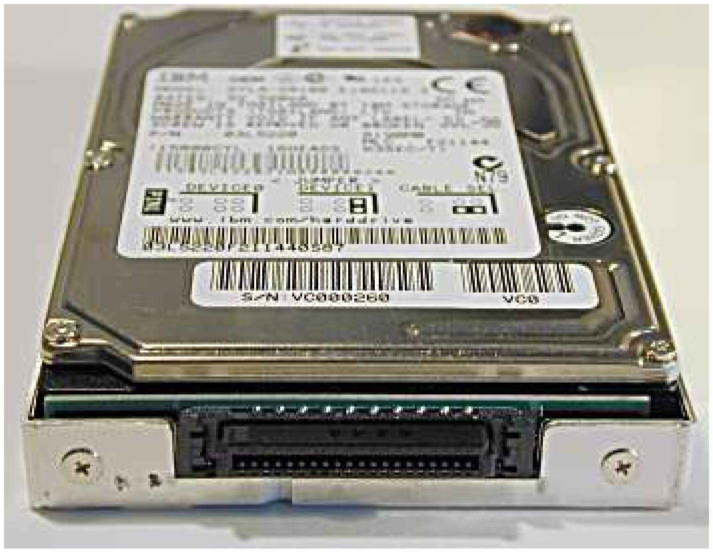
It is likely that the 3.5" form factor will continue to be the industry standard in PCs for years to come, due to the enormous installed base of systems that use this size, and no real compelling reason to change the form factor for the typical desktop machine.

Next: 2.5" Form Factor

2.5" Form Factor

2.5" form factor drives are the standard today for notebook computers (although not all notebooks use them, most do). Since the notebook market continues to grow by leaps and bounds, sales of 2.5" form factor drives have been increasing, on a percentage basis, faster than probably any other segment of the hard disk market overall. While older laptops originally used 3.5" drives, the move to 2.5" was done for several reasons that are very important to mobile PC users (you can also find related information on the reduction of platter sizes here):

- **Size Reduction:** Smaller drives take up less space and allow for laptops to be reduced in size. This trend began with the first 2.5" drives and continues with the continuous reduction in the heights of 2.5" drives (see below) and also the creation of still-smaller form factors.
- **Power Reduction:** Smaller drives use less power, important for PCs that run on batteries.
- **Enhanced Rigidity:** Smaller drives use smaller platters, which are less susceptible to damage as a result of shock, always a concern for a drive that will be moved around (often while operating!)



An 8.4 GB, 2.5" form factor IBM hard disk from my notebook. Note the single connector in the front, which is mated to a matching connector in the laptop's hard disk bay. This allows the drive to be easily replaced at a later time. The connector on the hard disk itself just uses straight pins like a 3.5" hard disk form factor drive; the drive is mounted into a carrier here, and the thin circuit board you can see in the front "adapts" the regular pin connector into the single Centronics-style connector my notebook uses.

Note: For more information on notebook hard disks, and a picture of a notebook drive's regular connector, see this page.

Unlike its larger, older siblings, the 2.5" form factor actually *is* named for the platter size of drives that use it (finally! :^)) The width of a 2.5" drive is 2.75", and depth is 3.94". These drives originally came in just one height (0.75" or 19 mm). Since for any storage technology level there is a tradeoff between size and capacity, over time several different heights were created in this form factor as standards for mobile PC users with different requirements. They are usually specified in metric (mm) and to my knowledge have no fancy names:

Form Factor	Width (in)	Depth (in)	Height (in)	Application
2.5" 19 mm Height	2.75	3.94	0.75	Highest-capacity 2.5" drives, used in full-featured laptop systems
2.5" 17 mm Height	2.75	3.94	0.67	Mid-range capacity drives used in some laptop systems
2.5" 12.5 mm Height	2.75	3.94	0.49	Low-capacity drives used in small laptops (subnotebooks)
2.5" 9.5 mm Height	2.75	3.94	0.37	Lowest-capacity drives used in very small laptops (mini-subnotebooks)

2.5" drives are pretty much entrenched as the standard for laptop machines. They are also used occasionally in industrial applications, where the smaller size and increased ruggedness of portable drives is important.

Next: PC Card (PCMCIA) Form Factor

PC Card (PCMCIA) Form Factor

One of the most popular interfaces used in the notebook PC world is the *PC Card* interface, also sometimes called *PCMCIA* after the group that created it in the late 1980s. This interface standard was created to allow for easier expansion of notebook systems, which at that time had very few options for adding hardware at all. For information on the PC Card interface as it relates to hard disks, see this page.

Despite the relatively small size of cards that adhere to the PC Card standard, hard disk engineers have managed to create hard disks to fit. There are actually three PC Card form factor sizes, defined by the PCMCIA in 1995. The width and depth of these devices is exactly the same as that of a credit card, which I am sure is not a coincidence! They are all 2.13" wide and 3.37" deep. The three sizes differ only in their height. Type I devices are 3.3 mm thick; Type II devices are 5.0 mm thick, and Type III devices are 10.5 mm thick. Originally, the intention was for solid state devices like memory, modems and the like to use the Type I and Type II devices, while the Type III devices were for hard disks. Due to the extreme height limits of PC Cards, it is difficult to make hard disks that will fit into the allowed space, and most PC Card hard disks are Type III. However, advances in miniaturization have allowed some companies to now make hard disks that actually fit into the Type II PC Card form factor as well. Since most laptops can only accept *either* two Type I/II cards or a single Type III, this is a significant advantage. Here's a summary table of the different sizes:

Form Factor	Width (in)	Depth (in)	Height (in/mm)	Application
PC Card Type I	2.13	3.37	0.13 / 3.3	Not used for hard disks (yet?)
PC Card Type II	2.13	3.37	0.20 / 5.0	Smaller-capacity expansion hard disks for laptops and consumer electronics
PC Card Type III	2.13	3.37	0.41 / 10.5	Higher-capacity expansion hard disks for laptops

The 2.13" width of this form factor puts a hard limit on the platter size of these drives--even 2.5" platters are too large. Most PC Card drives today use 1.8" platters. Interestingly, the first hard drive to use the PC Card form factor was probably the Hewlett Packard *Kittyhawk* drive, with much smaller 1.3" platters. This drive is a good example of a technology being "ahead of its time". It was actually introduced way back in 1992, very early on for such miniaturized technology. Unfortunately, at the time the market may not have been big enough to provide HP with sufficient revenues to keep making it. The Kittyhawk was used in early hand-helds and other small consumer electronic devices (even printers!) for a while, but was eventually discontinued, and HP is no longer making hard disk drives of any sort. If this technology had been introduced five years later, it may have been a runaway

success; certainly IBM is having great success with its slightly-smaller Micro drive.

A CompactFlash card (left) and a PC Card (right). The quarter is included for size context. Neither of these is a hard disk (though the SanDisk is a solid-state flash card "hard disk") but are the same size and shape as hard drives of their respective form factor.

Note: Many companies also make "solid state drives" using the PC Card form factor. These perform the same function as hard disk drives, but are not hard disks at all: they are actually flash memory, a type of ROM. I discuss this in more detail in the section on the CompactFlash form factor.

Interestingly, with a couple of exceptions, most of the smaller PC Card hard disks are not made by the bigger, well-known hard disk companies, but rather smaller niche companies. I am not sure what the reason is for this. I suspect that there just may not be enough profit potential there for the big names to bother with this market, which is small compared to the market for mainstream PC drives.

Next: CompactFlash Form Factor

CompactFlash Form Factor

In much the same way that the need for expansion capabilities in laptops led to the creation of the PCMCIA and PC Card devices, a consortium of electronics and computer industry companies in 1995 formed the *CompactFlash Association* to promote a new form factor called, of course, *CompactFlash* (abbreviated *CF* or *CF+*). This form factor is similar to the PC Card form factor, but amazingly enough, even smaller. CF cards are intended to be used not in laptop PCs but smaller electronic devices such as hand-held computers, digital cameras and communications devices (including cellular phones).

Unlike the PC Card standard, which is used for a wide variety of devices, CompactFlash is primarily designed around permanent storage. The "flash" in "CompactFlash" is from the primary technology used in these cards: flash memory. Flash memory is really electrically-erasable read-only memory, typically used in regular PCs only for holding the motherboard's BIOS code. The word "flash" refers to the ability to write and erase these ROMs electrically. Much the way you can "flash" your motherboard BIOS to update it, these flash memory storage cards have controllers in them that do this as part of their normal operation. Unlike regular memory, flash memory is of course non-volatile and retained when the power is removed.

The intention of the CompactFlash form factor was to allow consumer electronic devices to use these CompactFlash cards for their equivalent of a hard disk. Since the flash memory is not volatile, it does perform the same general function as a hard disk. Like PCMCIA devices, variants of the form factor were developed, differing only in thickness; the thicker cards provide more space to pack in additional flash memory chips for greater capacity. Here are the dimensions of the two types of CompactFlash cards:

Form Factor	Width (in)	Depth (in)	Height (in/mm)	Application
CF+ Type I	1.69	1.42	0.13 / 3.3	Smaller-capacity flash cards for digital cameras, hand-held computers and consumer electronics; not used for hard disks (yet)
CF+ Type II	1.69	1.42	0.20 / 5.0	Larger-capacity flash cards and hard disks for digital cameras, hand-held computers and consumer electronics

As you can see, the CF form factors are very small: so small that they were probably never designed with the thought that anyone would make a true hard disk using them. The engineers at IBM however had a different idea! In 1999, while the makers of regular flash memory cards were struggling to reach 64 MB capacity, IBM introduced the Microdrive, a true hard disk that fits into the small confines of the CF form factor. The original Microdrive was available in either 170 MB or 340 MB capacities, which is pretty impressive considering that the drive uses a single 1" platter... even more impressive is the new Microdrive released in 2000 with a whopping 1 GB capacity! The Microdrive uses the CF Type II format, which is 5.0 mm thick. No current hard disks are made for the even smaller Type I size (only 3.3 mm thick) but I have heard rumors that at least one company is working on it, and I'd be surprised if IBM themselves didn't have something brewing in this regard also. Pretty cool.

IBM's amazing Microdrive.

Image © IBM Corporation
Image used with permission.

Smaller drives generally have less performance than full-sized ones and the Microdrive is no exception; it certainly doesn't compete with the newest 2.5" or 3.5" form factor drives for performance (though in many ways it is superior to flash memory chips). For its size it is certainly no slouch, though: it has a 3,600 RPM spindle speed, and very decent maximum areal density of as much as 15.2 Gbits/in^2. (The original generation actually used a faster 4,500 RPM spindle; this was probably lowered to reduce power consumption and heat, both very serious issues for this form factor... however, the first Microdrives also had only one-third the areal density of the 1 GB model.) The extremely small size of the drive allows it to spin up to full speed in only half a second, a fraction of the time required by most large drives. This lets the Microdrive power down often when idle to save power, an essential feature for devices that use small batteries.

Tip: If you want to use the Microdrive in a laptop, an inexpensive adapter is available to convert it to the PC Card type II form factor.

As areal density continues to increase these drives will only continue to increase in capacity. Over the next few years they will battle for sales with flash memory cards; this market is likely to grow as the number of hand-held PCs, digital cameras and other electronic gizmos needing lots of storage continues to increase dramatically. Next: Form Factor Comparison

Form Factor Comparison

For ease of comparison, the summary table below lists all of the standard internal hard disk form factors with their dimensions, typical platter sizes found in hard disks that use the form factor, and common applications:

Form Factor	Profile	Platter Size (in)	Width (in)	Depth (in)	Height (in)	Application
5.25"	Full-Height	5.12	5.75	8.0	3.25	All drives in early 1980s; Large capacity drives with many platters as late as the mid-1990s
	Half-Height	5.12	5.75	8.0	1.63	Early 1980s through early 1990s
	Low-Profile	5.12	5.75	8.0	1.0	Quantum Bigfoot, mid-to-late 1990s
	Ultra-Low-Profile	5.12	5.75	8.0	0.75 - 0.80	Quantum Bigfoot, mid-to-late 1990s
3.5"	Half-Height	2.5, 3.0, 3.74	4.0	5.75	1.63	High-end, high-capacity drives
	Low-Profile	2.5, 3.0, 3.74	4.0	5.75	1.0	Industry standard, most common form factor for PC hard disks
2.5"	19 mm Height	2.5	2.75	3.94	0.75	Highest-capacity 2.5" drives, used in full-featured laptop systems
	17 mm Height	2.5	2.75	3.94	0.67	Mid-range capacity drives used in some laptop systems
	12.5 mm Height	2.5	2.75	3.94	0.49	Low-capacity drives used in small laptops (subnotebooks)
	9.5 mm Height	2.5	2.75	3.94	0.37	Lowest-capacity drives used in very

						small laptops (mini-subnotebooks)
PC Card	**Type I**	1.8	2.13	3.37	0.13	Not used for hard disks (yet?)
	Type II	1.8	2.13	3.37	0.20	Smaller-capacity expansion hard disks for laptops and consumer electronics
	Type III	1.8	2.13	3.37	0.41	Higher-capacity expansion hard disks for laptops
Compact Flash	**Type I**	1.0	1.69	1.42	0.13	Smaller-capacity flash cards for digital cameras, hand-held computers and consumer electronics; not used for hard disks (yet)
	Type II	1.0	1.69	1.42	0.20	Larger-capacity flash cards and hard disks for digital cameras, hand-held computers and consumer electronics

Next: External Hard Disks

External Hard Disks

The vast majority of hard disks are internal, which means they are designed to be mounted inside a PC, and hidden from the user. This is why they have a rather "rough" outside appearance, with the logic board exposed, etc. (For some reason I always liked the way internal drives looked... maybe just because I'm such a storage freak. :^)) The fact that they are intended to be installed internally is also the reason why they come in standardized form factors--so they will fit inside the PC case in predictable ways. :^)

Some hard disks are available as external drives, especially ones using the SCSI interface. These really don't differ much from internal drives, except that they include an additional outer plastic shell, a power supply to run the disk, and of course, a larger price tag. :^) They do offer some advantages over internal drives: more expandability, easier installation, usually better cooling, and also interoperability with other systems that use SCSI. Since they are external, they don't have to be made in standardized form factors. At one time, these SCSI external drives were more popular than they are today. The higher cost of these drives due to the duplication of support hardware, and the extra space required to put them "somewhere" are probably the main reasons why they are less common today than they once were.

The IBM Travelstar E, an external hard disk using a PC Card interface card.

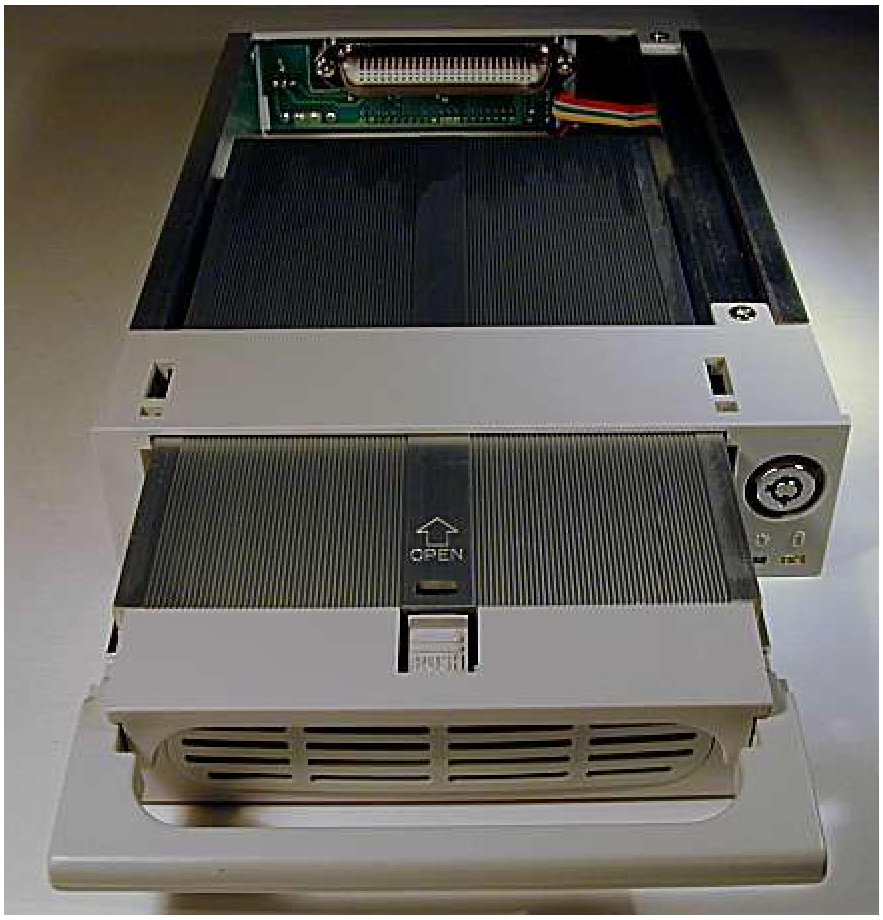
Image © IBM Corporation
Image used with permission.

External drives have found a new market role of sorts as expansion and backup devices for portable computers. Many varieties are now available using either the parallel port or the PC card interface. In the latter design, the hard disk is in an external enclosure, with an interface cable that runs to a PC card. The card connects to the laptop through a PC card slot. The fact that the hard disk is not constrained by the physical limits of the small PC card slot means it can be made much larger than the small drives available in the PC card form factor, while retaining the portability advantages of the PC card interface. Some companies also make just the enclosure itself, with the PC card interface; you supply your own standard internal hard disk. This can save money by letting you "recycle" an older internal drive, and gives you the flexibility of swapping in new drives as better technology becomes available.

Next: Removable Hard Disk Trays

Removable Hard Disk Trays

An interesting compromise between internal and external hard disks is the *removable hard disk drive tray*. A tray is installed into a standard PC case drive bay that allows regular internal hard disks to be placed into it. You then can swap the internal hard disk with another one without opening up the case, allowing you to use hard disks as a removable storage medium. In a way, the concept is similar to the way a removable car stereo is designed. These trays are also commonly called *mobile racks*, *drive caddies* and several other names.



Shown above is the *Kingwin KF-21-IPF mobile rack system* that I use on my work desktop PC for backups and file archiving. The drive fits into the removable tray (bottom) which fits into the stationary docking station (top). The
stationary portion is installed into a standard 5.25" drive bay and connected to
a regular IDE/ATA cable. On the right-hand side you can see the lock that secures the tray in place, as well as indicator lights for power and drive activity.

For certain applications, this are the ideal removable storage device: it uses regular hard disks, which are very fast, highly reliable, very high capacity and very inexpensive in terms of dollars per byte of storage. They can be used for backup purposes (secondary to a regularly-installed standard internal hard disk)--see here for more information on this idea. Removable trays can also be used to allow the use of different operating systems on the same PC (though there are simpler ways to do this) and to allow different people to use the same PC while keeping their data and programs totally separate.

If you decide to use a mobile rack system, be sure to check out the specifications of the unit you are considering carefully. Different models come with support for different speed drives, some are made primarily of metal and others of plastic, and so on. Metal units will provide better cooling than plastic ones. Some also come with an integrated cooling fan, which is a good idea especially for higher-RPM drives.

Warning: If you decide to set up this sort of arrangement with your PC, remember that regular internal hard drives are designed under the assumption that they will be installed inside the PC and left there. Be sure to handle them properly, and especially, to observe ESD precautions.

Next: Hard Disk Packaging and Mounting

Hard Disk Packaging and Mounting

This section discusses issues related to how the drive is packaged and mounted into the system. Packaging and mounting considerations are vital to any consideration of the reliability of a drive, due to its very sensitive components. This includes a look at the exterior of the hard disk, a discussion of how the drive is sealed against contamination from the outside air, and how drives should be oriented for maximum reliability.

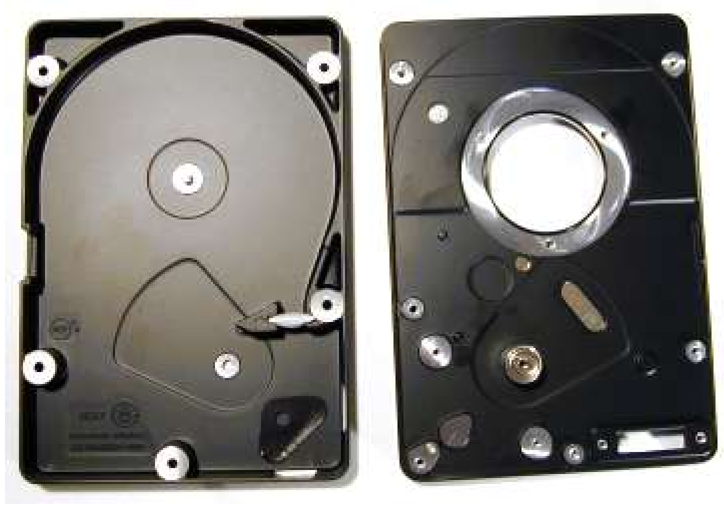
I also discuss here some issues related to drive handling, and also matters of cooling that are relevant to newer, faster drives. Finally, a discussion is provided on the differences between OEM and retail hard disks, a confusing matter that leads to many questions by hard disk buyers.

Next: Base Casting and Top Cover

Base Casting and Top Cover

The entire hard disk is mounted into a physical enclosure designed to protect it and also keep its internal environment separated from the outside air. This is necessary because of the requirement of keeping the internal environment free of dust and other contamination that could get between the read/write heads and the platters over which they float, and possibly lead to head crashes.

The bottom of the disk is often called the *base casting*, the name coming from the manufacturing process used to create the single piece of aluminum from which it is normally made. The drive mechanics are placed into the base casting, and another piece of usually aluminum is placed on top to enclose the heads and platters. A rubber gasket is placed between the base and cover to ensure a tight seal. On some drives, a metallic tape seal is applied around the perimeter of the drive to fully enclose the drive. The exact shape of the base and cover can vary significantly from drive to drive. Some models have the base flat and the cover like a bowl that goes over the contents; some are the opposite way.


Cover (left) and base casting (right) of a consumer-grade IDE/ATA hard disk drive. All the components have been removed to show the detail (except the recirculating filter is still in place on the cover... Oops! You can see a proper picture of that filter here.)

The base and cover are attached using a number of small screws, usually around the perimeter of the cover. Additional screws are also used in the middle of the cover; one to stabilize the spindle motor shaft, and one to secure the axis of the actuator assembly. Normally all of these are *Torx* screws (star-shaped). The reason these screws are usually found on hard disks is that screwdrivers for them are usually *not* found at your local hardware store--they have to be special-ordered (as I discovered when illustrating this material. :^)) The idea of course is to discourage tampering with the hard disk.

The entire contents of the base and cover chamber (including the platters, heads and actuator components) are collectively called the *head-disk assembly*. You should never open the assembly. If you do, you will quickly contaminate the heads and platters, and eventually ruin the drive. (Some folks get a bit melodramatic about this, claiming the drive will instantly die as soon as you try to run it with the cover off. That's not true, but if you do open the drive's case it's just a matter of time.) You will also void the warranty of the drive if you try to open it up, as the warning labels on the drive itself will attest. In fact, it is common for the hard disk manufacturers to hide one or more of the screws under a label saying "Warranty void if removed".

The logic board is normally mounted on the bottom of the base casting, exposed to the outside. It is separated from the base casting using foam or other cushioning material. Of course, the read/write heads must be linked somehow to the logic board. This is accomplished either with a flexible ribbon cable that runs from the logic board through a gasket and into the hard disk chamber, or a set of pins that goes through a hole in the base casting, mating to a special connector inside the head-disk assembly that connects to the heads.

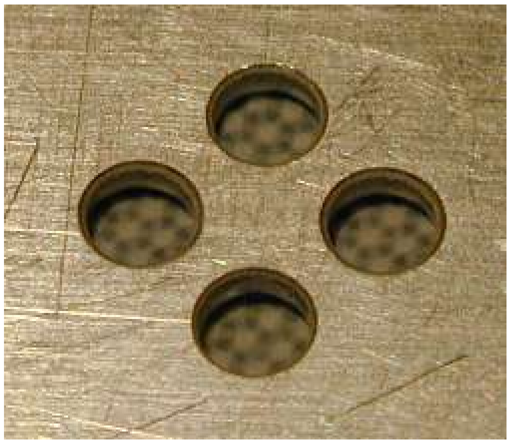
Some manufacturers apply a coating to the inside of the base casting before assembling the drive. The purpose of this coating is to seal in, against the walls of the casting, any minute particles of dirt or metal that might be hiding in the casting and might later dislodge and end up damaging the disk.

Next: Air Circulation and Air Filtration

Air Circulation and Air Filtration

As discussed in the section describing hard disk head flying height, the old myth about hard disk heads and platters being "in a sealed vacuum" or whatnot, is wrong in two ways: there's no vacuum in there, and the drive itself isn't *sealed*, at least not in the way most people believe. In fact, air is an essential component for proper drive operation. Regular hard disks aren't totally sealed from the outside air, but they definitely are separated from it, in order to ensure that the dirt and dust of the outside air is kept away from the delicate platters and heads of the drive. If foreign matter were to get onto the platters--even something as small as smoke particles--a head crash or other problems could easily result.

Hard disks aren't sealed, because they have to be able to pass air between the inside of the drive and the outside, in order to equalize any air pressure differential that may exist between the two environments. This allows the disk to maintain proper equilibrium when the weather changes, or the drive is moved to a different altitude; if pressure is not balanced the drive might not perform properly and damage could even result. You can actually see the small *breather holes* in the cases of many drives, placed there for this purpose. Of course just putting a hole in the case would cause contamination of the drive, so the holes are covered with a *breather filter* which lets air pass through slowly but not dirt or dust. These filters are placed permanently and do not need to be serviced or replaced.



Closeup shot of the breather holes in the top of a hard disk case. Part of the breather filter can be seen just under the holes.

Hard disks also have an internal air *flow* within their sealed chambers (caused by the rotation of the platters--there is no fan inside a hard disk). This air flow is used to continuously filter the air within the disk assembly. Despite building the hard disks in ultra-clean facilities and taking other precautions during manufacturing, a small *recirculating* filter is built into the drive itself as an added security measure. This filter is designed to work on the air that flows within the hard disk assembly, catching any minute bits of debris that might somehow make it inside. This reduces the chances that such dirt will end up on the disk platters. Like the breather filter, the recirculating filter is not changeable, nor does it need to be.

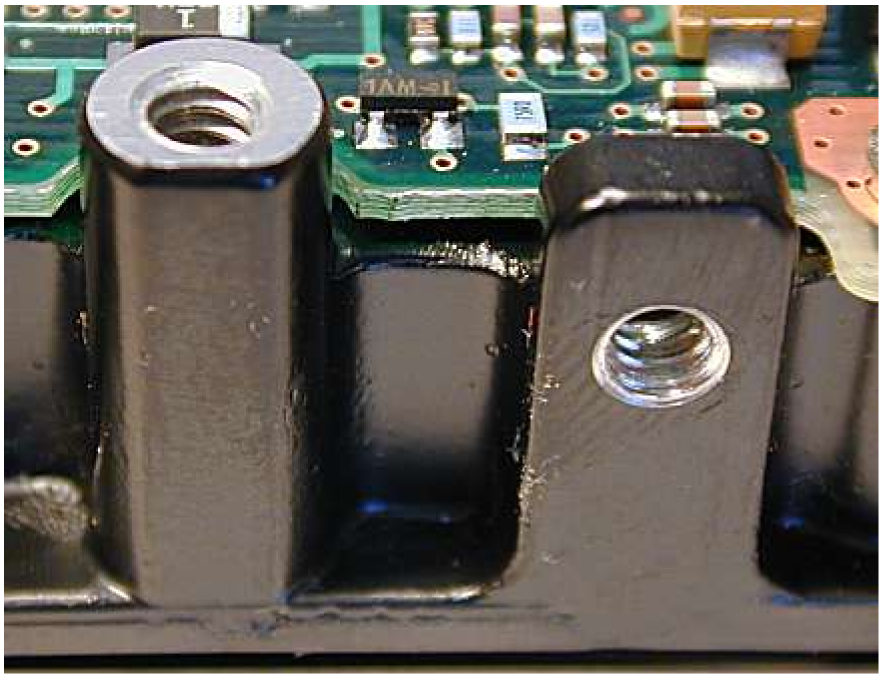
A recirculating filter in the top cover of a consumer-grade hard disk.

Next: Orientation and Mounting

Orientation and Mounting

Orientation refers to how the hard disk drive is physically installed into the PC. In the majority of cases the drive is installed in the "default" way: flat, with the drive parallel to the ground, the logic board facing down and the drive label facing the sky. Some PC cases, however, especially the smaller desktop boxes, have side-mounted hard disk bays. Either of these orientations is common and should present no problems. In fact, most manufacturers say that you can mount their drives any way you wish, even upside-down if it is necessary. That said, my personal opinion--one shared by at least one manufacturer--is that you should avoid strange mounting positions if it all possible. The reason is simple: most testing of drives is done with them in either of the standard orientations. Hard drive manufacturers don't do nearly as much testing of drives mounted upside-down as they do of drives right-side up. For the sake of reliability, it's best to keep things simple if at all possible, and well, go with the flow.

Older hard disks that used stepper motor actuators were much more sensitive to orientation. It was in fact often recommended that the low-level formatting of the disk always be done after the disk was installed in its operating position, to avoid any shifting of the location of the tracks that might be caused by changing the orientation. Since hard disks today are much more solidly built, and they use voice coil actuators for dynamic head positioning, this really is not an issue. A modern hard disk can be side-mounted in the case without any problems, and can also have its orientation changed after it has been in use for some time.



Mounting holes on a SCSI hard disk, viewed from the bottom of the drive. The one at left is part of the set of holes used if putting the mounting screws into the bottom of the drive; the one at right is for side mounting.

Most hard disks have screw holes built right into the aluminum base of the disk itself. They can usually be mounted by screwing into the bottom of the base, or the side. There are some caveats here. First, make sure you use small screws, preferably the ones that came with the hard disk--using ones that are too long can in theory damage the drive. Second, since the circuit board is exposed, it must be carefully watched to ensure that there is no chance that it will be contacting anything when the drive is installed, or the logic board could be shorted out.

Next: Drive Rails and Mounting Kits

Drive Rails and Mounting Kits

The simplest way to mount hard disks into the system case is the direct mounting method, where the disk is screwed directly into a drive bay. This has largely replaced the use of *drive rails*, which were common in older PCs. Drive rails are small metal strips which are screwed onto the drive and then used to slide the drive into the case. Some people like these, others hate them; they are in fact still used by some case manufacturers. Some cases also have drive *cages*, which are removable brackets into which the drive can be mounted. These different mounting techniques are described in detail on this page in the system case section.

Most hard disks used in PCs today are 3.5" form factor drives, designed of course to go into 3.5" drive bays. Pretty much all newer cases include these bays. However, there are some older cases that do not have 3.5" drive bays at all. To allow 3.5" drives to be used in the larger 5.25" drive bays, mounting kits are available. These are mechanical adapters that screw into the side of the drive to "widen" it to the size that a 5.25" drive would be, so it can fit into a 5.25" drive bay. You can find more information on these adapters (including a picture) on this page.

At one point these mounting kits were often included in retail-boxed hard disks. Today they are much less common, simply because it's rare to find a PC still in use today that doesn't have a 3.5" drive bay. They should still be available for purchase separately for a few dollars from stores that sell PC accessories.

Next: Temperature Limits and Drive Cooling

Temperature Limits and Drive Cooling

Hard disks can only be counted upon to perform reliably if they are kept within their specified temperature ranges. Cooling has become one of the most important considerations for reliability, particularly for newer drives running at high spindle speeds. See this discussion of cooling in the section on hard disk quality for more.

For the first 15 years of the PC's existence, hard drives were always passively cooled: the metal case of the drive itself was used to sink most of the heat generated inside it. Some of the heat was conducted to the metal insides of the system case, and the air flow within the case took care of the rest. Overheating problems were rare. Cases were generally large and better-ventilated than the average PC is today, and in fact before the mid-to-late 1990s, few people other than those designing large servers gave much thought at all to cooling disk drives. You put it in the case, made sure the power supply fan was working properly, and that was pretty much all there was to it.

This situation changed with the introduction of the first 7,200 RPM drives. These drives, with their faster more powerful spindle motors, generated more heat than had been seen before in hard disks. In fact, heat issues are common with each first-generation drive family using a newer, faster spindle speed (hard drive engineers generally do a good job of working on the heat issues with subsequent revisions of each design). The end result is that for some 7,200 RPM and 10,000 RPM drives, just sticking the drive in the case and hoping for the best is not sufficient.

When trying to keep a hot drive within operating parameters, the most important first step is to address the cooling of the case overall. It's essential that the fan(s) are functioning properly and have sufficient capacity for the case. The PC must not be operated in a very hot room or placed where it will be excessively heated (for example, in direct sunlight or right next to a heater). The case should not be too small for the number of devices it contains. Hard drives should also be spaced to allow proper air flow over them--putting two drives very close together is not a good idea. See this discussion of system cooling for more on this subject.

If the system's cooling isn't sufficient for a hot-running hard disk, you have some options. Various manufacturers make devices specifically for cooling hard disks. They usually come in two basic forms:

- **Drive Coolers:** These are essentially fan and heat sink combos, similar to those used for CPUs, but designed specially to be used with hard disks. They are attached to the drive using thermal conductive tape, and blow air directly onto the drive case to cool it.

A drive cooler mounted on top of a standard 3.5" form factor hard disk. The power plug is visible at right.

Original image © PC Power & Cooling, Inc. Image used with permission.

- **Bay Coolers:** These devices are similar in design to the drive bay adapters available for mounting a 3.5" form factor drive into a 5.25" drive bay, except that they add cooling for the drive. The disk is mounted into the cooler, which contains one or more integrated fans. The cooler then is mounted into one of the larger 5.25" drive bays found in most PCs.

A 5.25" bay cooler with a 3.5" form factor hard disk installed in it. This view is from the inside of the case; the external faceplate is at rear.

Original image © PC Power & Cooling, Inc. Image used with permission.

In both designs power is provided for the fan(s) through the use of a standard drive connector.

Tip: Simply mounting a 3.5" form factor hard disk into a 5.25" drive bay with a metal adapter kit will help to cool the drive in some cases by providing two additional metal pieces adjacent to the drive to act as small heat sinks.

Do you need active cooling for your hard disk? There's no easy answer to that question: it depends on a number of different factors. Most PC users do not need to add special coolers for their hard disks. This is especially true of consumer IDE/ATA drives--since manufacturers know most people pay little attention to cooling, they must assume no special cooling when selling these drives through retail channels, or they'd end up having to deal with a flood of warranty replacements. For higher-speed SCSI drives, additional cooling may be required. You will have to determine the need by assessing the cooling level of your system, the temperature requirements and heat generation level of the particular drive, how many drives are going to be put into the system, and similar factors. If you are buying a pre-made system, the system manufacturer should be dealing with these design issues for you.

Next: Retail and OEM Packaging

Retail and OEM Packaging

Most hard disk drive models are sold as two different packages: *OEM drives* and *retail drives* (sometimes called *retail kits*). Retail drives are of course drives that are distributed to retail stores and online dealers for sale to the general public. OEM drives are those sold to system manufacturers in large quantity, for inclusion in PCs built for resale: "OEM" stands for "original equipment manufacturer" and refers to a company that makes PCs (or other equipment). See this general discussion of OEM and retail parts in The PC Buyer's Guide for more.

Retail packaged hard disk drives normally include the following (though this varies from manufacturer to manufacturer and model to model):

- **Hard Disk Drive:** The hard disk drive itself, in an anti-static bag or similar package.
- **Installation Instructions:** Instructions on how to configure and install the hard disk.
- **Drivers and/or Overlay Software:** A floppy disk or CD-ROM containing any necessary drivers or utilities, and usually, a copy of that manufacturer's version of drive overlay software for working around BIOS capacity problems in older systems.
- **Mounting Hardware:** A set of appropriately-sized screws for mounting the drive into the system case.
- **Interface Cable:** A cable of the correct type for the drive's interface.
- **Warranty Card:** A card describing the warranty provided on the drive, usually three or five years in length.
- **Pretty Box:** A very colorful box that looks nifty and holds all of the above. :^)

In contrast, OEM drives typically contain the following:

- **Hard Disk Drive:** The hard disk drive itself, in an anti-static bag or similar package.
- **Jumpers:** One ore more jumpers needed for configuring the drive.

And that's it (heck, even the jumpers aren't a sure thing. :^)) The reason that OEM packaging is so "plain" is that most OEMs do not need the additional support materials and packaging required for a proper retail package--they are just going to put the drives into PCs, not resell them to end users. If you are SuperPeeceez Inc. and make 10,000 systems a month, you just want 10,000 drives--not 10,000 fancy boxes, 10,000 warranty cards, 10,000 driver disks, and so on. So the drive manufacturers just ship OEMs lots of bare drives, with the minimum required to protect them, and that's all. By skipping all the extras, the OEM is able to buy the drives at a lower price from the manufacturer (and the fact that they are buying them by the thousand certainly helps with price as well!)

Western Digital Hard Drive
Western Digital Hard Drive
WD Caviar
Quick Install
Hardware Installation
Western Digital Hard Drive
5400 RPM
Ultra ATA/66
The World's Most Recommended Hard Drive™

Contents of a Western Digital hard disk retail box kit.

Original image © Western Digital Corporation Image used with permission.

Originally, OEM drives were available only to, well, OEMs. If you needed a hard disk for your PC you bought it in a retail box and that was that. A few years ago however, OEM drives began to appear for sale to individuals and end-users. They usually enter the market either by an OEM selling off extra drives to another company that then offers them to the public, or increasingly these days, by companies that buy them in bulk specifically for resale. Like the OEMs, many experienced PC home-builders and upgraders realized they don't need most or even all of the goodies in a retail package, and preferred the lower price of the OEM drives. Many companies quickly added OEM drives to their product lines to fill this demand.

Warning: In most cases, the retail and OEM versions of a given drive are identical, however this is not always the case. Sometimes a manufacturer will make slightly different versions of the same drive for the OEM and retail channels. A common difference is having a larger cache on the retail drive than on the OEM drive.

There's nothing wrong with buying OEM drives if you can get a significantly better price and don't need the items included in the retail package. However, you must be sure of what you are buying. In some ways the most important thing in that retail box is the warranty card, because there can be serious warranty consequences when purchasing OEM hard disks made by certain manufacturers. See this discussion of warranty coverage on OEM and retail drives for details. Ironically, in many cases OEM drives are no cheaper than the equivalent retail versions with full warranty, and in some cases, I have seem retail boxes go for less than plain-wrapped OEM drives!

Warning: When buying a hard disk mail order, be 100% certain of what you are purchasing. As always, most online retailers are honest but there are always a couple of bad apples who will send you an OEM drive after you ordered a retail one. Sometimes the salespeople at these companies don't really understand the difference between OEM and retail and may not really understand the warranty issues involved. Shop with care, and send back for exchange anything you are sent that isn't what you ordered.

Next: Hard Disk Handling

Hard Disk Handling

Hard disks are very delicate and sensitive instruments. While certainly it is true that newer (and especially *smaller*) drives are less fragile than their older counterparts, it is also true that all hard disk drives have to be properly handled to avoid damage. In most cases, handling of drives is something that happens very little anyway: you get the drive, you install it and you leave it there. So for most people handling isn't too much of an issue. For those that handle drives a great deal however, proper handling technique is essential.

The first thing you will notice when you deal with hard disks (and most other computer components) is that they are always transported in an *anti-static bag*. This is of course to prevent the damage that can occur to the hard disk's circuits as a result of *electrostatic discharge* or *ESD*. This is such a common problem when dealing with components that I put a general warning about it right in the Site Guide.

A "new" (well, actually a few years old, but unused) hard disk in its original anti-static bag.

Seagate has actually come up with a neat improvement on the standard anti-static bag (which has been around for years and years). They call it the *SeaShell* (ha ha, good one guys) and instead of being a thin plastic bag, it's a solid plastic clam-shell case that not only provides ESD protection for the drive, but physically protects it against shock as well. Instead of being a smoky gray, the SeaShell is clear plastic, so you can examine the drive without having to open up the package. And these little cases are also both recyclable and easily reusable. I hope other hard drive makers start using similar devices.

A Seagate "SeaShell", containing a Cheetah SCSI drive.

Original image © Seagate Technology
Image used with permission.

Be sure to ground yourself to a metal object before removing a hard disk from its ESD protection. Handle the drive as little as possible, and avoid bumping or jarring it if at all possible. The safest thing to do is to simply get it installed as quickly as possible. Since the logic board on internal drives is exposed, make sure none of its components contact anything metallic that could cause a short or a static discharge.

When shipping a hard disk drive, it is *essential* that you properly package it. Slapping the unit in a box with loose foam "peanuts" will not cut it for a hard disk drive. The drive should be properly supported on all sides with solid foam or other similar padding, and fitted properly in a box of the correct dimensions. Hard drive makers use boxes with special inserts specifically designed for shipping drives; if possible, it is best to use these boxes and inserts. When performing warranty exchanges, hard drive manufacturers will commonly send the replacement drive in one of these packages, and ask you to return the defective drive to them in the same box. This is the best way to ship a drive back to a manufacturer for exchange or repair.

Warning: Some companies may dispute your warranty claim for service or replacement if you ship them a drive that has been grossly under-packaged.

Next: Hard Disk Geometry and Low-Level Data Structures

Hard Disk Geometry and Low-Level Data Structures

Hard Disk Geometry and Low-Level Data Structures

When I first wrote The PC Guide in 1997, this page mentioned how each platter of a hard disk was "capable of storing in excess of one billion bytes of information". As I revise this section in 2000, leading-edge hard disks now pack a whopping 20 GB of storage per platter in the same amount of space. Pretty amazing.

Of course, this trend is only going to continue, with new drives having more and more data in the same space. In order to use all this real estate to best advantage, special methods have evolved for dividing the disk up into usable pieces. The goals, as usual, are two-fold: increasing capacity and increasing performance. This section takes a detailed look at how information is encoded, stored, retrieved and managed on a modern hard disk. Many of the descriptions in this section in fact form the basis for how data is stored on other media as well.

Next: Data Encoding and Decoding

Hard Disk Data Encoding and Decoding

Digital information is a stream of ones and zeros. Hard disks store information in the form of magnetic pulses. In order for the PC's data to be stored on the hard disk, therefore, it must be converted to magnetic information. When it is read from the disk, it must be converted back to digital information. This work is done by the integrated controller built into the hard drive, in combination with sense and amplification circuits that are used to interpret the weak signals read from the platters themselves.

Magnetic information on the disk consists of a stream of (very, very small) magnetic fields. As you know, a magnet has two poles, north and south, and magnetic energy (called *flux*) flows from the north pole to the south pole. Information is stored on the hard disk by encoding information into a series of *magnetic fields*. This is done by placing the magnetic fields in one of two polarities: either so the north pole arrives before the south pole as the disk spins (N-S), or so the south pole arrives before the north (S-N). This is discussed in detail where the read/write heads are described.

Although it is conceptually simple to match "0 and 1" digital information to "N-S and S-N" magnetic fields, the reality is much more complex: a 1-to-1 correspondence is not possible, and special techniques must be employed to ensure that the data is written and read correctly. This section discusses the technical issues involved in encoding and decoding hard disk data.

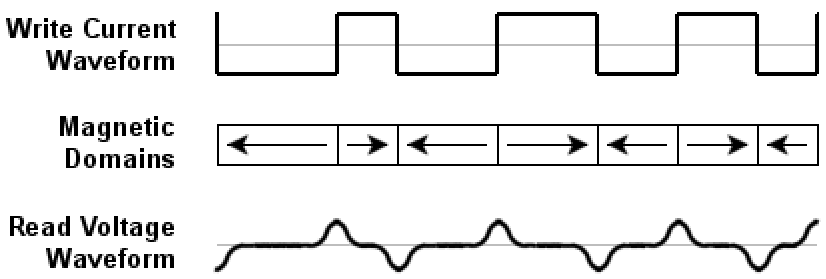
Next: Technical Requirements for Encoding and Decoding

Technical Requirements for Encoding and Decoding

You might think that since there are two magnetic polarities, N-S and S-N, they could be used nicely to represent a "one" and a "zero" respectively, to allow easy encoding of digital information. Simple! Well, that *would* be nice, but as with most things in real life, it usually doesn't work that way. :^) There are three key reasons why it is not possible to do this simple 1-to-1 encoding:

- **Fields vs. Reversals:** Read/write heads are designed not to measure the actual polarity of the magnetic fields, but rather *flux reversals*, which occur when the head moves from an area that has north-south polarity to one that has south-north polarity, or vice-versa. The reason the heads are designed based on flux reversals instead of absolute magnetic field, is that reversals are easier to measure. When the hard disk head passes from over a reversal a small voltage spike is produced that can be picked up by the detection circuitry. As disk density increases, the strength of each individual magnetic field continues to decrease, which makes detection sensitivity critical. What this all means is that the encoding of data must be done based on flux reversals, and not the contents of the individual fields.
- **Synchronization:** Another consideration in the encoding of data is the necessity of using some sort of method of indicating where one bit ends and another begins. Even if we could use one polarity to represent a "one" and another to represent a "zero", what would happen if we needed to encode on the disk a stream of 1,000 consecutive zeros? It would be very difficult to tell where, say, bit 787 ended and bit 788 began. Imagine driving down a highway with no odometer or highway markings and being asked to stop exactly at mile #787 on the highway. It would be pretty hard to do, even if you knew where you started from and your exact speed.
- **Field Separation:** Although we can conceptually think of putting 1000 tiny N-S pole magnets in a row one after the other, in reality magnetic fields don't work this way. They are additive. Aligning 1000 small magnetic fields near each other would create one *large* magnetic field, 1000 times the size and strength of the individual components. Without getting too far into the details, let's just say that this would, in layman's terms, create a mess. :^)

Therefore, in order to encode data on the hard disk so that we'll be able to read it back reliably, we need to take the issues above into account. We must encode using flux reversals, not absolute fields. We must keep the number of consecutive fields of same polarity to a minimum. And to keep track of which bit is where, some sort of clock synchronization must be added to the encoding sequence. Considering the highway example above, this is somewhat analogous to adding markers or milestones along the road.



Idealized depiction of the way hard disk data is written and then read. The top waveform shows how patterns are written to the disk. In the middle, a representation is shown of the way the media on the disk is magnetized into domains of opposite direction based on the polarity of the write current. The waveform on the bottom shows how the flux transitions on the disk translate into positive and negative voltage pulses when the disk is read. Note that the pattern above is made up and doesn't follow any particular pattern or encoding method.

In addition to the requirements we just examined, there's another design limit that must be taken into account: the magnetization limits of the media itself. Each linear inch of space on a track can only store so many flux reversals. This is one of the limitations in recording density, the number of bits that can be stored on the platter surface. Since we need to use some flux reversals to provide clock synchronization, these are not available for data. A prime goal of data encoding methods is therefore to decrease the number of flux reversals used for clocking relative to the number used for real data.

The earliest encoding methods were relatively primitive and wasted a lot of flux reversals on clock information. Over time, storage engineers discovered progressively better methods that used fewer flux reversals to encode the same amount of information. This allowed the data to effectively be packed tighter into the same amount of space. It's important to understand the distinction of what density means in this context. Hardware technology strives to allow more bits to be stored in the same area by allowing more flux reversals per linear inch of track. Encoding methods strive to allow more bits to be stored by allowing more bits to be encoded (on average) per flux reversal.

Note: There are in fact many data encoding and decoding methods. I only examine here the most common ones used for PC-related technologies.

Next: Frequency Modulation (FM)

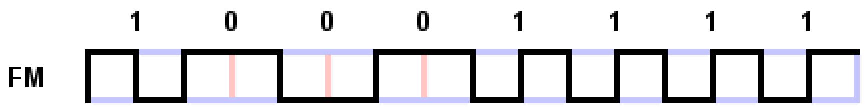
Frequency Modulation (FM)

The first common encoding system for recording digital data on magnetic media was *frequency modulation*, of course abbreviated *FM*. This is a simple scheme, where a one is recorded as two consecutive flux reversals, and a zero is recorded as a flux reversal followed by no flux reversal. This can also be thought of as follows: a flux reversal is made at the start of each bit to represent the clock, and then an additional reversal is added in the middle of each bit for a one, while the additional reversal is omitted for a zero.

This table shows the encoding pattern for FM (where I have designated "R" to represent a flux reversal and "N" to represent no flux reversal). The average number of flux reversals per bit on a random bit stream pattern is 1.5. The best case (all zeroes) would be 1, the worst case (all ones) would be 2:

Bit Pattern	Encoding Pattern	Flux Reversals Per Bit	Bit Pattern Commonality In Random Bit Stream
0	RN	1	50%
1	RR	2	50%
Weighted Average		1.5	100%

The name "frequency modulation" comes from the fact that the number of reversals is doubled for ones compared to that for zeros. This can be seen in the patterns that are created if you look at the encoding pattern of a stream of ones or zeros. A byte of zeroes would be encoded as "RNRNRNRNRNRNRNRN", while a byte of all ones would be "RRRRRRRRRRRRRRRR". As you can see, the ones have double the frequency of reversals compared to the zeros; hence frequency modulation (meaning, changing frequency based on data value).



FM encoding write waveform for the byte "10001111". Each bit cell is depicted as a blue rectangle with a pink line representing the position where a reversal is placed, if necessary, in the middle of the cell.

The problem with FM is that it is very wasteful: each bit requires two flux reversal positions, with a flux reversal being added for clocking every bit. Compared to more advanced encoding methods that try to reduce the number of clocking reversals, FM requires double (or more) the number of reversals for the same amount of data. This method was used on the earliest floppy disk drives, the immediate ancestors of those used in PCs. If you remember using "single density" floppy disks in the late 1970s or early 1980s, that designation commonly refers to magnetic storage using FM encoding. FM was actually made obsolete by MFM before the IBM PC was introduced, but it provides the basis for understanding MFM.

Note: This has nothing whatever to do with FM radio, of course, except for a similarity in the concept of how the data is encoded.

Next: Modified Frequency Modulation (MFM)

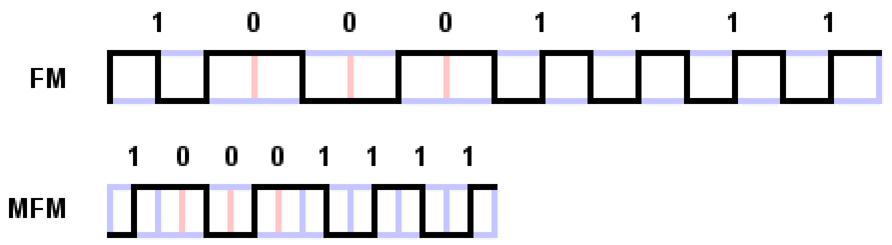
Modified Frequency Modulation (MFM)

A refinement of the FM encoding method is *modified frequency modulation*, or *MFM*. MFM improves on FM by reducing the number of flux reversals inserted just for the clock. Instead of inserting a clock reversal at the start of every bit, one is inserted only between consecutive zeros. When a 1 is involved there is already a reversal (in the middle of the bit) so additional clocking reversals are not needed. When a zero is preceded by a 1, we similarly know there was recently a reversal and another is not needed. Only long strings of zeros have to be "broken up" by adding clocking reversals.

This table shows the encoding pattern for MFM (where I have designated "R" to represent a flux reversal and "N" to represent no flux reversal). The average number of flux reversals per bit on a random bit stream pattern is 0.75. The best case (a repeating pattern of ones and zeros, "101010...") would be 0.25, the worst case (all ones or all zeros) would be 1:

Bit Pattern	**Encoding Pattern**	**Flux Reversals Per Bit**	**Bit Pattern Commonality In Random Bit Stream**
0 (preceded by 0)	RN	1	25%
0 (preceded by 1)	NN	0	25%
1	NR	1	50%
Weighted Average		0.75	100%

Since the average number of reversals per bit is half that of FM, the clock frequency of the encoding pattern can be doubled, allowing for approximately double the storage capacity of FM for the same areal density. The only cost is somewhat increased complexity in the encoding and decoding circuits, since the algorithm is a bit more complicated. However, this isn't a big deal for controller designers, and is a small price to pay for doubling capacity.



FM and MFM encoding write waveform for the byte "10001111". As you can see, MFM encodes the same data in half as much space, by using half as many flux reversals per bit of data.

MFM encoding was used on the earliest hard disks, and also on floppy disks. Since the MFM method about doubles the capacity of floppy disks compared to earlier FM ones, these disks were called "double density". In fact, MFM is still the standard that is used for floppy disks today. For hard disks it was replaced by the more efficient RLL methods. This did not happen for floppy disks, presumably because the need for more efficiency was not nearly so great, compared to the need for backward compatibility with existing media.

Next: Run Length Limited (RLL)

Run Length Limited (RLL)

An improvement on the MFM encoding technique used in earlier hard disks and used on all floppies is *run length limited* or *RLL*. This is a more sophisticated coding technique, or more correctly stated, "family" of techniques. I say that RLL is a family of techniques because there are two primary parameters that define how RLL works, and therefore, there are several different variations. (Of course, you don't really need to know which one your disk is using, since this is all internal to the drive anyway).

FM encoding has a simple one-to-one correspondence between the bit to be encoded and the flux reversal pattern. You only need to know the value of the current bit. MFM improves encoding efficiency over FM by more intelligently controlling where clock transitions are added into the data stream; this is enabled by considering not just the current bit but also the one before it. That's why there are is a different flux reversal pattern for a 0 preceded by another 0, and for a 0 preceded by a 1. This "looking backwards" allows improved efficiency by letting the controller consider more data in deciding when to add clock reversals.

RLL takes this technique one step further. It considers groups of several bits instead of encoding one bit at a time. The idea is to mix clock and data flux reversals to allow for even denser packing of encoded data, to improve efficiency. The two parameters that define RLL are the *run length* and the *run limit* (and hence the name). The word "run" here refers to a sequence of spaces in the output data stream without flux reversals. The run length is the *minimum* spacing between flux reversals, and the run limit is the *maximum* spacing between them. As mentioned before, the amount of time between reversals cannot be too large or the read head can get out of sync and lose track of which bit is where.

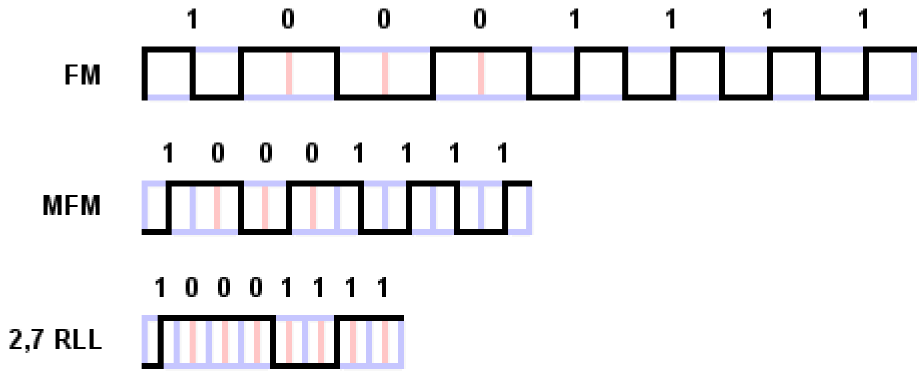
The particular variety of RLL used on a drive is expressed as "RLL (X,Y)" or "X,Y RLL" where X is the run length and Y is the run limit. The most commonly used types of RLL in hard drives are "RLL (1,7)", also seen as "1,7 RLL"; and "RLL (2,7)" ("2,7 RLL"). Alright, now consider the spacing of potential flux reversals in the encoded magnetic stream. In the case of "2,7", this means that the the smallest number of "spaces" between flux reversals is 2, and the largest number is 7. To create this encoding, a set of patterns is used to represent various bit sequences, as shown in the table below ("R" is a reversal, "N" no reversal, just as with the other data encoding examples):

Bit Pattern	**Encoding Pattern**	**Flux Reversals Per Bit**	**Bit Pattern Commonality In Random Bit Stream**
11	RNNN	1/2	25%
10	NRNN	1/2	25%
011	NNRNNN	1/3	12.5%
010	RNNRNN	2/3	12.5%

000	NNNRNN	1/3	12.5%
0010	NNRNNNRNN	2/4	6.25%
0011	NNNNRNNN	1/4	6.25%
Weighted Average		0.4635	100%

The controller these patterns by parsing the bit stream to be encoded, and matching the stream based on the bit patterns it encounters. If we were writing the byte "10001111" (8Fh), this would be matched as "10-0011-11" and encoded as "NRNN-NNNNRNNN-RNNN". Note that the since every pattern above ends in "NN", the minimum distance between reversals is indeed two. The maximum distance would be achieved with consecutive "0011" patterns, resulting in "NNNNRNNN-NNNNRNNN" or seven non-reversals between reversals. Thus, RLL (2,7).

Comparing the table above to the ones for FM and MFM, a few things become apparent. The most obvious is the increased complexity: seven different patterns are used, and up to four bits are considered a time for encoding. The average number of flux reversals per bit on a random bit stream pattern is 0.4635, or about 0.50. This is about a third of the requirement for FM (and about two thirds that of MFM). So relative to FM, data can be packed into one third the space. (For the example byte "10001111" we have been using, RLL requires 3 "R"s; MFM would require 7, and FM would need 13.)



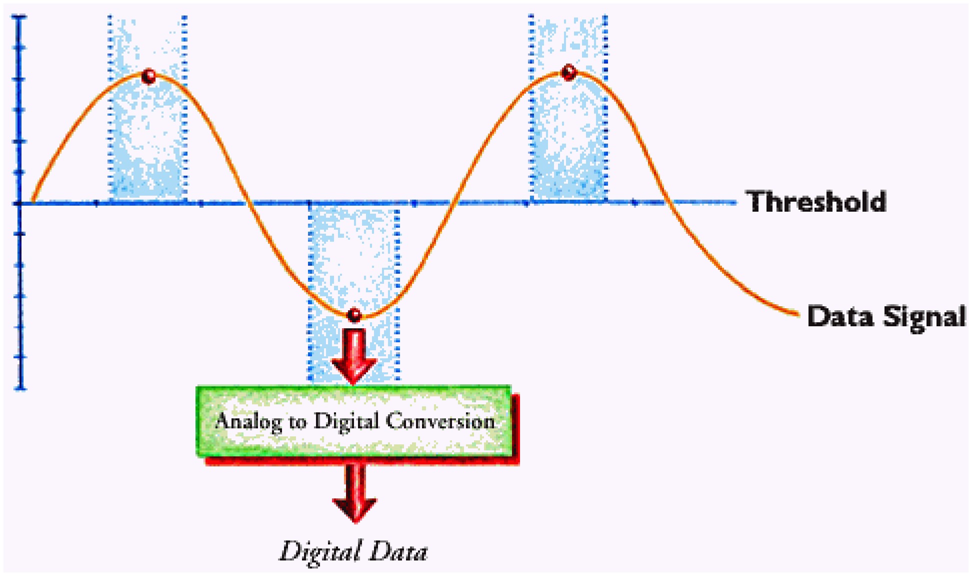
2,7 RLL, FM and MFM encoding write waveform for the byte "10001111". RLL improves further on MFM by reducing the amount of space required for the same data bits to one third that required for regular FM encoding.

Due to its greater efficiency, RLL encoding has replaced MFM everywhere but on floppy disks, where MFM continues to be used for historical compatibility reasons.

Next: Partial Response, Maximum Likelihood (PRML)

Partial Response, Maximum Likelihood (PRML)

Standard read circuits work by detecting flux reversals and interpreting them based on the encoding method that the controller knows has been used on the platters to record bits. The data signal is read from the disk using the head, amplified, and delivered to the controller. The controller converts the signal to digital information by analyzing it continuously, synchronized to its internal clock, and looking for small voltage spikes in the signal that represent flux reversals. This traditional method of reading and interpreting hard disk data is called *peak detection*.

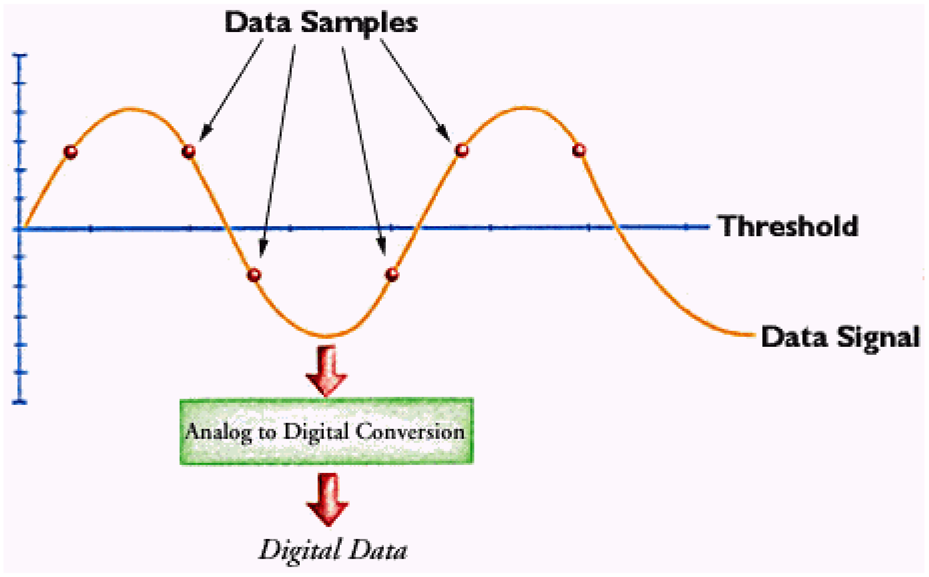


Conceptual drawing demonstrating the principles behind analog peak detection.
The circuitry scans the data read from the disk looking for positive or negative "spikes" that represent flux reversals on the surface of the hard disk platters.

Image © Quantum Corporation
Image used with permission.

This method works fine as long as the peaks are large enough to be picked out from the background noise of the signal. As data density increases, the flux reversals are packed more tightly and the signal becomes much more difficult to analyze, because the peaks get very close together and start to interfere with each other. This can potentially cause bits to be misread from the disk. Since this is something that must be avoided, in practical terms what happens instead is that the maximum areal density on the disk is limited to ensure that interference does not occur. To take the next step up in density, the magnetic fields must be made weaker. This reduces interference, but causes peak detection to be much more difficult. At some point it becomes very hard for the circuitry to actually tell where the flux reversals are.

To combat this problem a new method was developed that takes a different approach to solving the data interpretation problem. This technology, called *partial response, maximum likelihood* or *PRML*, changes entirely the way that the signal is read and decoded from the surface of the disk. Instead of trying to distinguish individual peaks to find flux reversals, a controller using PRML employs sophisticated digital signal sampling, processing and detection algorithms to manipulate the analog data stream coming from the disk (the "partial response" component) and then determine the most likely sequence of bits this represents ("maximum likelihood").



Conceptual drawing demonstrating the principles behind PRML. The data stream is sampled and analyzed using digital signal processing techniques.

Image © Quantum Corporation
Image used with permission.

While this may seem like an odd (and unreliable) way to read data from a hard disk, it is in fact reliable enough that PRML, and its successor, EPRML, have become the standard for data decoding on modern hard disks. PRML allows areal densities to be increased by a full 30-40% compared to standard peak detection, resulting in much greater capacities in the same number of platters.

Next: Extended PRML (EPRML)

Extended PRML (EPRML)

An evolutionary improvement on the PRML design has been developed over the last few years. Called *extended partial response, maximum likelihood*, *extended PRML* or just *EPRML*, this advance was the result of engineers tweaking the basic PRML design to improve its performance. EPRML devices work in a similar way to PRML ones: they are still based on analyzing the analog data stream coming form the read/write head to determine the correct data sequence. They just use better algorithms and signal-processing circuits to enable them to more effectively and accurately interpret the information coming from the disk.

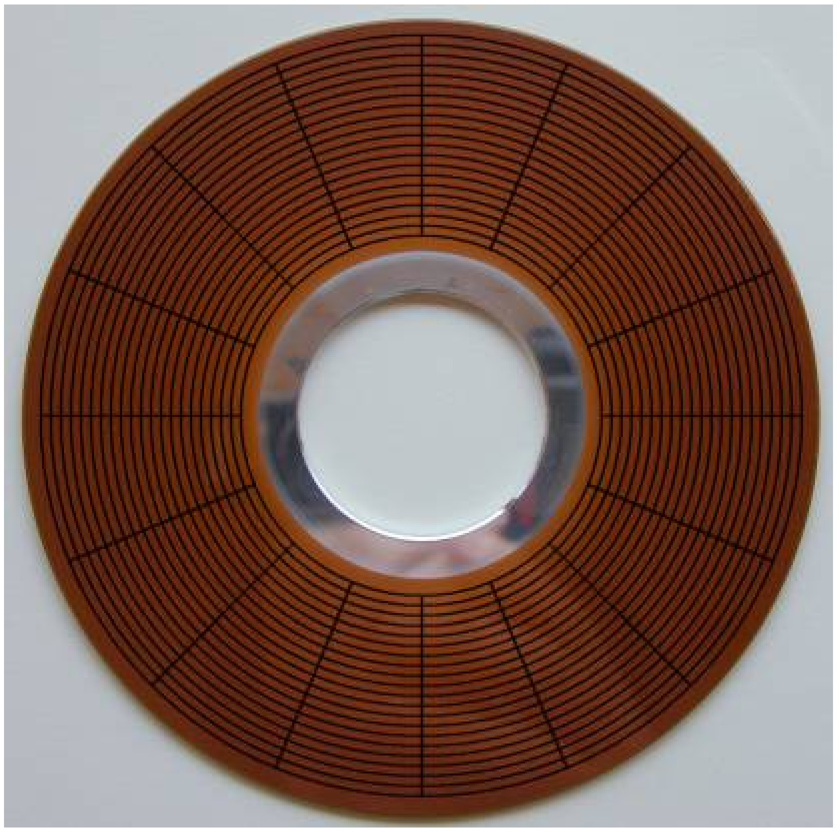
The chief benefit of using EPRML is that due to its higher performance, areal density (or more correctly, the linear component of areal density) can be increased without increasing the error rate. Claims regarding this increase range from around 20% to as much as 70%, compared to "regular" PRML. Those numbers represent a fairly significant improvement.

EPRML has now been widely adopted in the hard disk industry and is replacing PRML on new drives.

Next: Tracks, Cylinders and Sectors

Hard Disk Tracks, Cylinders and Sectors

All information stored on a hard disk is recorded in tracks, which are concentric circles placed on the surface of each platter, much like the annual rings of a tree. The tracks are numbered, starting from zero, starting at the outside of the platter and increasing as you go in. A modern hard disk has tens of thousands of tracks on each platter.



A platter from a 5.25" hard disk, with 20 concentric tracks drawn over the surface. Each track is divided into 16 imaginary sectors.

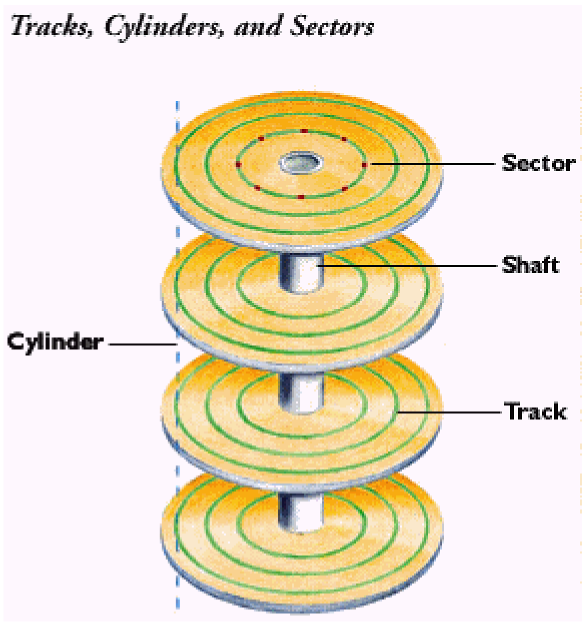
Data is accessed by moving the heads from the inner to the outer part of the disk, driven by the head actuator. This organization of data allows for easy access to any part of the disk, which is why disks are called *random access* storage devices. Each track can hold many thousands of bytes of data. It would be wasteful to make a track the smallest unit of storage on the disk, since this would mean small files wasted a large amount of space. Therefore, each track is broken into smaller units called *sectors*. Each sector holds 512 bytes of user data, plus as many as a few dozen additional bytes used for internal drive control and for error detection and correction.

Next: The Difference Between Tracks and Cylinders

The Difference Between Tracks and Cylinders

A hard disk is usually made up of multiple platters, each of which use two heads to record and read data, one for the top of the platter and one for the bottom (this isn't always the case, but usually is; see here for more details). The heads that access the platters are locked together on an assembly of head arms. This means that all the heads move in and out together, so each head is always physically located at the same track number. It is not possible to have one head at track 0 and another at track 1,000.

Because of this arrangement, often the track location of the heads is not referred to as a track number but rather as a *cylinder* number. A cylinder is basically the set of all tracks that all the heads are currently located at. So if a disk had four platters, it would (normally) have eight heads, and cylinder number 720 (for example) would be made up of the set of eight tracks, one per platter surface, at track number 720. The name comes from the fact that if you mentally visualize these tracks, they form a skeletal cylinder because they are equal-sized circles stacked one on top of the other in space.



This diagram illustrates what "cylinder" means on on a hard disk. This conceptual hard disk spindle has four platters, and each platter has three tracks shown on it. The cylinder indicated would be made up of the 8 tracks (2 per surface) intersected by the dotted vertical line shown.

Image © Quantum Corporation
Image used with permission.

For most practical purposes, there really isn't much difference between tracks and cylinders--its basically a different way of thinking about the same thing. The addressing of individual sectors of the disk is traditionally done by referring to cylinders, heads and sectors (CHS). Since a cylinder is the collection of track numbers located at all of the heads of the disk, the specification "track number plus head number" is equal to "(cylinder number plus head number) plus head number", which is thus the same as "track number plus head number".

Next: Track Density and Areal Density

Track Density and Areal Density

The *track density* of a hard disk refers, unsurprisingly, to how tightly packed the tracks are on the surface of each platter. Every platter has the same track density. The greater the track density of a disk, the more information that can be placed on the hard disk. Track density is one component of areal density, which refers to the number of bits that can be packed into each unit of area on the surface of the disk. More is better--both in terms of capacity and performance. See this page for a full discussion of density issues.

The earliest PC hard disks had only a few hundred tracks on them, and used larger 5.25" form factor platters, resulting in a track density of only a few hundred tracks per inch. Modern hard disks have *tens of thousands* of tracks and can have a density of 30,000 tracks per inch or more.

The chief obstacle to increasing track density is making sure that the tracks don't get close enough together that reading one track causes the heads to pick up data from adjacent tracks. To avoid this problem, magnetic fields are made weaker to prevent interference, which leads to other design impacts, such as the requirement for better read/write head technologies and/or the use of PRML methods to improve signal detection and processing.

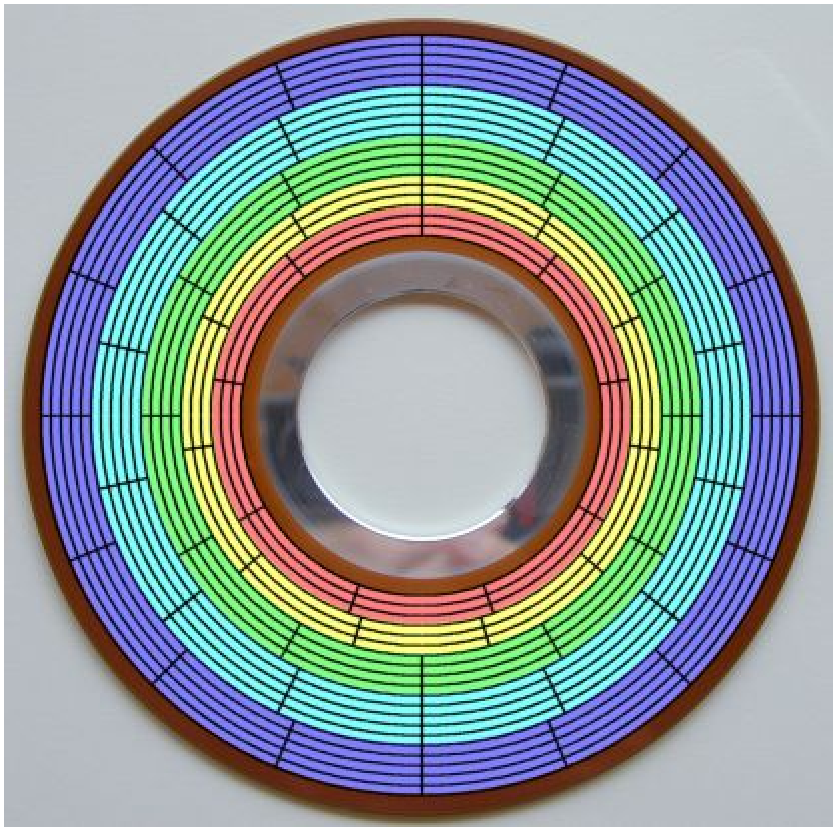
Next: Zoned Bit Recording

Zoned Bit Recording

One way that capacity and speed have been improved on hard disks over time is by improving the utilization of the larger, outer tracks of the disk. The first hard disks were rather primitive affairs and their controllers couldn't handle complicated arrangements that changed between tracks. As a result, every track had the same number of sectors. The standard for the first hard disks was 17 sectors per track.

Of course, the tracks are concentric circles, and the ones on the outside of the platter are much larger than the ones on the inside--typically double the circumference or more. Since there is a constraint on how tight the inner circles can be packed with bits, they were packed as tight as was practically possible given the state of technology, and then the outer circles were set to use the same number of sectors by reducing their bit density. This means that the outer tracks were greatly underutilized, because in theory they could hold many more sectors given the same linear bit density limitations.

To eliminate this wasted space, modern hard disks employ a technique called *zoned bit recording* (*ZBR*), also sometimes called *multiple zone recording* or even just *zone recording*. With this technique, tracks are grouped into zones based on their distance from the center of the disk, and each zone is assigned a number of sectors per track. As you move from the innermost part of the disk to the outer edge, you move through different zones, each containing more sectors per track than the one before. This allows for more efficient use of the larger tracks on the outside of the disk.



A graphical illustration of zoned bit recording. This model hard disk has 20 tracks. They have been divided into five zones, each of which is shown as a different color. The blue zone has 5 tracks, each with 16 sectors; the cyan zone 5 tracks of 14 sectors each; the green zone 4 tracks of 12 sectors; the yellow 3 tracks of 11 sectors, and the red 3 tracks of 9 sectors. You can see that the size (length) of a sector remains fairly constant over the entire surface of the disk (contrast to the non-ZBR diagram on this page.) If not for ZBR, if the inner-most zone had its data packed as densely as possible, every track on this hard disk would be limited to only 9 sectors, greatly reducing capacity.

One interesting side effect of this design is that the raw data transfer rate (sometimes called the *media transfer rate*) of the disk when reading the outside cylinders is much higher than when reading the inside ones. This is because the outer cylinders contain more data, but the angular velocity of the platters is constant regardless of which track is being read (note that this constant angular velocity is *not* the case for some technologies, like older CD-ROM drives!) Since hard disks are filled from the outside in, the fastest data transfer occurs when the drive is first used. Sometimes, people benchmark their disks when new, and then many months later, and are surprised to find that the disk is getting slower! In fact, the disk most likely has not changed at all, but the second benchmark may have been run on tracks closer to the middle of the disk. (Fragmentation of the file system can have an impact as well in some cases.)

As an example, the table below shows the zones used by a 3.8 GB Quantum Fireball TM hard disk, which has a total of 6,810 user data tracks on each platter surface. Also included is the raw data transfer rate for each zone; notice how it decreases as you move from the outer edge of the disk (zone 0) to the hub of the disk (zone 14)--the data transfer rate at the edge is almost double what it is in the middle:

Zone	**Tracks in Zone**	**Sectors Per Track**	**Data Transfer Rate (Mbits/s)**
0	454	232	92.9
1	454	229	91.7
2	454	225	90.4
3	454	225	89.2
4	454	214	85.8
5	454	205	82.1
6	454	195	77.9
7	454	185	74.4
8	454	180	71.4
9	454	170	68.2
10	454	162	65.2
11	454	153	61.7
12	454	142	57.4
13	454	135	53.7
14	454	122	49.5

(From Quantum Fireball TM Product Manual, © 1996 Quantum Corporation.)

A couple of additional thoughts on this data. First, having the same number of tracks per zone is not a requirement; that is just how Quantum set up this disk family. (Compare to the newer IBM drive below.) Second, notice how much larger the sector per track numbers are, compared to the 17 of the earliest disks! Modern drives can pack a lot of storage into a track. Also, this is a 1996-era drive; modern units have even higher numbers of sectors per track in all zones, and much higher data transfer rates. Here's the same chart for the 20 GB/platter, 5400 RPM IBM 40GV drive:

Zone	**Tracks in Zone**	**Sectors Per Track**	**Data Transfer Rate (Mbits/s)**

0	624	792	372.0
1	1,424	780	366.4
2	1,680	760	357.0
3	1,616	740	347.6
4	2,752	720	338.2
5	2,880	680	319.4
6	1,904	660	310.0
7	2,384	630	295.9
8	3,328	600	281.8
9	4,432	540	253.6
10	4,528	480	225.5
11	2,192	440	206.7
12	1,600	420	197.3
13	1,168	400	187.9
14	18,15	370	173.8

(From Deskstar 40GV and 75GXP Product Manual, © 2000 International Business Machines Corporation.)

As you can see, the number of tracks per zone, sectors per track and data transfer rates are all several times higher than the numbers for the older drive, showing how dramatically capacity and performance have increased in four years. The number of tracks per zone is different for each zone here, unlike the Quantum drive. However, you will notice that the number of zones is the same. The fact that both drives have exactly 15 zones is a coincidence--some drives have a few more, and some a few less--but new drives do tend to have roughly the same number of zones as older ones. Increasing the number of zones makes the controller more complicated, and there usually isn't any great benefit to doing this. (Ideally you would maximize the storage potential if every track had its own "zone" with *just* the right number of sectors, but the added storage would be relatively small compared to just using larger zones as above, and the engineering cost very high.)

The standard BIOS settings for IDE/ATA hard disks only allow the specification of a single number for "sectors per track". Since all modern hard disks use ZBR and don't *have* a single number of sectors per track across the disk, they use logical geometry for the BIOS setup. IDE hard disks up to 8.4 GB usually tell the BIOS 63 sectors per track and then translate to the real geometry internally; no modern drive uses 63 sectors on *any* track, much less all of them. Hard drives over 8.4 GB can't have their parameters expressed using the IDE BIOS geometry parameters anyway (because the regular BIOS limit is 8.4 GB) so these drives always have 63 sectors per track as "dummy" geometry parameters, and are accessed using logical block addressing. (See here for further discussion.)

All of the above is one reason why modern drives are low-level formatted at the factory. The hard disk controller has to know the intricate details of the various recording zones, how many sectors are in each track for each zone, and how everything is organized.

Next: Write Precompensation

Write Precompensation

As discussed in the section on zoned bit recording, older hard disks used the same number of sectors per track. This meant that older disks had a varying bit density as you moved from the outside edge to the inner part of the platter. Many of these older disks required that an adjustment be made when writing the inside tracks, and a setting was placed in the BIOS to allow the user to specify at what track number this compensation was to begin.

This entire matter is no longer relevant to modern hard disks, but the BIOS setting remains for compatibility reasons. Write precompensation is not done with today's drives; even if it were, the function would be implemented within the integrated controller and would be transparent to the user.

Next: Interleaving

Interleaving

A common operation when working with a hard disk is reading or writing a number of sectors of information in sequence. After all, a sector only contains 512 bytes of user data, and most files are much larger than that. Let's assume that the sectors on each track are numbered consecutively, and say that we want to read the first 10 sectors of a given track on the hard disk. Under ideal conditions, the controller would read the first sector, then immediately read the second, and so on, until all 10 sectors had been read. Just like reading 10 words in a row in this sentence.

However, the physical sectors on a track are adjacent to each other and not separated by very much space. Reading sectors consecutively requires a certain amount of speed from the hard disk controller. The platters never stop spinning, and as soon as the controller is done reading all of sector #1, it has little time before the start of sector #2 is under the head. Many older controllers used with early hard disks did not have sufficient processing capacity to be able to do this. They would not be ready to read the second sector of the track until after the start of the second physical sector had already spun past the head, at which point it would be too late.

If the controller *is* slow in this manner, and no compensation is made in the controller, the controller must wait for almost an entire revolution of the platters before the start of sector #2 comes around and it can read it. Then, of course, when it tried to read sector #3, the same thing would happen, and another complete rotation would be required. All this waiting around would kill performance: if a disk had 17 sectors per track, it would take 17 times as long to read those 10 sectors as it should have in the ideal case!

To address this problem, older controllers employed a function called *interleaving*, allowing the setting of a disk parameter called the *interleave factor*. When interleaving is used, the sectors on a track are logically re-numbered so that they do not correspond to the physical sequence on the disk. The goal of this technique is to arrange the sectors so that their position on the track matches the speed of the controller, to avoid the need for extra "rotations". Interleave is expressed as a ratio, "N:1", where "N" represents how far away the second logical sector is from the first, how far the third is from the second, and so on.

An example is the easiest way to demonstrate this method. The standard for older hard disks was 17 sectors per track. Using an interleave factor of 1:1, the sectors would be numbered 1, 2, 3, .. , 17, and the problem described above with the controller not being ready in time to read sector #2 would often occur for sequential reads. Instead, an interleave factor of 2:1 could be used. With this arrangement, the sectors on a 17-sector track would be numbered as follows: 1, 10, 2, 11, 3, 12, 4, 13, 5, 14, 6, 15, 7, 16, 8, 17, 9. Using this interleave factor means that while sector 1 is being processed, sector *10* is passing under the read head, and so when the controller is ready, sector 2 is just arriving at the head. To read the entire track, two revolutions of the platters are required. This is twice as long as the ideal case (1:1 interleaving with a controller fast enough to handle it) but it is almost 90%

better than what would result from using 1:1 interleaving with a controller that is too slow (which would mean 17 rotations were required).

What if the controller was too slow for a 2:1 interleave? It might only be fast enough to read every third physical sector in sequence. If so, an interleave of 3:1 could be used, with the sectors numbered as follows: 1, 7, 13, 2, 8, 14, 3, 9, 15, 4, 10, 16, 5, 11, 17, 6, 12. Again here, this would reduce performance compared to 2:1, if the controller was fast enough for 2:1, but it would greatly *improve* performance if the controller couldn't handle 2:1.

So this begs the question then: how do you know what interleave factor to use? Well, on older hard disks, the interleave factor was one parameter that had to be tinkered with to maximize performance. Setting it too conservatively caused the drive to not live up to its maximum potential, but setting it too aggressively could result in severe performance hits due to extra revolutions being needed. The perfect interleave setting depended on the speeds of the hard disk, the controller, and the system. Special utilities were written to allow the analysis of the hard disk and controller, and would help determine the optimal interleave setting. The interleave setting would be used when the drive was low-level formatted, to set up the sector locations for each track.

On modern disk drives, the interleave setting is always 1:1. Controller too slow? Ha! Today's controllers are so fast, much of the time they sit around waiting for the platters, tapping their virtual fingers. How did this situation come to change so drastically in 15 years? Well, it's pretty simple. The spindle speed of a hard disk has increased from 3,600 RPM on the first hard disks, to today's standards of 5,400 to 10,000 RPM. An increase in speed of 50% to 177%. The faster spindle speed means that much less time for the controller to be ready before the next physical sector comes under the head. However, look at what processing power has done in the same time frame: CPUs have gone from 4.77 MHz speeds to the environs of 1 GHz; an increase of over 20,000%! The speed of other chips in the PC and its peripherals have similarly gotten faster by many multiples.

As a result of this increase in speed in modern circuits, controller speed is no longer an issue for current drives. There is in fact no way to set the interleave for a modern drive; it is fixed at 1:1 and no other setting would be necessary. Understanding interleaving is still important because the concept forms the basis for more advanced techniques such as head and cylinder skew, which *are* used on modern drives. And, well, you never know when you might have to deal with some old 20 MB clunker. ;^)

Warning: Some older systems still have BIOS functions for doing "media analysis" or setting interleave factors for hard disks. These are the interleave analysis utilities I mentioned above. They are intended for older style drives that do not have an integrated controller. They should not be used with modern IDE or SCSI disks.

Note: The hard disk interleave setting described here is a totally different concept than memory interleaving.

Next: Cylinder and Head Skew

Cylinder and Head Skew

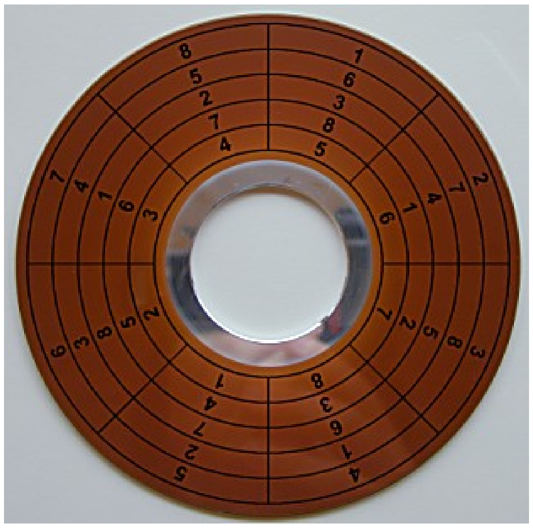
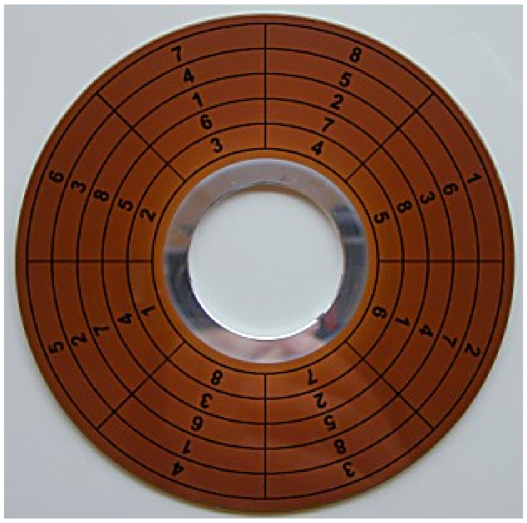
Sector interleaving was once used on older hard disks to ensure that the sectors were efficiently spaced on the track. This was needed to ensure that sector #2 didn't rotate past the head while sector #1 was being processed. The high-speed disk controllers on modern drives are now fast enough that they no longer are a performance-limiting factor in how the sectors on the disk are arranged. However, there are other delay issues within the drive that require spacing to be optimized in even the fastest drives, to maximize performance. And unlike the interleaving situation, these delays are caused by electromechanical concerns and are therefore likely to be with us for as long as hard drives use their current general design.

The first issue is the delay in time incurred when switching between cylinders on the hard disk, called appropriately enough, cylinder switch time. Let's imagine that we "lined up" all of the tracks on a platter so that the first sector on each track started at the same position on the disk. Now let's say that we want to read the entire contents of two consecutive tracks, a fairly common thing to need to do. We read all the sectors of track #1 (in sequence, since we can use a 1:1 interleave) and then switch to track #2 to start reading it at its first sector.

The problem here is that it takes time to physically move the heads (or more actually, the actuator assembly) to track #2. In fact, it often takes a millisecond or more. Let's consider a modern 10,000 RPM drive. The IBM Ultrastar 72ZX has a specification of only 0.6 milliseconds for seeking from one track to an adjacent one. That's actually quite fast by today's standards. But consider that in that amount of time, a 10,000 RPM drive will perform approximately 10% of a complete revolution of the platters! If sector #1 on track #2 is lined up with sector #1 on track #1, it will be long gone by the time we switch from track #1 to track #2. We'd have to wait for the remaining 90% of a revolution of the platters to do the next read, a big performance penalty. This problem isn't as bad as the interleave one was, because it occurs only when changing tracks, and not every sector. But it's still bad, and it's avoidable.

The issue is avoided by offsetting the start sector of adjacent tracks to minimize the likely wait time (rotational latency) when switching tracks. This is called *cylinder skew*. Let's say that in the particular zone where tracks #1 and #2 are, there are 450 sectors per track. If 10% of the disk spins by on a track-to-track seek, 45 sectors go past. Allowing some room for error and controller overhead, perhaps the design engineers would shift each track so that sector #1 of track #2 was adjacent to sector #51 of track #1. Similarly, sector #1 of track #3 would be adjacent to sector #51 of track #2 (and hence, adjacent to sector #101 of track #1). And so on. By doing this, we can read multiple adjacent tracks virtually seamlessly, and with no performance hit due to unnecessary platter rotations.

The same problem, only to a lesser degree, occurs when we change heads within a cylinder. Here there is no physical movement, but it still takes time for the switch to be made from reading one head to reading another, so it makes sense to offset the start sector of tracks within the same cylinder so that after reading from the first head/track in the cylinder, we can switch to the next one without losing our "pace". This is called *head skew*. Since switching heads takes much less time than switching cylinders, head skew usually means a smaller number of sectors being offset than cylinder skew does.

These two diagrams illustrate the concept of cylinder and head skew. Assume that these platters spin counter-clockwise (as seen from your vantage point) and that they are adjacent to each other (they might be the two surfaces of the same platter.) They each have a cylinder skew of three, meaning that adjacent tracks are offset by three sectors. In addition, the platter on the right has a head skew of one relative to the one on the left. (Of course, real drives have thousands of tracks with hundreds of sectors each.)

Both cylinder and head skew must be simultaneously "overlaid" onto all the tracks of the hard disk, resulting in a "two-dimensional pattern" of sorts, with different offsets being applied depending on the specific timing characteristics of the disk. The layout of the tracks is adjusted to account for cylinder skew and head skew, based on the way the designers intend the hard disk to store sequential data. All of the details are taken care of by the controller. This is one reason why having integrated, dedicated drive electronics on the disk itself, is such a good idea. No universal, external controller could possibly know how to take all these hard disk characteristics and performance requirements into account.

Next: Sector Format and Structure

Sector Format and Structure

The basic unit of data storage on a hard disk is the *sector*. The name "sector" comes from the mathematical term, which refers to a "pie-shaped" angular section of a circle, bounded on two sides by radii and the third by the perimeter of the circle. On a hard disk containing concentric circular tracks, that shape would define a sector of each track of the platter surface that it intercepted. This is what is called a *sector* in the hard disk world: a small segment along the length of a track. At one time, all hard disks had the same number of sectors per track, and in fact, the number of sectors in each track was fairly standard between models. Today's advances have allowed the number of sectors per track ("SPT") to vary significantly, as discussed here.

In the PC world, each sector of a hard disk can store 512 bytes of user data. (There are some disks where this number can be modified, but 512 is the standard, and found on virtually all hard drives by default.) Each sector, however, actually holds much more than 512 bytes of information. Additional bytes are needed for control structures and other information necessary to manage the drive, locate data and perform other "support functions". The exact details of how a sector is structured depends on the drive model and manufacturer. However, the contents of a sector usually include the following general elements:

- **ID Information:** Conventionally, space is left in each sector to identify the sector's number and location. This is used for locating the sector on the disk. Also included in this area is status information about the sector. For example, a bit is commonly used to indicate if the sector has been marked defective and remapped.
- **Synchronization Fields:** These are used internally by the drive controller to guide the read process.
- **Data:** The actual data in the sector.
- **ECC:** Error correcting code used to ensure data integrity.
- **Gaps:** One or more "spacers" added as necessary to separate other areas of the sector, or provide time for the controller to process what it has read before reading more bits.

Note: In addition to the sectors, each containing the items above, space on each track is also used for servo information (on embedded servo drives, which is the design used by all modern units).

The amount of space taken up by each sector for overhead items is important, because the more bits used for "management", the fewer overall that can be used for data. Therefore, hard disk manufacturers strive to reduce the amount of non-user-data information that must be stored on the disk. The term *format efficiency* refers to the percentage of bits on each disk that are used for data, as opposed to "other things". The higher the format efficiency of a drive, the better (but don't expect to find statistics on this for your favorite drive easy to find!)

One of the most important improvements in sector format was IBM's creation of the *No-ID Format* in the mid-1990s. The idea behind this innovation is betrayed by the name: the ID fields are removed from the sector format. Instead of labeling each sector within the sector header itself, a format map is stored in memory and referenced when a sector must be located. This map also contains information about what sectors have been marked bad and relocated, where the sectors are relative to the location of servo information, and so on. Not only does this improve format efficiency, allowing up to 10% more data to be stored on the surface of each platter, it also improves performance. Since this critical positioning information is present in high-speed memory, it can be accessed much more quickly. "Detours" in chasing down remapped sectors are also eliminated.

Next: Formatting and Capacity

Hard Disk Formatting and Capacity

Most PC users are familiar with the concept that a hard disk--in fact, all storage media--must be formatted before it can be used. There is usually some confusion, however, regarding exactly what formatting means and what it does. This is exacerbated by the fact that modern hard disks are not formatted in the same way that older ones were, and also the fact that the utilities used for formatting behave differently when acting on hard disks than when used for floppy disks.

This section takes a look at issues surrounding disk formatting and capacity, discusses unformatted and formatted hard disk capacity, and looks briefly at formatting utilities.

Next: Two Formatting Steps

Two Formatting Steps

Many PC users don't realize that formatting a hard disk isn't done in a single step. In fact, three steps are involved:

1. **Low-Level Formatting:** This is the "true" formatting process for the disk. It creates the physical structures (tracks, sectors, control information) on the hard disk. Normally, this step begins with the hard disk platters "clean", containing no information. It is discussed in more detail here.
2. **Partitioning:** This process divides the disk into logical "pieces" that become different hard disk volumes (drive letters). This is an operating system function and is discussed in detail in its own section.
3. **High-Level Formatting:** This final step is also an operating-system-level command. It defines the logical structures on the partition and places at the start of the disk any necessary operating system files. Read more about it here.

As you can see, two of the three steps are "formatting", and this dual use of the word is a big part of what leads to a lot of confusion when the term "formatting" is used. Another strange artifact of history is that the DOS "FORMAT" command behaves differently when it is used on a hard disk than when it is used on a floppy disk. Floppy disks have simple, standard geometry and cannot be partitioned, so the FORMAT command is programmed to automatically both low-level and high-level format a floppy disk, if necessary. For hard disks, however, FORMAT will only do a high-level format. Low-level formatting is performed by the controller for older drives, and at the factory for newer drives.

Next: Low-Level Formatting

Low-Level Formatting

Low-level formatting is the process of outlining the positions of the tracks and sectors on the hard disk, and writing the control structures that define where the tracks and sectors are. This is often called a "true" formatting operation, because it really creates the physical format that defines where the data is stored on the disk. The first time that a low-level format ("LLF") is performed on a hard disk, the disk's platters start out empty. That's the last time the platters will be empty for the life of the drive. If an LLF is done on a disk with data on it already, the data is permanently erased (save heroic data recovery measures which are sometimes possible).

If you've explored other areas of this material describing hard disks, you have learned that modern hard disks are much more precisely designed and built, and much more complicated than older disks. Older disks had the same number of sectors per track, and did not use dedicated controllers. It was necessary for the external controller to do the low-level format, and quite easy to describe the geometry of the drive to the controller so it could do the LLF. Newer disks use many complex internal structures, including zoned bit recording to put more sectors on the outer tracks than the inner ones, and embedded servo data to control the head actuator. They also transparently map out bad sectors. Due to this complexity, all modern hard disks are low-level formatted at the factory for the life of the drive. There's no way for the PC to do an LLF on a modern IDE/ATA or SCSI hard disk, and there's no reason to try to do so.

Older drives needed to be re-low-level-formatted occasionally because of the thermal expansion problems associated with using stepper motor actuators. Over time, the tracks on the platters would move relative to where the heads expected them to be, and errors would result. These could be corrected by doing a low-level format, rewriting the tracks in the new positions that the stepper motor moved the heads to. This is totally unnecessary with modern voice-coil-actuated hard disks.

Warning: You should never attempt to do a low-level format on an IDE/ATA or SCSI hard disk. Do not try to use BIOS-based low-level formatting tools on these newer drives. It's unlikely that you will damage anything if you try to do this (since the drive controller is programmed to ignore any such LLF attempts), but at best you will be wasting your time. A modern disk can usually be restored to "like-new" condition by using a zero-fill utility.

Next: High-Level Formatting

High-Level Formatting

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 be 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.

Next: Defect Mapping and Spare Sectoring

Defect Mapping and Spare Sectoring

Despite the precision manufacturing processes used to create hard disks, it is virtually impossible to create a disk with tens of millions of sectors and not have some errors show up. Imperfections in the media coating on the platter or other problems can make a sector inoperable. A problem with a sector, if uncorrected, would normally manifest as an error when attempting to read or write the sector, but can appear in other ways as well. Most of us have experienced these errors on occasion when using floppy disk drives.

Modern disks use ECC to help identify when errors occur and in some cases correct them, however, there will still be physical flaws that ECC cannot overcome, and that therefore prevent parts of a disk from being used. Usually these are individual sectors that don't work, and they are appropriately enough called *bad sectors*. Tracks where there are bad sectors are sometimes called *bad tracks*.

If you've ever used a disk information utility on a floppy disk (or on a very old hard disk), you've likely at some point seen a report showing a few kilobytes worth of bad sectors. However, if you run such a utility on a modern hard disk, you will normally never see any reports of bad sectors on the disk. Why is this?

Sectors that are bad cannot be used to store data, for obvious reasons: they are bad because they cannot be trusted to reliably write and/or reproduce the data at a later time. It is therefore necessary for *some* part of the system to keep track of where they are, and not use them. The best way for this to be done is for the drive to detect and avoid them. If the drive does not do this, the operating system must do it. If any bad sectors are not detected until after they have been used, data loss will probably result.

To allow for maximum reliability then, each disk drive is thoroughly tested for any areas that might have errors at the time it is manufactured. All the sectors that have problems or are thought to be unreliable, are recorded in a special table. This is called *defect mapping*. Some drives go even further than this, mapping out not only the sectors that are questionable, but the ones surrounding them as well. Some drives will map out entire tracks as a safety precaution.

On older hard disks, these problem areas were actually recorded right on the top cover of the disk, usually in hand-writing by the technician testing the drive! This process was necessary because low-level formatting was done by the company assembling the PC--or even the end-user--and this information was used to tell the controller which areas of the disk to avoid when formatting the disk. Part of the low-level format process was to have the person doing the LLF tell the controller which sectors were bad, so it would avoid them, and also tell any high-level format program not to try to use that part of the disk. These markings are what cause "bad sectors" reports to show up when examining older hard disks: these are the areas the disk has been told not to use. Since floppy disks are low-level formatted and high-level formatted at the same time, the same situation applies, even today. If any sectors cannot be reliably formatted, they are marked as "bad" and the operating system will stay away from them.

While early PC users accepted that a few bad sectors on a drive was normal, there was something distasteful about plopping down $1,000 for a new hard disk and having it report "bad sectors" as soon as you turned it on. There is no way to produce 100% perfect hard disks without them costing a small fortune, so hard disk manufacturers devised an interesting compromise.

On modern hard disks, a small number of sectors are *reserved* as substitutes for any bad sectors discovered in the main data storage area. During testing, any bad sectors that are found on the disk are programmed into the controller. When the controller receives a read or write for one of these sectors, it uses its designated substitute instead, taken from the pool of extra reserves. This is called *spare sectoring*. In fact, some drives have entire spare tracks available, if they are needed. This is all done completely transparently to the user, and the net effect is that all of the drives of a given model have the exact same capacity and there are no visible errors. This means that the operating system never sees the bad areas, and therefore never reports "bad sectors". They are still there though, just cleverly hidden.

Really, when you think about it, the hard disk companies are sacrificing a small amount of storage for "good looks". It would be more efficient to use all of the sectors on the disk and just map out the few bad ones. However, sometimes marketing wins out over engineering, and it seems that more people want the warm feeling of thinking they have a perfect drive, even if it costs them theoretical storage in the process. Today's drives are so enormous that few people would even care much anyway about a few extra megabytes, but that wasn't always the case!

Due to spare sectoring, a brand new disk should not have any bad sectors. It is possible, however, for a modern IDE/ATA or SCSI hard disk to develop new bad sectors over time. These will normally be detected either during a routine scan of the hard disk for errors (the easy way) or when a read error is encountered trying access a program or data file (the hard way). When this happens, it is possible to tell the system to avoid using that bad area of the disk. Again, this can be done two ways. At the high level, the operating system can be told to mark the area as bad and avoid it (creating "bad sector" reports at the operating system level.). Alternately, the disk itself can be told at a low level to *remap* the bad area and use one of its spares instead. This is normally done by using a zero-fill or diagnostic utility, which will scan the entire disk surface for errors and tell the controller to map out any problem areas.

Warning: Bad sectors on a modern hard disk are almost always an indication of a greater problem with the disk. A new hard disk should *never* have bad sectors on it; if you buy one that does have bad sectors, immediately return it to the vendor for exchange (and don't let them tell you "it's normal", because it isn't.) For existing hard disks, the vast majority of time, a single bad sector that appears will soon be accompanied by friends. While you can map out and ignore bad sectors, you should make sure to contact the vendor if you see bad sectors appearing during scans, and make sure the data is backed up as well. Personally, I will not use any hard disk that is developing bad sectors. The risk of data loss is too high, and hard drives today are inexpensive compared to the cost of even an hour or two of recovering lost data (which takes a lot more than an hour or two!) See here for more on troubleshooting hard disk errors.

On some disks, remapped sectors cause a performance penalty. The drive first seeks and reads the sector header of the data, thinking it will be there; then, it sees that the sector has been remapped, and has to do another seek for the new sector. Newer drives using the No-ID sector format eliminate this problem by storing a format map, including sector remaps, in the memory of the drive's controller. See the discussion of sector format for more.

Next: Low-Level Format, Zero-Fill and Diagnostic Utilities

Low-Level Format, Zero-Fill and Diagnostic Utilities

Older hard disks required periodic low-level formatting by the system configurator or end-user. To facilitate this, low-level format utilities were created. These are small programs written to control the low-level formatting process for the hard disk. The hard disk controller would normally include one of these programs in a ROM chip in hardware, enabling access to the software without requiring any drives to be running in the system, and thus avoiding a possible "chicken and egg" quandary. In addition, more sophisticated, third-party utilities were available that would perform an LLF and also do other related features such as scanning for bad sectors or analyzing the drive to determine an optimal interleave setting. These would typically be loaded from a floppy disk.

Low-level formatting an older hard disk could be a rather complicated procedure, particularly for one who was not very familiar with PCs and hard disks. Various factors needed to be taken into account, such as defect mapping and setting the interleave factor. The particular conditions of the drive when formatting were also important: due to the vagaries of stepper-motor actuators, doing an LLF when the drive was very cold or very hot could lead to errors when the drive returned to a more normal temperature. Even the orientation of the drive when it was formatted was an issue.

As I have said (probably too often, sorry) modern drives do not need to be low-level formatted by the end user, and in fact cannot be LLFed outside the factory due to their precision and complexity. However, it seems that the *need* to LLF hard disks on the part of users has never gone away. Like some primordial instinct, many PC users seem to have a fundamental desire to LLF their modern disks. Maybe it is built into the genetic code in some way yet undiscovered. ;^) In fact, even if it were possible, the vast majority of the time that someone "needs" to LLF a hard disk today, it is not really necessary. Many users jump quickly to wanting to try an "LLF" whenever they have a problem with their hard disk, much the way many jump to re-installing their operating system whenever it gives them trouble.

Hard drive manufacturers have created for modern drives replacements for the old LLF utilities. They cause some confusion, because they are often still *called* "low-level format" utilities. The name is incorrect because, again, no utility that a user can run on a PC can LLF a modern drive. A more proper name for this sort of program is a *zero-fill and diagnostic utility*. This software *does* work on the drive at a low level, usually including the following functions (and perhaps others):

- **Drive Recognition Test:** Lets you test to see if the software can "see" the drive. This is the first step in ensuring that the drive is properly installed and connected.
- **Display Drive Details:** Tells you detailed information about the drive, such as its exact model number, firmware revision level, date of manufacture, etc.
- **Test For Errors:** Analyzes the entire surface of the hard disk, looking for problem areas (bad sectors) and instructing the integrated drive controller to remap them.
- **Zero-Fill:** Wipes off all data on the drive by filling every sector with zeroes. Normally a test for errors (as above) is done at the same time.

When most users today talk about "low-level formatting" a drive, what they are really talking about is doing a zero-fill. That procedure will restore a functional drive (that is, one that does not have mechanical problems) to the condition it was in when received from the factory. There are occasions when a modern hard disk can become so badly corrupted that the operating system cannot recover it, and a zero-fill can help in this situation. Stubborn boot sector viruses for example can be hard to eradicate without resorting to low-level intervention. Since the zero-fill cleans all programs and data off the drive it will get rid of almost any data-related problem on the drive, such as viruses, corrupted partitions and the like. Just remember that it's a bit like burning down your house to get rid of termites: you lose *everything* on the drive.

This type of utility can also be used to "hide" bad sectors by telling the drive to remap them to its collection of spares. Just remember that a drive that continues to "grow" bad sectors over time is one whose reliability is highly suspect. I discuss this matter in more detail here.

Warning: Only use a low-level zero-fill or diagnostic utility designed for your particular hard disk. You can download one for free from your drive manufacturer's web site. Even though damage probably won't result from using the wrong program, you may lose data and you may also complicate any warranty service you try to have performed on the drive. (Technical support people at "Company X" generally don't like to hear that you used a utility on their drive written by "Company Y".)

Warning: Always back up your data before you use a low-level utility, and make sure that you carefully follow all of the instructions provided.

Warning: Never run a low-level disk utility from within a multi-tasking operating system such as Windows 9x. Other programs running in the background could interfere with the utility. Restart the computer in MS-DOS mode first, or reboot the computer from a floppy.

Next: Unformatted and Formatted Capacity

Unformatted and Formatted Capacity

Some portion of the space on a hard disk is taken up by the formatting information that marks the start and end of sectors, ECC, and other "overhead". For this reason, a hard disk's storage total depends on if you are looking at the formatted or unformatted capacity. The difference can be quite significant: 20% or even more.

Older drives that were typically low-level formatted by the user, often had their size listed in terms of unformatted capacity. For example, take the Seagate ST-412, the first drive used on the original IBM PC/XT in the early 1980s. The "12" in this model number refers to the drive's *unformatted* capacity of 12.76 MB. Formatted, it is actually a 10.65 MB drive.

Now, let's be honest: stating the capacity of the hard disk in unformatted terms is lame. Since nobody can use a drive that is unformatted, the only thing that matters is the formatted capacity. Stating the drive in terms of unformatted capacity is not quite as bad as how tape drive manufacturers always report the size of their drives assuming 2:1 compression, of course. But it's still *lame*. :^)

Fortunately, this is no longer an issue today. Since modern drives are always low-level formatted at the factory, it would be *extremely* weird to state their sizes in terms of unformatted capacity, and manufacturers have stopped doing this. In fact, there usually isn't any easy way to find *out* the unformatted capacity of new drives! So to take another example from our friends at Seagate, the ST-315330A, the "15330" refers to the drive's approximate *formatted* capacity, 15,364 MB (15.4 GB).

Next: Binary vs. Decimal Capacity Measurements

Binary vs. Decimal Capacity Measurements

Computer measurements are expressed in both binary *and* decimal terms, often using the same notation. Due to a mathematical coincidence, the fact that 2^10 (1024) is almost the same number as 10^3 (1000), there are two similar but different ways to express a megabyte or a gigabyte. This phenomenon, and the general problems it causes, are discussed in detail in this fundamentals section. I also discuss there how and why I have begun using alternative measurement notations for binary numbers.

The problems with binary and decimal are probably more noticed in the area of hard disk capacity than anywhere else. Hard disk manufacturers always use decimal figures for their products' capacity: a 72 GB hard disk has about 72,000,000,000 bytes of storage. However, hard disk makers also use binary numbers where they are normally used--for example, buffer capacities are expressed in binary kilobytes or megabytes--but the same notation ("kB" or "MB") is used as for decimal figures. Hard disks are large, and larger numbers cause the discrepancy between decimal and binary terms to be exaggerated. For example, a 72 GB hard disk, expressed in binary terms, is "only" 67 GB. Since most software uses binary terms, this difference in numbers is the source of frequent confusion regarding "where the rest of the gigabytes went". In fact, they didn't go anywhere. It's just a different way of expressing the same thing.

This is also the source of much confusion surrounding 2.1 GB hard disks (or 2.1 GB hard disk volumes) and the 2 GB DOS limit on partition size. Since DOS uses binary gigabytes, and 2.1 GB hard disks are expressed in decimal terms, a 2.1 GB hard disk can in fact be entirely placed within a single DOS partition. 2.1 decimal gigabytes is actually 1.96 binary gigabytes. Another example is the BIOS limit on regular IDE/ATA hard disks, which is either 504 MB or 528 MB, depending on which "MB" you are talking about.

Next: Geometry Specifications and Translation

Hard Disk Geometry Specifications and Translation

The generic term used to refer to the way the disk structures its data into platters, tracks and sectors, is its *geometry*. In the early days this was a relatively simple concept: the disk had a certain number of heads, tracks per surface, and sectors per track. These were entered into the BIOS set up so the PC knew how to access the drive, and that was basically that.

With newer drives the situation is more complicated. The simplistic limits placed in the older BIOSes have persisted to this day, but the disks themselves have moved on to more complicated ways of storing data, and much larger capacities. The result is that tricks must be employed to ensure compatibility between old BIOS standards and newer hard disks.

Note: These issues relate to IDE/ATA hard disks, not SCSI drives, which use a different addressing methodology.

Next: Physical Geometry

Physical Geometry

Physical Geometry

The physical geometry of a hard disk is the actual physical number of heads, cylinders and sectors used by the disk. On older disks this is the only type of geometry that is ever used--the physical geometry and the geometry used by the PC are one and the same. The original setup parameters in the system BIOS are designed to support the geometries of these older drives. Classically, there are three figures that describe the geometry of a drive: the number of cylinders on the drive ("C"), the number of heads on the drive ("H") and the number of sectors per track ("S"). Together they comprise the "CHS" method of addressing the hard disk. This method of description is described in more detail in this description of CHS mode addressing.

At the time the PC BIOS interfaces to the hard disk were designed, hard disks were simple. They had only a few hundred cylinders, a few heads and all had the same number of sectors in each track. Today's drives do not have simple geometries; they use zoned bit recording and therefore do not have the same number of sectors for each track, and they use defect mapping to remove bad sectors from use. As a result, their geometry can no longer be described using simple "CHS" terms. These drives must be accessed using logical geometry figures, with the physical geometry hidden behind routines inside the drive controller. For a comparison of physical and logical geometry, see this page on logical geometry.

Often, you have to request detailed specifications for a modern drive to find out the true physical geometry. Even then you might have problems--I called one major drive manufacturer when first writing the site, and the technician had no idea what I was talking about. He kept giving me the logical parameters and insisting they were the physical ones. Finally, I asked him how his drive could have 16 heads when it had only 3 platters, and he got very confused. :^)

Tip: It's easy to tell if you are looking at physical or logical hard disk geometry numbers. Since no current hard drive has the same number of sectors on each track, if you are given a single number for "sectors per track", that *must* be a logical parameter. Also, I am aware of no current hard disk product that uses 8 platters and either 15 or 16 heads. However, all modern, larger IDE/ATA hard disks have a nominal logical geometry specification of 15 or 16 heads, so either of those numbers is a dead giveaway.

Next: Logical Geometry

Logical Geometry

When you perform a drive parameter autodetection in your system BIOS setup or look in your new IDE/ATA hard disk's setup manual to see what the drive parameters are, you are seeing the *logical geometry* values that the hard disk manufacturer has specified for the drive. Since newer drives use zoned bit recording and hence have ten or more values for sectors per track depending on which region of the disk is being examined, it is not possible to set up the disk in the BIOS using the physical geometry. Also, the BIOS has a limit of 63 sectors per track, and all newer hard disks *average* more than 100 sectors per track, so even without zoned bit recording, there would be a problem.

Older hard disks that had simple structures and low capacity did not need special logical geometry. Their physical and logical geometry was the same. Take for example the Seagate ST-251, a 42.8 MB drive that was one of the most popular drives of its day. This drive's "CHS" physical geometry numbers are 820 cylinders, 6 heads, and 17 sectors, and those numbers are what is used by a system that has this drive.

Newer drives cannot have their true geometries expressed using three simple numbers. To get around this issue, for disks 8.4 GB or smaller, the BIOS is given bogus parameters that give the approximate capacity of the disk, and the hard disk controller is given intelligence so that it can do automatic translation between the logical and physical geometry. The actual physical geometry is totally different, but the BIOS (and your system) need know nothing about this. Here's an example showing the difference between the physical and logical geometry for a sample drive, a 3.8 GB Quantum Fireball TM:

Specification	**Physical Geometry**	**Logical Geometry**
Read/Write Heads	6	16
Cylinders (Tracks per Surface)	6,810	7,480
Sectors Per Track	122 to 232	63
Total Sectors	7,539,840	7,539,840

If you install this drive, as far as the system is concerned, the disk has 16 heads and 63 sectors on every track, and the hard disk itself takes care of all the "dirty work" of translating requests to their real internal locations. The physical geometry is totally hidden from view. The fact that both geometries equate to the same number of total sectors is *not* a coincidence. The purpose of the logical geometry is to enable access to the entire disk using terms that the BIOS can handle. The logical geometry could theoretically end up with a smaller number of sectors than the physical, but this would mean wasted space on the disk. It can never specify *more* sectors than physically exist, of course.

Another way to get around the problem of complex internal geometry is to change the way the drive is addressed completely. Instead of using the logical geometry numbers directly, most modern drives can be accessed using *logical block addressing* (*LBA*). With this method a totally different form of logical "geometry" is used: the sectors are just given a numerical sequence starting with 0. Again, the drive just internally translates these sequential numbers into physical sector locations. So the drive above would have sectors numbered from 0 to 7,539,839. This is just yet another way of providing access to the same sectors. You can read more about LBA here.

Today's drives are over 8.4 GB in size and have therefore run into an important hard disk capacity barrier: the 8.4 GB (7.8 GiB) capacity barrier. The largest logical parameters that can be used for accessing a standard IDE/ATA drive using normal Int 13h BIOS routines are 1,024 cylinders, 256 heads, and 63 sectors. Since the ATA standard only allows a maximum of 16 for the number of heads, BIOS translation is used to reduce the number of heads and increase the number of cylinders in the specification (see here for details on this). The practical result of all of this, is that the largest logical geometry numbers for IDE/ATA drives are 16,383 cylinders, 16 heads and 63 sectors. This yields a maximum capacity of 8.4 GB.

Drives larger than 8.4 GB can no longer be accessed using regular BIOS routines, and require extended Int 13h capabilities. There is no way to even *represent* their full capacity using regular IDE/ATA geometry numbers. Therefore, these drives just specify 16,383 cylinders, 16 heads and 63 sectors to the BIOS for compatibility. Then, access to the drive is performed directly by the Int 13h extension routines, and the logical parameters are completely ignored. Here's how a modern drive, the 34.2 GB IBM Deskstar 34GXP (model DPTA-373420), looks:

Specification	**Physical Geometry**	**Logical Geometry**
Read/Write Heads	10	16
Cylinders (Tracks per Surface)	17,494	16,383
Sectors Per Track	272 to 452	63
Total Sectors	66,835,440	16,514,064

As you can see, the logical and physical geometries clearly have nothing to do with each other on drives this large, and even the total number of sectors is wrong in the logical geometry. The drive *must* be accessed directly by an operating system supporting Int 13h BIOS extensions to see the whole drive, or drive overlay software used. If the drive is addressed using conventional geometry parameters, it will be limited in capacity to only 8.4 GB, which in the case of this drive would mean wasting over 75% of its capacity. Since they cannot have their true capacity expressed in terms of even *logical* geometry, all large modern drives are accessed using logical block addressing.

The translation between logical and physical geometry is the lowest level of translation that occurs when using a modern hard disk. It is different from BIOS translation, which occurs at a higher level and is described in the next section.

Next: BIOS Geometry Translation

BIOS Geometry Translation

The use of logical hard disk geometry gets around the problem that physical hard disk geometries cannot be properly expressed using simple BIOS settings. Logical geometries don't go far enough, however. In most cases, higher levels of translation are needed as well, because other problems relating to old design decisions make it impossible for even the logical geometry to be used with modern large hard disks. These are the infamous BIOS capacity barriers such as the 504 MB limit on standard IDE/ATA hard disks, the 8.4 GB limit for standard Int 13h BIOS addressing, and other similar issues.

In order to get around these barriers, another layer of translation is often applied on top of the geometry translation that occurs inside the hard disk. This translation is performed by the BIOS (or sometimes, third-party overlay software.) There are many issues involved in BIOS-level translation; the matter is discussed in detail here.

Next: Error Management and Recovery

Hard Disk Error Management and Recovery

Many people don't realize that it is normal for a hard disk to encounter *errors* during reading, as part of its regular operation. As hard disks are pushed to the limits of technology, with tracks and sectors spaced closer together, weaker signals used to prevent interference, and faster spin rates produced by the spindle motor, the chances of an error occurring while reading the disk go up dramatically. In fact, the state of technology has advanced to the point where it is not practical to even try to avoid them.

Of course having actual errors appear to *you* while *using* the hard disk is unacceptable, since you count on your disk reproducing the data you store on it reliably, for a period of years. Hard disk manufacturers know how important this is, and so incorporate special techniques that allow them to detect and correct hard disk errors. This allows them to make faster, higher-capacity drives that appear to the user to be error-free. The more the technology for storing data is pushed, the more sophisticated the error correction protocols must be to maintain the same level of reliability.

Making a drive that actually produced read errors infrequently enough that error detection and correction wasn't necessary, would mean greatly reducing performance and capacity. This is sort of like touch-typing: there's a school of thought that says "if you aren't making any mistakes at all, you're going too slow". If correcting mistakes is easy, as it is with a word-processor, it's better to type 100 words per minute and correct an error or two, than to type 75 words per minute error-free. As long as the errors are detectable and correctable, and they don't occur too often, it's better to plan for them, and then tolerate and deal with them, than to be too conservative in order to eliminate them.

Note: The errors we are talking about here are those related to reading correct information off the disk, not issues like head crashes or motor burn-outs or other hardware problems. Similarly, I don't get into the discussion of general reliability and failure issues here, nor related technologies such as SMART. See the section on hard disk quality for more on these related issues.

Next: Error Correcting Code (ECC)

Error Correcting Code (ECC)

The basis of all error detection and correction in hard disks is the inclusion of redundant information and special hardware or software to use it. Each sector of data on the hard disk contains 512 bytes, or 4,096 bits, of user data. In addition to these bits, an additional number of bits are added to each sector for the implementation of *error correcting code* or *ECC* (sometimes also called *error correction code* or *error correcting circuits*). These bits do not contain data; rather, they contain information about the data that can be used to correct any problems encountered trying to access the real data bits.

There are several different types of error correcting codes that have been invented over the years, but the type commonly used on PCs is the *Reed-Solomon* algorithm, named for researchers Irving Reed and Gustave Solomon, who first discovered the general technique that the algorithm employs. Reed-Solomon codes are widely used for error detection and correction in various computing and communications media, including magnetic storage, optical storage, high-speed modems, and data transmission channels. They have been chosen because they are easier to decode than most other similar codes, can detect (and correct) large numbers of missing bits of data, and require the least number of extra ECC bits for a given number of data bits. Look in the memory section for much more general information on error detection and correction.

When a sector is written to the hard disk, the appropriate ECC codes are generated and stored in the bits reserved for them. When the sector is read back, the user data read, combined with the ECC bits, can tell the controller if any errors occurred during the read. Errors that can be corrected using the redundant information are corrected before passing the data to the rest of the system. The system can also tell when there is too much damage to the data to correct, and will issue an error notification in that event. The sophisticated firmware present in all modern drives uses ECC as part of its overall error management protocols. This is all done "on the fly" with no intervention from the user required, and no slowdown in performance even when errors are encountered and must be corrected.

The capability of a Reed Solomon ECC implementation is based on the number of additional ECC bits it includes. The more bits that are included for a given amount of data, the more errors that can be tolerated. There are multiple tradeoffs involved in deciding how many bits of ECC information to use. Including more bits per sector of data allows for more robust error detection and correction, but means fewer sectors can be put on each track, since more of the linear distance of the track is used up with non-data bits. On the other hand, if you make the system more capable of detecting and correcting errors, you make it possible to increase areal density or make other performance improvements, which could pay back the "investment" of extra ECC bits, and then some. Another complicating factor is that the more ECC bits included, the more processing power the controller must possess to process the Reed Solomon algorithm. The engineers who design hard disks take these various factors into account in deciding how many ECC bits to include for each sector.

If you are interested, *take this link to read more about the theory underlying ECC*. Note that some of this information is complicated to read.

Next: Read Error Severities and Error Management Logic

Read Error Severities and Error Management Logic

The hard disk's controller employs a sequence of sophisticated techniques to manage errors that occur when reading data from the disk. In a way, the system is kind of like a troubleshooting flowchart. When a problem occurs, the simplest techniques are tried first, and if they don't work, the problem is escalated to a higher level. Every manufacturer uses different techniques, so this is just a rough example guideline of how a hard disk will approach error management:

1. **ECC Error Detection:** The sector is read, and error detection is applied to check for any read errors. If there are no errors, the sector is passed on to the interface and the read is concluded successfully.
2. **ECC Error Correction:** The controller will attempt to correct the error using the ECC codes read for the sector. The data can be corrected very quickly using these codes, normally "on the fly" with no delay. If this is the case, the data is fixed and the read considered successful. Most drive manufacturers consider this occurrence common enough that it is not even considered a "real" read error. An error corrected at this level can be considered "automatically corrected".
3. **Automatic Retry:** The next step is usually to wait for the disk to spin around again, and retry the read. Sometimes the first error can be caused by a stray magnetic field, physical shock or other non-repeating problem, and the retry will work. If it doesn't, more retries may be done. Most controllers are programmed to retry the sector a certain number of times before giving up. An error corrected after a straight retry is often considered "recovered" or "corrected after retry".
4. **Advanced Error Correction:** Many drives will, on subsequent retries after the first, invoke more advanced error correction algorithms that are slower and more complex than the regular correction protocols, but have an increased chance of success. These errors are "recovered after multiple reads" or "recovered after advanced correction".
5. **Failure:** If the sector still cannot be read, the drive will signal a read error to the system. These are "real", unrecoverable read errors, the kind that result in a dreaded error message on the screen.

Any problems occurred during a read, even if recovery is successful, are potentially cause for concern, and error notification or logging may be performed.

Even before the matter of actually reading the data comes up, drives can have problems with locating the track where the data is. Such a problem is called a *seek error*. In the event of a seek error, a similar management program is instituted as that used for read errors. Normally a series of retries is performed, and if the seek still cannot be performed, an unrecoverable seek error is generated. This is considered a drive failure, since the data may still be present, but it is inaccessible.

Every hard disk model has analysis done on it to determine the likelihood of these various errors. This is based on actual tests on the drive, on statistical analysis, and on the error history of prior models. Each drive is given a rating in terms of how often each error is likely to occur. Looking again at the Quantum Fireball TM, we see the following error rate specifications:

Error Severity	**Worst-Case Frequency of Error (Number of Bits Read Between Occurrences)**
Automatically Corrected	Not Specified
Recovered Read Errors	1 billion (1 Gb)
Recovered After Multiple Reads (Full Error Correction)	1 trillion (1,000 Gb)
Unrecoverable Read Errors	100 trillion (100,000 Gb)

Drives also typically specify the rate of data *miscorrection*. This situation arises if the ECC algorithm detects and corrects an error but itself makes a mistake! Clearly this is a *very* bad situation, since an error would be returned to the system and the fact that an error occurred would not even be known. Fortunately, it is very, very rare. A typical value for this occurrence is less than 1 bit in 10^{21}. That means a miscorrection occurs every trillion gigabits read from the disk--on average you could read the entire contents of a 40 GB drive over a million times before it happened!

I find the numbers above--even the "smaller" ones--pretty impressive. While your hard disk does a lot of reads and writes, 100,000 gigabits is a pretty enormous number! This is why the reliability of modern hard disks is so high. Interestingly, the error rates on drives haven't changed all that much in the last few years. Presumably, any improvements in error rates are "used up" by pushing the performance envelope. Meanwhile, the reliability concerns associated with individual drives are typically addressed through the use of multiple drive arrays.

Next: Error Notification and Defect Mapping

Error Notification and Defect Mapping

Many drives are smart enough to realize that if a sector can only be read after retries, the chances are good that something bad may be happening to that sector, and the next time it is read it might not be recoverable. For this reason, the drive will usually do something when it has to use retries to read a sector (but usually not when ECC will correct the problem on the fly). What the drive does depends on how it is designed.

Modern drives support SMART, a reliability feature that tries to predict drive failure based on technological "leading indicators". Read errors, excessive numbers of retries, or problems seeking are very commonly included in the set of parameters used to signal impending hard drive doom. Activity of this sort that exceeds a safety threshold determined by the drive's designers may trigger a SMART warning, telling the user that the drive may be failing.

Today's hard disks will also often take corrective action on their own if they detect that errors are occurring. The occasional difficulty reading a sector would typically be ignored as a random occurrence, but if multiple retries or other advanced error correction procedures were needed to read a sector, many drives would automatically mark the sector bad and relocate its contents to one of the drive's spare sectors. In doing so, the drive would avoid the possibility of whatever problem caused the trouble worsening, and thereby not allow the data to be read at all on the next attempt.

Next: Hard Disk Performance, Quality and Reliability

Hard Disk Performance, Quality and Reliability

When considering the actual "real world" daily use of hard disks, and contemplating a hard disk purchase, PC users often ask three key questions:

- Is this hard disk fast?
- Is this hard disk well manufactured?
- Is this hard disk going to last?

These questions are answered by considering carefully the three words that appear in the title of this page. Hard disk performance is important because hard disks are one of the slowest internal PC components, and therefore often limit the performance of the system as a whole. Quality and reliability are *critical* with hard disks, because they are where your data resides! No other PC component can lead so readily to disaster if it fails.

This section is quite large, and it has to be. Why? Because these matters are so crucial, and unfortunately, few of the "big players" in the industry do a good enough job of educating hard disk users about their products. Here I am talking especially about those who make hard disks, whole PCs and operating systems. Hard disks continue to be sold by flashing appetizing numbers on boxes, placed there to entice consumers who can't easily understand where they fit into the big picture of overall performance. Benchmarks are tossed around the Internet with little knowledge and even less scientific control. And hard disk quality and reliability remain mysteries to many users, who buy whatever seems fast and cheap, and simply hope for the best.

It's my hope that after reading this section you will be in a much better position to evaluate the numbers and claims made by hard disk manufacturers and others. There are three major subsections contained here. The first takes a very detailed look at hard disk performance, discussing different ways that performance can be assessed, common performance specifications, ways of measuring performance, and also describing the various factors inside and outside the hard disk that affect its speed. The second looks at hard disk quality and reliability issues in detail, including a look at warranty issues and features being put into hard disks to improve their reliability. The third major subsection is devoted to the discussion of Redundant Arrays of Inexpensive Disks, or RAID. The use of disk arrays is increasing dramatically as the PC world seeks to improve performance, expand storage capacity, and improve the reliability of the storage subsystem.

Next: Hard Disk Performance

Hard Disk Performance

There was a time when the performance of hard disks was one of the most underrated aspects of overall system performance. Hard disks were considered only "a place to keep stuff" and little heed given to how they affected the operation of the PC as a whole. Over the last few years this has changed dramatically, and hard disk performance issues are now getting the attention they deserve. The problem with hard disk performance however is that it is not easy to understand. There are many different issues in how performance is assessed and measured. There are interactions between components with which anyone seeking to really grasp hard disk performance must contend. And the technology changes so rapidly that what is the "fastest" today will probably be "second best" within a few months--or even weeks in many cases!

The purpose of this section is to take a detailed look at all the facets of hard disk performance. I start by discussing various issues in how performance is evaluated and measured. Then, I cover in full detail the common hard disk performance specifications and also the internal and external factors that affect performance.

In order to avoid some confusion, let me try to explain the difference between "performance specifications" and "performance factors", at least as I see them. A performance *specification* is a figure provided by a manufacturer that attempts to express an element of the drive's performance relative to other drives on the market. A performance *factor* is something about the drive (internal factor) or its environment (external factor) that affects the performance of the drive. Clearly, these are interrelated; factors affect specifications, and specifications indicate the impact of various factors. I considered putting them all together, but this would be confusing, since some specifications sort of "stand alone" while others are derived from one or several performance factors. Also, some specifications are a result of various factors. To be honest, this was one of the most difficult areas of the site to organize.

Before even delving into this section, it's important to remember that performance always comes at a price. As with most things, there are tradeoffs involved. If performance is your chief goal in a storage system you can have it, but you will either pay for it, or give up something else in exchange (capacity, flexibility, simplicity, etc.) There are no free lunches, although the general increase in performance of *all* hard disks means that even inexpensive drives today perform very well compared to their predecessors.

Next: Hard Disk General Performance Issues

Hard Disk General Performance Issues

Alright, so you want to know about hard disk performance. But what exactly does "performance" mean? I'm not trying to get mystical on you. :^) It's just that performance means different things to different people, and in fact there are many different *kinds* of "performance". How important a particular aspect of performance is, depends to a great extent on how you are using your system, and what your needs are.

This introductory section looks at some of the basic issues involved in understanding hard disk performance. This includes a discussion of the importance of hard disk performance, a look at various ways performance can be considered, and a discussion of the relative importance of various performance specifications and factors.

Next: The Importance of Hard Disk Performance

The Importance of Hard Disk Performance

I keep insisting that the performance of the hard disk is important. But why? Surely the hard disk's performance level can't be as important as that of the CPU, or memory, or other core system components. Or can it?

It is true that for many aspects of computing, the hard disk's performance level is not much of an issue. If you are recalculating a massive spreadsheet, or doing complex rendering of 3D objects, the amount of sheer processing power in the system is of paramount concern; the hard disk will only come into play periodically. However, these sorts of specialty operations are not indicative of how most of us use our PCs. Typical PC use involves loading programs, and loading and saving data frequently. All of these operations require access to the hard disk. And therefore, hard disk performance becomes an issue constantly as we use our machines.

The importance of hard disk performance even goes beyond this however. After all, we also use all the *other* main components of our system constantly, so aren't they equally important to the performance equation? Well, yes and no. The importance of the CPU, motherboard and other core components *is* very important. But much as the strength of a chain is equal only to that of its weakest link, in many ways the performance of a system is only equal to that of its poorest-performing component. Compared to the solid state components in a PC, hard disks have by far the worst performance. And even as hard disks improve in speed, CPUs, video cards and motherboards improve in speed even faster, widening the gap. Thus, hard disks continue to constrain the overall performance of many systems.

In the amount of time it takes to perform one random hard disk access, one of today's CPUs can execute over a million instructions! Making the CPU fast enough to process *two* million instructions while it waits doesn't really gain you much unless it has something to do with that time. Only improving the hard disk's speed to reduce the wait time will get you where you want to go. Any time you see your hard disk's activity light flashing, the odds are that the rest of the system is twiddling its thumbs waiting for those slow mechanical components to "do their thing".

The applications where hard disk performance issues are most important are obviously those that do a lot of reading and writing to the hard disk, instead of doing a lot of processing. Such tasks are said to be "I/O bound", where "I/O" stands for "input/output". These tasks are contrasted to those described earlier which use the CPU a great deal and are called (unsurprisingly) "CPU bound". Multimedia editing applications, especially those dealing with large audio and video files, are probably the ones most affected by the speed of the storage subsystem. Also up there are applications that process files, including compilers and many disk utilities. Initially starting your PC is also a very I/O-intensive application, as the operating system loads literally hundreds of files. Improving hard disk performance can shave time off the boot process in a very palpable way.

The need for improved performance is a major driving factor behind the rise in popularity of RAID.

Next: Internal vs. External Performance

Internal vs. External Performance

The hard disk's job is to store data from the system, or get data to the system, as fast as possible. When considering performance, it is this ability to move data into or out of the hard disk that we are looking to measure. There are two separate parts of this data movement job. For a write, the data must be fetched from the system, and then written to the correct sector(s) on the disk. For a read, the process is reversed; data must be read from the disk, and then transmitted over the hard disk interface to the system.

Clearly, the hard disk itself is only responsible for some portion of the overall performance we attribute to the hard disk subsystem. Some of the factors that affect performance are related to characteristics of the PC that are not specific to the particular hard drive being used. Performance characteristics that are largely a matter of how the hard disk itself is designed and implemented I call *internal performance factors*; those that are mostly affected by the rest of the system, or how the hard disk is used (and hence are largely independent of the particular hard disk model) are *external performance factors*.

The distinction between internal and external is a very important one! In any system, the bottleneck to high performance can reside either within the disk, or within the rest of the system (the interface, the system bus, CPU, drivers, file system, and so on.) It's usually not in both at the same time. If the main limiting factor in a particular system is, say, the system bus being used for the hard disk interface, putting a faster hard disk into that system will have very little impact on performance. Similarly, if the hard disk itself is slow, putting it on a faster interface will yield little improvement.

In some cases, performance bottlenecks can change from being primarily affected by internal factors, to being more influenced by external factors, depending on what type of work is being done on the PC. In addition, making a change to a system can cause the bottleneck to "shift". Let's say you have a high-speed hard disk that is on a slow interface. The interface may be the bottleneck to high performance; if you move the disk onto a faster interface, the disk itself may become the bottleneck. In many PCs, external performance can be enhanced simply by making no-cost performance enhancements to the system.

In this discussion of hard disk performance, the important distinction between internal and external factors is the reason why they are in separate sections; internal performance factors are discussed here; and external performance factors are here.

Next: Positioning vs. Transfer Performance

Positioning vs. Transfer Performance

As discussed here, the process of reading or writing to the hard disk really comprises two large parts: the work done by the drive itself (measured by internal performance factors) and the rest of the system (external factors). The internal work of the drive can itself be thought of as two functions: finding the correct location on the drive, and then reading or writing the data. These are very different jobs, and two drives can be very similar in one regard but very different in another. They also depend on different design characteristics. I call these two different tasks *positioning* and *transfer*, and their performance *positioning performance* and *transfer performance*.

Both of these are important to overall performance, although if you read the literature and the numbers that people talk about, positioning metrics are probably more commonly discussed than transfer measurements. You might be fooled by this into thinking they are more important, but often they are not--they are just simpler to explain in many cases, or people are used to using them to compare drives.

Which influences on performance are most important also depends on how you are using the device. If you are running a file server, the hard disk will be doing a lot of random accesses to files all over the disk, and positioning performance will be extremely important. If you are a single user doing multimedia editing where you need to read multi-gigabyte consecutive files as fast as possible, data transfer is far more important than positioning speed.

Most of the performance specifications that hard disk manufacturers provide to describe their products can be broken down into categories by which aspect of performance they measure. I have designed the section on performance specifications with this in mind: there are sections discussing positioning performance specifications and transfer performance specifications. In addition, there are two key specifications that reflect aspects of both positioning and transfer. There are also some specifications that don't really fit into these categories.

Next: Read vs. Write Performance

Read vs. Write Performance

Hard disks can of course both read and write data (which isn't true of all storage devices.) The performance of a hard disk isn't exactly the same when it is doing a read as when it is doing a write, however. For some performance measurements there is no difference in how the system performs when doing a read or a write; for example, the platters are always spinning at the same speed, and so the latency of the drive doesn't change depending on what the heads are doing. Other measurements though, such as seek time, are different for reads as opposed to writes.

Almost all performance specifications given for hard disks are based upon how the hard disk performs while reading, not while writing. This is probably because hard disks spend more time reading than writing, and also because hard disks are generally *faster* when reading than when writing, so the numbers look better.

Some companies provide explicit write specifications in addition to their read specifications, while others do not. The most important specification that differs between reads and writes is seek time--a good rule of thumb is that the average seek time for writes on a hard disk is about 1 millisecond higher (slower) than the specification for reads. If a particular hard disk model doesn't mention the numbers you are interested in for writes, and if write performance is particularly important for your application, contact the manufacturer's technical support department. Someone there will know the answer, if you can get a hold of the right person. :^) It may be easier to try downloading the product manual for your model from the manufacturer's web site.

Next: Component vs. System Performance

Component vs. System Performance

As with every component of the PC, the hard disk is but one part of an integrated whole. It is not possible to measure the performance of the hard disk in isolation, since running any benchmark program involves using the processor, memory and other parts of the PC. The only way to isolate the hard disk would be if you were to use specialized test hardware, connected directly to the hard disk itself, and then you'd have a hard time being sure that the results really related to "real world" PC performance at all.

Many benchmarks are designed to try to isolate the hard disk from the rest of the system to test "only" the hard disk. Some are more successful than others in doing this. Unfortunately, many of them don't take all the factors into account and end up (for example) testing the system's hard disk cache instead of the hard disk itself. They are getting smarter over time, however, but still, virtually every hard disk benchmark I have ever seen has allowed the rest of the system to impact the number. You take the hard disk out of a Pentium II 300 PC and put it into a Pentium III 600 system, run the benchmark again, and the score goes up.

As with other components, the best way to compare two hard disks is still a comparative benchmark. Set up the system they are intended to be used in, test one with a benchmark that represents the intended use, and then replace it with the second drive, retest and compare. This eliminates much of the "background noise" that is associated with absolute benchmark numbers. Most better review sites do exactly this, maintaining constant "test benches" with hardware that does not change between tests of various drives.

Another thing that the matter of "component vs. system" means is that there is no way to (legitimately) compare directly two drives that run under different interfaces or in different systems entirely. In particular, you can't directly compare the speed of an IDE/ATA drive to that of a SCSI drive and be sure that you are measuring *only the drive*. Some would rightly point out that the distinction isn't really that important however, since the "end result" of system performance is really most important.

As discussed here, hard disks have a very substantial impact on the performance of the system as a whole.

Next: Ranking Performance Specifications and Factors

Ranking Performance Specifications and Factors

Amongst those who are concerned with hard disk performance, there has been considerable attention given lately to *ranking* the various performance specifications and factors by how important they are for a hard disk. The reason for this is usually to assist in choosing hardware: if you are comparing two hard disks that have similar specifications, how do you decide which one is likely to be better? Of course, reading reviews and benchmarks is one part of the answer. It's still useful at times to have a way to rank the various specifications in terms of which is most important to overall performance.

There's just one *little* problem with this: it's virtually impossible to do. Which specification is most critical depends entirely on how you use your hard disk and your PC as a whole. People disagree greatly on the matter of which specification or performance factor translates most directly into improved overall performance. There is no one "correct answer".

The biggest argument seems to be over which is more important, positioning or transfer. The answer is: it depends. If you are doing a lot of work with large files, for example editing video or audio streams, transfer is typically more important. Few people fall into this category, however. If you are running a server or other application where many people are accessing smallish files on a regular basis, positioning is definitely more important. Even fewer people fall into this category. :^)

Beyond the special cases, things get much more complex. While transfer rates have typically been under-emphasized in recent years, lately some tests have shown that they are now often being over-emphasized. Since most operating systems now use a large number of different files and drivers, and since effects such as fragmentation cause parts of files to be spread over the surface of the disk, many users access their disks less often in a sequential manner than was previously thought.

In my opinion, it is best to value both and not worry excessively over the issue. :^) To this end, when selecting a hard disk I believe the most important things to look at are spindle speed, areal density, and seek time. The first two are important because they have an influence on both positioning and transfer performance: other specifications generally do not. Higher spindle speed in particular is the most obvious indicator of a faster hard disk; a 7200 RPM hard disk of the same age and interface as a 5400 RPM drive will almost always be faster, not just due to the faster spindle but also due to other improvements that appear first on faster-spindled drives. Seek time is important because of the issue of reading large numbers of scattered small files mentioned above.

Another key point to keep in mind: if two hard disks are very close in specifications, then most likely the two drives will perform similarly. It's not worth spending a lot of energy worrying about small discrepancies between them. See here for more on this subject.

Next: Hard Disk Performance Measurement

Hard Disk Performance Measurement

There are many issues involved in measuring the performance of hard disk drives. For starters, there are different ways it can be measured, there are different ways to do it, and important considerations to keep in mind when making the measurements.

The process of measuring the performance of a system or component is called *benchmarking*. Its intention is to express the overall performance of a piece of hardware in numeric terms. Benchmarking can provide useful data if it is done properly, and kept in perspective. Unfortunately, benchmarking is a process that is greatly prone to error and to exaggeration. In order to use benchmarks wisely, one must have a proper understanding of the factors in measuring performance. It is also important to keep in mind that benchmarking is not the only way to look at the matter of performance.

This section discusses some of the issues that are specifically relevant to the matter of measuring the performance of hard disk drives. You will also find this more general article on benchmarking informative

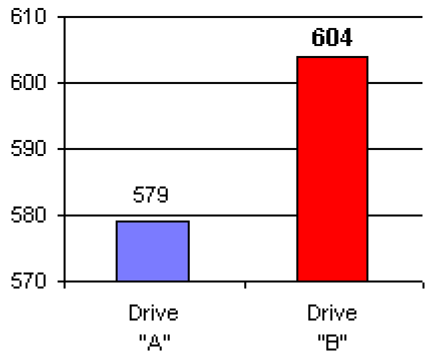
Next: Putting Performance Measurement In Perspective

Putting Performance Measurement In Perspective

"There are three types of lies: lies, damned lies, and statistics." -- Benjamin Disraeli

I seem to have a habit of taking somewhat controversial stands that fly in the face of what most other hardware writers say on various subjects. This page will certainly be no exception. :^) Here it is, blurted out in simple terms: hard disk performance is important, but hard disk performance *measurement* is often overrated and over-emphasized. The Internet is filled with discussions about various benchmarks; arguments over which benchmarks are better and worse; people getting upset when their hardware gets benchmarks scores that are "too low"; review sites recommending one hard disk over another on the basis of it scoring a few percentage points over another in a synthetic test; and so on. In my opinion, most of this has little relevance to the way most people use their PCs, and the people who are so concerned about "which hard disk to get" when they are looking at comparable models are worrying too much.

Sure, it's useful to see benchmark scores on hardware before you plan a purchase. But you should keep them in perspective. An important rule of thumb in the PC industry is that in real-world computer use, most PC users don't notice an increase (or decrease) in performance of less than about 10%. That means that even if you could prove without a shadow of a doubt that hard disk "A" was 7% faster than hard disk "B", you probably wouldn't really notice any difference in your system. And in most cases, competitive products put out at about the same time by high-end manufacturers often don't differ in performance by more than 10%. The biggest mistake people make is to over-value the numbers they read about various hardware benchmarks. Use them as a rough guideline only. A month down the road you probably won't even remember what "your benchmark scores" were, and they certainly won't matter much to you. I've been down this road, believe me.



Here's an example of the type of benchmark chart you will run into all too frequently (and one that would make Disraeli shake his head). At first glance, drive "B" appears to blow the doors off of drive "A", right? In reality, the numbers differ by less than 4%--not even noticeable to most users. The scale has had its zero point "chopped off" to magnify the differences between the numbers. Note also the other marketing tricks: the larger number has its column in bright red to attract the eye; the smaller number is a plain light blue. The larger figure is also in bold and a larger font has been used.

Also remember that there are other attributes besides performance that are important for selecting hardware. Lots of people try to get the very fastest hard disk but don't consider other equally important issues: quality, reliability, warranty, and data backup. People agonize over which hard disk is a teeny bit faster than another--and then never defragment their file system, or fill the hard disk up with junk so it runs less efficiently. Be sure to keep the big picture view.

Finally, bear in mind that whatever is on the top of the hill in the hard disk world doesn't stay there for long. Sure, it's a good feeling to think you are getting *the* fastest disk around. But every few months, a new model comes out that is faster than anything that preceded it. If you really want to always have the best hard disk, you have to keep buying more hardware, which is an expensive proposition that few opt for. See here for more on this subject.

Next: Objective Performance Measurement (Benchmarking)

Objective Performance Measurement (Benchmarking)

Benchmarking hardware is a form of *objective* performance measurement; it is measurement based on logic and analysis, as opposed to subjective measurement, which is more of a "feel" method of gauging performance. Benchmarking is typically done using benchmark programs specifically developed for the purpose of measuring hard disk performance.

There are many different programs used to ways to benchmark hard drives, and they generally fall into the following different categories:

- **High-Level (Application-Derived) Benchmarks:** These are programs that use code from popular applications software--usually office applications, web browsers and the like--to simulate the impact of hard disk performance on the use of those applications in the real world. The basic idea is to run a suite of tests that are comprised of typical actions that a user would take while using those applications, and time how much time elapses with the hardware in question. Then the hardware is changed and the test run again. This is generally a good concept for benchmarking, but only has relevance if you are actually using the types of applications around which the benchmarks is designed. If you are primarily a gamer for example, what do you care about the performance of spreadsheet software? Also, since the benchmark is running at a high level, there is a lot of room for interference from operating system and file system issues. One of the most common benchmarks of this type is the ZDNet WinBench series.
- **Low-Level (Synthetic) Benchmarks:** These programs attempt to test the hard disk directly, isolating it as much as possible from the rest of the system. They are often called "synthetic" because they don't try to reproduce the access patterns of real applications, instead using artificial patterns created by the programmers specifically for the benchmark, to test different types of hard disk use. They are often derided as being unrealistic because of their synthetic nature, and much of this criticism is in my opinion accurate. At the same time however, they provide much more control over the test process than application-derived benchmarks. This control lets you better "focus in" on one particular aspect of performance and more accurately compare different hardware units in a number of different areas. Common disk benchmarks of this variety include Adaptec's Threadmark and Intel's IOMeter.
- **"Real-World" Benchmarks:** These are not "formally" benchmarks, but are commonly used by hardware enthusiasts to compare real-world performance of hard disks. The idea is simple: take something that you do often, measure how long it takes with one drive, and then how long it takes with another. For example, if you have a system with an old hard disk that is very slow to boot up, measure how long it takes and then repeat the process with a newer disk to see how much things improve. In some ways these are the most realistic, and also the most *relevant* benchmarks. However, they are entirely system-dependent and therefore of no use whatsoever in communicating much in objective terms about the power of the hardware in question: the improvement you see between hard disk "A" and hard disk "B" on

your system may be very different than the same hardware used on a friend's PC. Also, these measurements are usually fairly crude and can't be done on activities that take relatively little time, since the timing is often done with a regular clock or wristwatch.

As I've said elsewhere, I'm not a big fan of benchmarks, especially when it comes to hard disks. While they have their uses, it's too easy to succumb to the temptation to view them as absolute indicators of performance, to overvalue them and not consider what they really mean, bottom line, for the typical end user. Small differences in hard disk performance have *virtually no impact on the typical hard disk user*. Some people really get carried away, sweating over every percentage point of their favorite benchmark, as if it were a competition of some sort (and for some people, I suppose it is--a competition for bragging rights.) Even leaving the matter of over-emphasizing benchmarks aside, there are some common "benchmark traps" I see all the time:

- **Poor Control Of Environmental Factors:** The only way to properly compare two pieces of hardware is to test them under *identical* conditions. Even seemingly irrelevant issues can influence the outcome. Most better hardware sites understand this, but many individual enthusiasts do not. The exact number you get from testing one drive on your system can be very different from the number someone else gets with the same drive, without this meaning anything is "wrong".
- **Small Sample Size:** All benchmarks have a tendency to produce different numbers if you run them more than once. To properly use a benchmark it must be run several times and the results averaged. It's even better to run at least five times and discard both the highest and lowest score for each piece of hardware.
- **Paying No Attention To Cost:** You will frequently see people talk about the "benchmark X" score of one drive versus another, but when's the last time you saw anyone take the ratio of two drives' respective benchmarks to their current market prices? I've seen people recommend "drive A" over "drive B" due to a difference in performance of well under 10% despite "drive A" costing 50% more than "drive B". That's rarely money well-spent.
- **Benchmark (In)Validity:** It's not uncommon to see a particular benchmark be used for a long time by many people... and then it is discovered that due to a flaw in how it is written, or the manner in which it interacts with the hardware, operating system or drivers, that its results were inaccurate or misleading. Another reason to use benchmarks only as guidelines. Next: Subjective Performance Measurement

Subjective Performance Measurement

The opposite of objective performance measurement is *subjective* performance measurement. This technique dispenses with the use of benchmarks in favor of more intangible issues of assessing performance. In a nutshell, this method of measurement is based upon the system's usability or "feel" under different hardware conditions. It's not nearly as commonly discussed as objective measurement, for a couple of obvious reasons. It's much harder to quantify, to get a "handle" on than benchmarking. Subjective measurement also doesn't provide you with neat, easily-compared numbers. It's not really well-suited for testing hard disks in a general way that will be applicable to people reading an analysis or review of a piece of hardware.

The guiding principle behind subjective performance measurement is this: "If you can't tell the difference, what does it matter?" As such, subjective evaluation is a very personal matter, best suited for an individual to use in assessing the performance of a particular piece of hardware. The best benchmark is always using your own system with your own software. If you change a piece of hardware and it doesn't make a difference big enough to really impact your use of the system, then the change is probably not worthwhile.

The main problem with subjective measurement is that you can't always easily "test drive" hardware. If you have a slow hard disk and a friend is willing to let you borrow a faster one for a "trial run" then that's great--take advantage of it. Few of us have such a luxury, but fortunately, it's not strictly necessary. With subjective measures we are not dealing with specific numbers but rather "big picture" performance. As such, you can learn vicariously from others' experiences. Let's suppose a friend has a system that is similar to yours and has a similar "feel" when you use it. He upgrades to a new hard disk and suddenly the system feels much faster to you. If this is the case, the chances are good that you will experience a similar improvement in the usability of your own PC if you upgrade.

Next: Hard Disk Performance Made Easy

Hard Disk Performance Made Easy

If you don't want to bother to concern yourself over the intricacies of hard disk performance measurement, reading specifications and performance-affecting factors, and analyzing tables of benchmarks, there's a crude but very simple and easy way to ensure your system always has very good hard disk performance (not *the best possible* but very good). It's a technique that I call "hard disk performance in one step". Here's the one step: buy a new hard disk every 12 to 18 months; not necessarily the best one available at the time, but one that represents good value for the money based on the market at the time you make the purchase.

It sounds like I am being tongue-in-cheek, but I'm really not. There are people who actually take this approach, and it can be very effective, if not "optimal". The reason that it works is due to the nature of the hard disk industry. Performance increases so rapidly, and the cost per gigabyte of storage *decrease* so rapidly, that the hard disks of a given class and price range available today are almost always far superior to the ones available a year or two earlier. If you remember nothing else remember this: the difference in performance between *any* new drive of a given type and *any* 18-month old drive of the same type, is far greater than the differences between current offerings of the same type from competing manufacturers. You could pick a 2000 model hard disk *at random* and it would be a better performer than the top drive of its class and price range from 1998.

Is it too expensive to do this? Well, for some people it is; for others it definitely is not. A good-quality hard disk of a reasonable size costs under $150 these days. It's largely a matter of figuring out what your time is worth. If it saves you hours and hours of research, this may be a worthwhile option for you.

In a similar vein, you should avoid the syndrome that plagues many who make very occasional hard disk purchases. They spend tons of time researching hard drives and buy what is at that time the fastest unit around, often paying far more for it than a drive that is slightly slower. Then they use this same drive for two or three years. Within six months their drive is no longer the fastest on the market; within 18 months it is slower than drives selling for far less than that drive cost when new (and the newer drives have double or triple the capacity to boot).

Am I trying to convince you to buy a new hard disk every year? No, I'm not. I have a frugal streak and certainly wouldn't suggest spending money on hardware without good cause. I'm trying to point out that hard disk performance improves so quickly and so regularly that trying to pick the perfect drive at any given time is often not worth the effort. I'm also saying that if performance is really important to you, you don't want to keep using the same drive year after year after year.

Next: Hard Disk Performance Specifications

Hard Disk Performance Specifications

Hard disks are often compared by contrasting their specifications. This is a valid way of attempting to discern whether one hard disk is better than another, though due to the large number of specifications often provided, this can also be somewhat difficult to do while being sure to cover all the bases and perform the analysis in a balanced way. It helps to focus on the more important specifications, of course. Certainly, understanding hard disk specifications is essential to understanding hard disk performance.

Structuring a discussion of performance specifications and performance factors in a way that covers all the issues in a way that's not too confusing isn't easy. After (excessive) deliberation, I decided to create two separate sections, one covering specs and the other discussing factors. The difference between them, as I see it, is that a specification is a number or label that expresses an element of performance; a factor is something that causes performance, and hence a specification, to be affected. So specifications are indirectly derived from, and are a result of, the various design factors used to create a drive. A single specification can be affected by a number of different performance factors; and a single performance factor can influence a number of specifications.

Due to this "many-to-one and one-to-many" relationship between these two concepts, there is some overlap between the "specifications" and "factors" sections--there are headings and topics that are the same in both places. In this section, I am focused primarily on what the specifications are and what they represent, how to interpret them and what their importance is. I then relate each of these to the performance factors that influence them, and talk more about the direct impact on performance in the internal performance factors and external performance factors sections. I apologize in advance for any confusion this may cause! :^)

Note: I attempt in this area to be as comprehensive as practical in covering the most important and common hard disk specifications, but at the same time not overwhelm you with every single specification that exists on hard disks. You can find many more specifications in the product manual of a hard disk family, usually available free on the manufacturer's web site.

Next: General Notes On Performance Specifications

General Notes On Performance Specifications

Before diving into the specifications themselves, I want to discuss a couple of topics of relevance to hard disk performance specifications in general. Keep these in mind as you read the various pages describing the specifications themselves. Most of them are really *caveats* about the specifications and some of the little tricks some manufacturers use in providing them:

- **Look At *All* The Specifications:** Some manufacturers publish two "sets" of specifications. For "public consumption" they put out a short list of specs, often highlighting the areas they feel presents the drive in the best possible light. There's really nothing wrong with that, but if you are contemplating a purchase you want the whole story. Check the manufacturer's web site for a downloadable product manual or comprehensive data sheet that contains more detail.
- **Watch For "Maximum":** Some performance characteristics vary depending on what part of the disk is being used, or in what way the disk is being exercised. Therefore, it is common for some specifications to be listed with the word "maximum" next to them. Don't gloss over this very important word! :^) What it really is telling you is that the *average* value of whatever that specification is will be lower. Make sure that if one manufacturer provides an "average" value for something, you don't compare it to a "maximum" value provided by another manufacturer. (Fortunately the specifications are mostly standardized between hard disk makers.)
- **A Matter Of Trust:** Bear in mind when you read the specifications for a device that the manufacturer is providing them. Usually companies are *mostly* honest about the numbers, but exaggeration is certainly not unheard of. Mistakes also occur on occasion; if a number seems too good to be true, verify it against another source before accepting it.
- **Don't Overvalue A Single Performance Metric:** Hard disk manufacturers often play the "magic number" game where they try to convince you that one or maybe two specifications show how great their product is. The culprit for this used to be seek time, which manufacturers overemphasized to the point of laughability--seek time *is* important, but it was still being incredibly "oversold". Today, the worst specifications in this regard are probably interface speed and cache (buffer) size. The latter in particular has been shown to have a rather insignificant impact on overall performance, but some companies act like a hard disk with a 2 MiB buffer will be twice as fast as one with a 1 MiB buffer.
- **Some Specifications Are Derived:** Some of the specifications I discuss here are actually derived from other ones, so you won't find them listed on most spec sheets. They sometimes do a better job of expressing performance than the "official" specs upon which they are based. I explain these in detail in the appropriate sections.
- **Read Vs. Write Performance:** Some performance specifications that manufacturers provide are accurate only when the drive is reading, not when it is writing; sometimes this is stated explicitly, and sometimes it is not! See here for more on this.

Next: Positioning Plus Transfer Performance Specifications

Positioning Plus Transfer Performance Specifications

The two specifications discussed in this section are given the "privilege" of being discussed first--like they care, right? :^)--for an important reason. They are the only specs that illustrate aspects of the performance of the hard disk in both fundamental ways: positioning and transfer. As such, they are very important to look for and understand. They certainly are not the only important specifications, and there are some knowledgeable people who rank other metrics higher than these, but few would dispute that they are essential enough that you should always look for them on any hard disk you evaluate.

Next: Spindle Speed

Spindle Speed

The hard disk spindle is the shaft upon which the platters are mounted; it is driven by the spindle motor, one of the most important components in the hard disk. Obviously, the faster the motor turns, the faster the platters spin. The spindle speed of the hard disk is always given in RPM (revolutions per minute). Typical speeds of drives today range from 4,200 RPM to 15,000 RPM, with 5,400 to 10,000 RPM being most common on desktop machines. See this operational discussion of spindle speed for a table of the most common speeds employed today and in the past, and a list of different applications that use them.

Spindle speed has gone from being one of the least-discussed to one of the most-discussed hard disk specifications in only a few short years. The reason is the creation of increasingly fast spindles. For the first several years that hard disks were used in PCs, they all had the same spindle speed--3,600 RPM--so there was literally nothing to talk about in this regard! Over time, faster drives began to appear on the market, but slowly, and starting with high-end SCSI drives not used in most systems. Once the trend started, however, and the obvious advantages of higher spin speeds became apparent, the trend accelerated. Still, it is only since about 1998 that mainstream IDE/ATA drives have been readily available for the desktop in spindle speeds higher than 5,400 RPM. The most common speeds today are 5,400 and 7,200 RPM, and 10,000 RPM IDE/ATA drives are likely just around the corner (since they are now standard on SCSI with the SCSI high-end moving to 15,000 RPM!)

Today, spindle speed is the first thing you really should look for when assessing a drive; the speed of the spindle is the primary method by which drives are categorized into "classes". Almost any 7,200 RPM drive will be faster, and more expensive, than a 5,400 RPM drive of the same size and generation. The spindle speed directly correlates to the drive's rotational latency, affecting positioning performance, and also influences the drive's internal transfer rate. However, there is more to this: the difference in speed between different classes of drives is due not only to the speed of the spindle, but the fact that manufacturers tend to design these drives to be faster in other ways as well, knowing they are targeting a market more concerned with all facets of performance.

The spindle speed is of course influenced primarily by the spindle motor's speed and power. However, there are other issues involved in designing higher-RPM drives: you can't just slap a faster motor into an existing model! The size and number of platters is also an important design consideration, and the areal density of the drive also has an impact--faster drives sometimes require reductions in density compared to slower drives.

In fact, the overall quality of the entire hard disk becomes much more critical the faster you spin the platters. Issues with higher-speed drives include increased noise and vibration, and cooling concerns, though these have improved greatly with second, third and subsequent generation high-speed drives.

Next: Areal Density

Areal Density

Areal density, sometimes also (imprecisely) called *bit density* or even just *density*, refers to the amount of data that can be stored in a given amount of hard disk platter space. It is one of the most important indicators of overall hard disk performance, though one that outside the PC enthusiast community is sadly under-discussed. If you do not understand what areal density is about, I would advise that you read this operation page discussing it in detail before continuing with this page.

Areal density is a two-dimensional measure calculated by multiplying two linear measures: recording density (bit density) and track density. The result is measured in bits per square inch (BPSI). Since densities today are in the billions of bits per square inch, the most commonly seen unit is "Gbits/in^2". Sometimes the two measures that comprise areal density, are specified separately; other data sheets don't show these components individually. It's much better to be able to evaluate the numbers separately, since they are very different in terms of how they reflect aspects of performance.

Areal density is strongly correlated to the transfer rate specifications of a drive. The higher the drive's areal density, in general, the higher its transfer rates will be, however, most of the improvement in transfer rate is due to increases in *bit* density, not track density. (When more bits are in a given length of track, the heads will read more data in a unit of time, assuming the spindle speed is constant.) If drive "A" has an areal density 5% lower than that of drive "B", but its bit density is 10% higher, it will have a higher transfer rate than drive "B".

Both bit density and track density have an impact on positioning performance. Increases in either one allow the data on the hard disk to be stored physically closer together on the disk. This reduces the distance that the read/write heads must seek to find different files on the disk, slightly improving seek time. Do keep in mind though that the improvements here are relatively small compared to the impact areal density has on transfer rates. Also, improvements only in track density don't do a lot to improve performance.

Areal density specifications are usually *maximum* specifications; look for the magic "M word" near the spec. The areal density will only be this high in certain regions of the disk. Modern drives use zoned bit recording to allow the areal density not to vary too greatly over the surface of the platter, but density will still be higher or lower in different parts of the disk. See the full discussion of areal density for more on this.

There's also a "rough cut" areal density measure commonly used when talking about hard drives or comparing one generation of drives to another. Often, the total formatted capacity of the disk will be divided by the number of platters, and the density of the drive discussed in terms of "GB per platter". For example, the 30 GB Maxtor DiamondMax Plus 40 is a three-platter drive; it's rough density then is 10 GB/platter, and that applies to all the members of that family. The IBM GXP75 family is 15 GB/platter, and so on.

This is a convenient short-hand and is useful when discussing drives, just keep in mind its limitations. For starters, it's rather crude, so it's only good for contrasting different generations of drives with big differences in density. Second, implied in the "GB/platter" measure is the size of each platter. A 10 GB/platter drive with 2.5" platters has *much* higher density than a 10 GB/platter drive using 3.5" platters. Also, some drives use only one side of one of their platters; the 15 GB DiamondMax Plus 40 for example uses two platters but only three of the four surfaces, so it is still a 10 GB/platter drive, not 7.5 GB/platter. (A better measure would be "GB per *surface*, but nobody seems to use that since most drives use both sides of each platter.)

The primary factors that influence areal density specifications are those that relate to data and recording: this means that *all* the factors discussed in this section are relevant. It is also influenced by the design and speed of the spindle motor; faster motors may require density to be reduced for reliability reasons.

Next: Positioning Performance Specifications

Positioning Performance Specifications

In this section I take a look at some of the more important hard disk specifications relevant to positioning performance. These include several very well-known and widely quoted specs, along with some specifications that are a bit more obscure but often just as important.

Note that these specifications are not listed in any particular order. (OK, I *did* put seek time first since it is so important and interesting to talk about. :^))

Next: Seek Time

Seek Time

The *seek time* of a hard disk measures the amount of time required for the read/write heads to move between tracks over the surfaces of the platters. Seek time is one of the most commonly discussed metrics for hard disks, and it *is* one of the most important positioning performance specifications. However, using this number to compare drives can be somewhat fraught with danger. Alright, that's a bit melodramatic; nobody's going to get hurt or anything. :^) Still, to use seek time properly, we must figure out exactly what it means.

Switching between tracks requires the head actuator to move the head arms physically, which being a mechanical process, takes a specific amount of time. The amount of time required to switch between two tracks depends on the distance between the tracks. However, there is a certain amount of "overhead" involved in track switching, so the relationship is not linear. It does not take double the time to switch from track 1 to track 3 that it does to switch from track 1 to track 2, much as a trip to the drug store 2 miles away does not take double the time of a trip to the grocery store 1 mile away, when you include the overhead of getting into the car, starting it, etc.

Seek time is normally expressed in milliseconds (commonly abbreviated "msec" or "ms"), with average seek times for most modern drives today in a rather tight range of 8 to 10 ms. Of course, in the modern PC, a millisecond is an *enormous* amount of time: your system memory has speed measured in nanoseconds, for example (one million times smaller). A 1 GHz processor can (theoretically) execute over one million instructions in a millisecond! Obviously, even small reductions in seek times can result in improvements in overall system performance, because the rest of the system is often sitting and waiting for the hard disk during this time. It is for this reason that seek time is usually considered one of the most important hard disk performance specifications. Some consider it the most important.

At one point many years ago seek times were difficult to use because manufacturers wouldn't agree on a standardized way of reporting them. Today, this has largely been corrected. While seek time is usually given as a single number, in fact there are three different seek time specifications you should examine for a drive, as they represent the drive's performance when doing different types of seeks:

- **Average:** As discussed, this is meant to represent an average seek time from one random track (cylinder) to any other. This is the most common seek time metric, and is usually 8 to 10 ms, though older drives had much higher numbers, and top-of-the-line SCSI drives are now down to as low as 4 ms!
- **Track-to-Track:** This is the amount of time that is required to seek between adjacent tracks. This is similar in concept (but not exactly the same as) the track switch time and is usually around 1 ms. (Incidentally, getting this figure without at least two significant digits is pretty meaningless; don't accept "1 ms" for an answer, get the number after the decimal point! Otherwise every drive will probably round off to "1 ms".)
- **Full Stroke:** This number is the amount of time to seek the entire width of the disk, from the innermost track to the outermost. This is of course the largest number, typically being in the 15 to 20 ms range. In some ways, combining this number with the average seek time represents the way the drive will behave when it is close to being full.

While I believe that seek time is a very important specification, I have become somewhat cynical in the last few years regarding the amount of attention paid to it. The reason is that there is so little difference between the seek time specs of most comparable drives in any given class or category. For example, almost all IDE/ATA 7200 RPM drives shipping in 2000 have an average seek time specification of 8.0, 8.5 or 9.0 milliseconds. This doesn't leave a lot to work with. However, at the same time, we must realize that of the four components that comprise the drive's access time, if you are comparing two drives of the same class and spindle speed, only seek time will differ much between them. So this small differential may be the only thing to distinguish drives; and small differences are what you are likely to see. (Larger discrepancies though, directly translate into often *very* noticeable differences in performance. A drive with a 5 ms seek time will generally blow the doors off one with a seek time of 8.0 to 9.0 ms in random positioning tasks, which is why these fast drives are preferred for servers and other multi-user environments.)

To really put seek time in proper context, it should be remembered that it is the largest component of access time, which is the composite metric that best represents positioning performance. However, it is only one component, and there is at one that is of at least equal importance (see the discussion of access time for more on seek time's role in overall positioning performance). Also, bear in mind that seek times are *averages* that make certain assumptions of how the disk will be used. For example, file system factors will always have an impact on seek performance in the real world.

A couple of additional caveats on seek times. First, unless you see two numbers, one for read performance and one for write, seek times always refer to reads; see here for more details. Ask for the write numbers if you are interested, or you can approximate by adding 1 ms to the *average* read numbers. Second, watch out for "less than X ms" specifications. Rather bogus, and fortunately not seen as often as in the past, I interpret "less than X ms" as "X ms" and you generally should do so as well--if the true average were under "X-1", they'd say "less than X-1 ms" instead of "less than X ms".

Seek time is almost entirely a function of the design and characteristics of the hard disk's actuator assembly. It is affected slightly by the read/write head design since the size of the heads affects movement speed.

Note: Some manufacturers include settle time as part of their seek time specification. Since settle time is relatively small this doesn't really change the seek time numbers much.

Next: Settle Time

Settle Time

The *settle time* specification (sometimes called *settling time*) refers to the amount of time required, after the actuator has moved the head assembly during a seek, for the heads to stabilize sufficiently for the data to begin to be read. Since it is a component of access time and therefore part of the time required to position for reading a random file, I include it here for completeness. However, since settle time is usually so short (typically less than 0.1 msec) it is dwarfed in importance by seek time and rotational latency, and differences between drives in this regard are not really significant. Some manufacturers do not even bother to specify settle time, and some just lump it in with seek time.

Settle time, like seek time, is a function of the drive's actuator characteristics.

Next: Command Overhead Time

Command Overhead Time

Command overhead refers to the time that elapses from when a command is given to the hard disk until something actually starts happening to fulfill the command. In a way, it's sort of like a "reaction time" for the disk. Consider when you're driving a car and a streetlight suddenly turns red; your "command overhead" is the time that elapses from when the light changes, until your foot starts to move toward the brake pedal. (Or if you live in the greater Boston area, the accelerator. ;^)

Like settle time, command overhead is a component of access time and thus part of the overall equation of random positioning performance. Also like settle time, it is generally very small and not highly variable between drive designs; it is generally around 0.5 ms for pretty much all modern drives and therefore not something that requires a lot of attention. Also like settle time, it is sometimes not even specified separately from seek time but rather "lumped in" with it. It is dominated by seek time and rotational latency in the overall positioning performance picture.

Command overhead is influenced primarily by the design of the disk's integrated controller, and to some extent, the nature of the interface used (which of course is a major influence on the design of the controller!)

Next: Latency

Latency

The hard disk platters are spinning around at high speed, and the spin speed is not synchronized to the process that moves the read/write heads to the correct cylinder on a random access on the hard disk. Therefore, at the time that the heads arrive at the correct cylinder, the actual sector that is needed may be anywhere. After the actuator assembly has completed its seek to the correct track, the drive must wait for the correct sector to come around to where the read/write heads are located. This time is called *latency*. Latency is directly related to the spindle speed of the drive and such is influenced solely by the drive's spindle characteristics. This operation page discussing spindle speeds also contains information relevant to latency.

Conceptually, latency is rather simple to understand; it is also easy to calculate. The faster the disk is spinning, the quicker the correct sector will rotate under the heads, and the lower latency will be. Sometimes the sector will be at just the right spot when the seek is completed, and the latency for that access will be close to zero. Sometimes the needed sector will have just passed the head and in this "worst case", a full rotation will be needed before the sector can be read. On average, latency will be half the time it takes for a full rotation of the disk. This table shows the latency for the most common hard disk spindle speeds:

Spindle Speed (RPM)	**Worst-Case Latency (Full Rotation) (ms)**	**Average Latency (Half Rotation) (ms)**
3,600	16.7	8.3
4,200	14.2	7.1
4,500	13.3	6.7
4,900	12.2	6.1
5,200	11.5	5.8
5,400	11.1	5.6
7,200	8.3	4.2
10,000	6.0	3.0
12,000	5.0	2.5
15,000	4.0	2.0

The "average" value is almost always the one provided as a specification for the drive; sometimes the "worst case" number is also mentioned. Sometimes latency is not even mentioned specifically at all, but it can always be calculated using this formula:

(1 / (SpindleSpeed / 60)) * 0.5 * 1000

Which factors down to this much simpler formula:

30000 / SpindleSpeed

The result is a value in milliseconds.

In looking at the table above, notice that the first increases in spindle speed yielded the greatest percentage improvements in performance. As speeds continue to increase, there are diminishing returns for the extra RPMs. Going from 5,400 RPM to 7,200 RPM shaved 1.4 milliseconds off the average latency, but going from 7,200 to 10,000 (which is a bigger jump in both absolute and percentage terms) only reduces it 1.2 milliseconds. At some point companies will likely "max out" on spindle speeds because there won't be any point in increasing further, especially considering the problems that are created when speeds are increased. The 12,000 speed introduced by the Hitachi Pegasus, while very fast, never really caught on as an industry standard. It looks like 15,000 RPM will be the next standard spindle speed for top-of-the-line SCSI drives. It has yet to be seen what price will be paid for jumping to such a high spindle speed; the improvement in latency over standard 10,000 RPM drives is "only" 1.0 milliseconds.As with seek time, figures in milliseconds are big numbers when dealing with computer system performance, but to shave another 1.0 ms off latency from 15,000 RPM would require going to 30,000 RPM, which would be a very significant engineering challenge probably not justified by shaving 1.0 ms off the total access time for the drive.

Again, as with seek times, latency is most relevant only to certain types of accesses. For multiple, frequent reads of random sectors on the disk, it is an important performance-limiting factor. For reading large continuous blocks of data, latency is a relatively minor factor because it will only happen while waiting to read the first sector of a file. The use of cylinder and head skewing on modern drives is intentionally designed to reduce latency considerations when switching between consecutive heads or cylinders on long sequential reads or writes.

Next: Access Time

Access Time

Access time is the metric that represents the composite of all the other specifications reflecting random performance positioning in the hard disk. As such, it is the best figure for assessing overall positioning performance, and you'd expect it to be the specification most used by hard disk manufacturers and enthusiasts alike. Depending on your level of cynicism then, you will either be very surprised, or not surprised much at all, to learn that it is rarely even discussed. :^) Ironically, in the world of CD-ROMs and other optical storage it *is* the figure that is universally used for comparing positioning speed. I am really not sure why this discrepancy exists.

Perhaps the problem is that access time is really a derived figure, comprised of the other positioning performance specifications. The most common definition is:

Access Time = Command Overhead Time + Seek Time + Settle Time + Latency

Unfortunately, this definition is not universal, and is made complicated by the fact that manufacturers refuse to standardize on even what access time's subcomponents mean. Some companies incorporate settle time into seek time, some don't, for example. And to make matters worse, some companies use the term "access time" to mean "seek time"! They really are not the same thing at all.

In the end though, when you are looking at the ability of a drive to randomly position, access time is the number you want to look at. Since command overhead and settle time are both relatively small and relatively similar between drives, that leaves the sum of seek time and latency as the defining characteristic between drives. Seek time and latency are a result of very different drive performance factors--seek time being primarily a matter of the actuator and latency the spindle motor--resulting in the possibility of some drives being better in one area and worse in another. In practice, high-end drives with faster spindles usually have better seek times as well since these drives are targeted to a performance-sensitive market that wouldn't buy a drive with slow seek time.

Let's compare a high-end, mainstream IDE/ATA drive, the Maxtor DiamondMax Plus 40, to a high-end, mainstream SCSI drive, the IBM Ultrastar 72ZX. (When I say "high end" I mean that the drives are good performers, but neither drive is the fastest in its interface class at the time I write this.) The Maxtor is a 7200 RPM drive with a seek time spec of "< 9.0 ms", which to me means 9 ms. Its sum of its seek time and latency is about 13.2 ms. The IBM is a 10,000 RPM drive with a seek time spec of 5.3 ms. It's sum of seek time and latency is about 8.3 ms. This difference of 5 ms represents an enormous performance difference between these two drives, one that would be readily apparent to any serious user of the two drives.

As you can see, the Cheetah beats the DiamondMax on both scores, seek time and latency. When comparing drives of a given class, say, IDE/ATA 7200 RPM drives, they will all have the same latency, which means, of course that the only number to differentiate them is seek time. Comparing the Maxtor above to say, the Seagate Barracuda ATA II with its 8.2 ms seek time shows a difference of 0.8 ms, or around 10%. But the proper comparison includes the other components of access time. So the theoretical access time of the Maxtor drive is about 13.7 ms (including 0.5 ms for command overhead) and that of the Seagate Barracuda drive 12.9. The difference now is about 6%. Is that significant? Only you can judge, but you also have to remember that even access time is only one portion of the overall performance picture.

Remember that access time is an average figure, comprised of other averages. In fact, access time on any particular read or write can vary greatly. For an illustration, let's consider the IBM 34GXP drive, look at its minimums and maximums, and see how they translate into access time minimums and maximums:

Attribute	Best-Case Figure (ms)	Worst-Case Figure (ms)
Command Overhead	0.5	0.5
Seek Time	2.2	15.5
Settle Time	<0.1	<0.1
Latency	0.0	8.3
Total	2.8	28.4

As you can see, there's quite a range! In the real world these extremes will rarely occur, and over time will be "averaged out" anyway, which is the reason that average figures are used. However, it's important to remember that this wide range can occur on any given access, and random perturbations can affect benchmarks and other performance tests.

Next: Transfer Performance Specifications

Transfer Performance Specifications

Since the obvious objective in using a hard disk is to transfer data to the hard drive and onto the disks, or off the disks and out of the drive, the rate of data transfer is of paramount importance. Traditionally, real transfer rate metrics have been very underrated and given almost no attention compared to positioning specifications like seek time. The only transfer specification that is really commonly mentioned is the speed of the interface, which is actually the *least* important indicator of overall disk performance.

Before we look at transfer specifications, we need to have a short word about terminology. :^) Transfer rates are confusing in part because of the phrase "transfer rate" can mean so many different things. Data transfer occurs in two main steps. For a read, data is first read from the disk platters by the heads and transferred to the drive's internal buffer; then it is moved from the buffer, over the interface, to the rest of the system. For a write, the process is reversed. The rate that transfer occurs within the disk is of course the *internal* transfer rate; the rate that transfer occurs over the interface is the *external* transfer rate. They are usually not the same, and in some cases can differ by an order of magnitude.

Internal transfer rates are further broken down into the media transfer rate and the sustained transfer rate, and further complicating things is the fact that transfer rates are not constant over the surface of the drive. It sounds impossible to get a handle on, but it's not that bad once you place it all in the proper context and perspective, and that's exactly what we will do in this section.

Tip: Whenever you are reading a spec sheet, or discussing transfer rates with someone, be sure to find out exactly *what* transfer rate is being discussed. By itself the term "transfer rate" is meaningless.

Next: Internal Media Transfer Rate

Internal Media Transfer Rate

The *internal media transfer rate* of a drive (often just called the *media transfer rate* or the *media rate*) refers to the actual speed that the drive can read bits from the surface of the platter, or write bits to the surface of the platter. It is normally quoted in units of megabits per second, abbreviated Mbit/sec or Mb/s. Typical values for today's drives are in the hundreds of Mb/s, with a maximum media rate of about 500 Mb/s being high-end at the time of this writing.

Note: Media transfer rates are *not* normally specified in mega*bytes* per second, even though sustained transfer rates *are*; this is not 100% consistent, however. Watch out for this discrepancy when looking at the numbers.

Media transfer rate can be confusing to understand even for the serious hard disk enthusiast; it's equally difficult to describe. :^) For starters, let's explain what it is *not*. It is only related to what is going on inside the hard disk, and therefore has nothing directly to do with the interface transfer rate. It differs from the sustained transfer rate in that it refers *only* to the speed of reading or writing bits to a *single* track of one surface of the disk. Nothing else is included--no positioning, no track or head switching. A track holds a relatively small amount of data--under 0.25 MB with current technology. This means that almost no real-world reads or writes occur on a single track except for very short files, and the performance when reading those is primarily limited by positioning, not transfer. The end result of this is that the media transfer rate does not have much relevance to real-world use of a drive. It is primarily a "theoretical" specification that illustrates the state of the drive's technology. It is used almost exclusively for comparing drives against each other. It is also the basis for the calculation of the sustained transfer rate specification.

Media transfer rates are not constant across the entire surface of a platter. Let's recall for a moment the fact that modern disk drives use zoned bit recording. This is done because the length of the inner tracks on the disk is much shorter than that of the outer tracks. ZBR allows the outer tracks to have more sectors per track than the inner tracks. However, since every track is spinning at the same speed, this means that when reading the outer tracks, the disk is transferring more data per second when when reading the inner tracks. For this reason, the media transfer rate decreases as you move from the outer tracks of the disk to the inner ones.

The explanation above is the reason that there is no single "media transfer rate" figure for a modern hard disk. They are typically stated as a range, from minimum to maximum (with the maximum figure given alone, of course, if only one number is provided). For example, the IBM Deskstar 34GXP (model DPTA-373420) has a media transfer rate of between approximately 171 Mb/s and 284 Mb/s depending where on the disk you are reading: that drive has 12 different zones. This drive has 272 sectors in its innermost zone, and 452 sectors on its outside tracks.

Another important thing to remember about the media transfer rate (and another reason why it is a theoretical measure only) is that it includes *all* bits read or written to the disk, not just user data. As discussed in detail here, some of the data storage space in a sector is reserved for overhead. This means that you cannot assume that the media rate represents the rate at which user data can be read from the disk. Taking the IBM drive above again as an example, its maximum media transfer rate is 284 Mb/s, but the maximum rate that the drive can read user data is about 222 Mb/s in the outside zone.

It's not really feasible to calculate the media transfer rate from other drive specifications, because manufacturers typically do not publish details of their sector format and other pertinent overhead characteristics. The best that you can do is approximate the value by looking at the rate at which user data can be streamed from a given part of the disk. To so do so, we need to know how much data is able to pass under the read/write heads in one second. This is dependent on the density of the data (how tightly packed the data is into each linear inch of disk track), and also how fast the disk is spinning. The density of the data can be calculated easily if we know how many sectors are on the track, since we know how many bytes of user data there are in a sector (512). The speed of the disk is calculated in RPM, so we divide it by 60 to get revolutions per second. This gives us a calculation of the data transfer rate in megabits per second as follows (to get the result in mega*bytes* per second, simply divide by 8):

User Data Transfer Rate = (Spindle Speed / 60 * Sectors Per Track * 512 * 8) / 1,000,000

This formula shows the derivation of the 222 Mb/s figure above: use 7200 for the 34GXP's spindle speed, and 452 sectors on its outside tracks. Note that you need the true physical geometry here; the logical BIOS setup parameters will give incorrect results. (If the geometry you are using says the disk has 63 sectors per track and 16 heads, it's almost certain that you are looking at the logical BIOS geometry!) And again, remember that this is not the same as the media transfer rate; to get that figure you'd have to replace the "512" above with the total number of bits, including overhead, contained in each sectors of the disk.

The media transfer rate of the drive is primarily affected by all of the various data recording and encoding factors, as well as the size of the platters, and the drive's spindle speed. In addition, the drive's controller must be fast enough to be able to handle the fastest rate that the disk can read or write, but manufacturers ensure that this is never an issue by beefing up their controllers where necessary.

Next: Head Switch Time

Head Switch Time

Each cylinder contains a number of tracks, each accessible by one of the heads on the drive (one head per surface). To improve efficiency, the drive will normally use all of the tracks in a cylinder before going to the next cylinder when doing a sequential read or write; this saves the time required to physically move the heads to a new cylinder. Switching between heads is a purely electronic process instead of a mechanical one. However, switching between heads within a cylinder still requires a certain amount of time, called the *head switch time*. This is usually less than the track switch time, and is usually on the order of 1 to 2 milliseconds. (Seems kind of slow for an electronic process, doesn't it? The reason is that this time includes all of the overhead of the switch as well; it is all of the time that passes between when the read stops on one head and when it actually starts again on the next one.)

Head switch time is not commonly discussed, but it is an important component of sustained transfer rate, since STR measures transfer rate over larger reads or writes that encompass more than one track. See the discussion of sustained transfer rate for more details. You may also want to read about head and cylinder skew here.

Head switch time is primarily influenced by the characteristics of the hard disk's controller. It does not vary greatly from drive model to model or between manufacturers.

Tip: Even though it is typically smaller than cylinder switch time, head switch time is more important to transfer performance because head switches occur more often than cylinder switches (unless you are using a single-platter disk). See the discussion of calculating of sustained transfer rate for more on this subject.

Next: Cylinder Switch Time

Cylinder Switch Time

Similar in concept to head switch time, *cylinder switch time* is the time that elapses when the drive finishes reading (or writing) all the data on a given cylinder and needs to switch to the next one. This normally only occurs during fairly long reads or writes, since the drive will read all the tracks in a cylinder before switching cylinders. Cylinder switch time is slower than head switch time because it involves a mechanical process: moving the actuator assembly. It is usually somewhere around 2 to 3 milliseconds.

Note: You might think that cylinder switch time would be the same as track-to-track seek time, after all, it's the same thing, isn't it? They aren't the same however, because cylinder switch time includes all of the overhead time that passes from the time the read stops on one track until it starts again on the next one. This is why cylinder switch times are typically double those of track-to-track seeks.

Cylinder switch time is another specification that is fairly obscure and not commonly discussed, but is an important component of sustained transfer rate, since STR measures transfer rate over larger reads or writes that can encompass more than one cylinder. See the discussion of sustained transfer rate for more details. You may also want to read about head and cylinder skew here.

Cylinder switch time is influenced by the characteristics of the hard disk's controller as well as its actuator mechanics. It does not vary greatly from drive model to model or between manufacturers.

Next: Internal Sustained Transfer Rate (STR)

Internal Sustained Transfer Rate (STR)

The media transfer rate is the maximum rate that any particular track of the hard disk can have bits written to it or read from it. However, most transfers from the hard disk involve more than a single track (and the performance of accesses short enough to fit in a single track is typically dominated by positioning concerns more than transfer issues anyway). For real-world transfers of average files, what we are concerned with is the rate at which the drive can transfer data sequentially from multiple tracks and cylinders on the disk. This specification is the drive's *sustained transfer rate* (sometimes the *sequential transfer rate*), abbreviated *STR*.

Sustained transfer rates are most relevant for reflecting the drive's performance when dealing with largish files. It is based upon the drive's media transfer rate, but includes the overheads required for head switch time and cylinder switch time. Also, STR is normally measured in *bytes*, not *bits* like the media transfer rate, and includes only data, not the overhead portions of each sector or track. An example: let's say we want to read a 4 MB file from a hard disk that has 300 sectors per track in the zone where the file is located; that's about 0.15 MB per track. If the drive has three platters and six surfaces, this means that if this file is stored sequentially, it will on average occupy 26 tracks over some portion of 5 cylinders. Reading this file in its entirety would require (at least) 25 head switches and 4 cylinder switches.

STR can be calculated from various characteristics of a disk, but this isn't nearly as conceptually simple as calculating a media transfer rate on a single track. Rather than just provide a lengthy formula, I'll try to explain how the calculation is done. A transfer rate is of course data transferred per unit of time. So our equation will be a ratio of data transferred to the time taken to transfer it. Now, to represent a sustained transfer we need to cover an entire cylinder, so we include all the head switches while reading the cylinder, and one cylinder switch time as well (to get us to the next cylinder). The data that is transferred for an entire cylinder read is as follows:

Data transferred per cylinder = Number of surfaces * Sectors per track * 512 bytes

where "number of surfaces" is identical to the number of tracks per cylinder, of course. Now, how much time is taken? First, we of course have to wait for the disk to make one complete revolution for each track read, as the data is read. Then we need to add a number of head switches equal to the number of surfaces *less one*, and finally, one cylinder switch. So the time taken to transfer an entire cylinder is as follows:

Time per cylinder transfer = Number of surfaces * Platter revolution time + (Number of surfaces - 1) * Head Switch Time + Cylinder Switch Time

The easiest way to calculate platter revolution is to double the disk's latency specification. The final equation then looks like this:

STR = (Number of surfaces * Sectors per track * 512) / (2 * Number of surfaces * Latency + (Number of surfaces - 1) * Head Switch Time + Cylinder Switch Time)

The result is in bytes per second. Simple, right? ;^) Let's use the same IBM Deskstar 34GXP model that we discussed in the media transfer rate section. This drive has 452 sectors in its outermost zone, and a 7200 RPM spin speed (for latency of 4.17 ms). This family's head switch time is 1.5 ms and cylinder switch time is 2.0 ms. We'll consider the flagship drive that has five platters and hence ten surfaces:

STR = (10 * 452 * 512) / (2 * 10 * 0.00417 + (10 - 1) * 0.0015 + 0.002) = 23,399,798 bytes per second

The specification for maximum STR for this drive is in fact 23.4 MB/s. Out of curiosity, let's do the same calculation for the 13.6 GB version of this drive, which has only two platters:

STR = (4 * 452 * 512) / (2 * 4 * 0.00417 + (4 - 1) * 0.0015 + 0.002) = 23,223,683 bytes per second

The change is due entirely to the difference between head switch time and cylinder switch time: if they were identical the STRs would be as well. Since the drive with more platters performs a higher ratio of (faster) head switches compared to (slower) cylinder switches, its STR is a bit higher. Still, it's only a difference of less than 1% between the biggest and smallest members of the family.

An important question to consider is how meaningful the STR numbers really are: if you have the drive above, will it really let you read at a rate of about 23 MB/second? I'm sure you won't be shocked if I say "no". There are a number of issues involved. First, since STR is derived directly from the media transfer rate, its value also depends on what part of the disk is being read; larger outer cylinders have the highest STR, smaller inner cylinders have the lowest. Second, there's the matter of whether the access is *really* sequential. There is a big difference between a 10 MB file that is laid out contiguously on the disk, and one that is fragmented into a dozen pieces. Once you fragment the file, you aren't doing a consecutive data transfer any more. Each fragment of the file introduces the need for an additional positioning step to the location where the next piece starts, which slows the transfer and introduces other factors into the performance measurement. Finally, real-world transfers incur all sorts of penalties due to operating system overhead and other considerations. A good rule of thumb in the computer world is that you never get the theoretical maximum of *anything*. :^)

STR has in the last few years started to get more attention than it traditionally has--some would say too much. :^) It is important to those who do a lot of work with large files, but not as critical to those who work with a large number of smaller files, which includes many, if not most, Windows users. It is probably best to value it roughly equally with key positioning specifications such as access time.

Sustained transfer rate is affected by just about every internal performance factor you can name. :^) The number of platters influences it by changing the mix of head and cylinder switches; actuator design and controller circuitry affect the switch times; media issues and spindle speed influence the all-important underlying media transfer rates. There's probably no other performance specification that is affected by so many different design factors.

A final point about internal sustained transfer rates vs. external (interface) transfer rates. In order to get the most from the hard disk, the interface *must* be fast enough to be able to handle the maximum STR of the drive. This is usually not a problem because most disk interfaces have sufficient "headroom" to handle drives that run on them. However, many interfaces are also backward-compatible; for example, you can put the drive discussed above on an older IDE/ATA interface running at 16.6 MB/s. It will work, but clearly you will not get STR of 23 MB/s over that interface. The converse is that putting a drive on an interface much faster than it won't *improve* performance much; but that's a topic for another section.

Next: External (Interface) Transfer Rate

External (Interface) Transfer Rate

The internal transfer rate of the drive represents the speed with which bits can be moved to (from) the hard disk platters from (to) the hard disk's integrated controller. The *external* or *interface* transfer rate represents the speed which which those bits are moved between the hard disk and the rest of the PC. This is usually faster than the internal rate because it is a purely electronic operation, which is typically much faster than the mechanical operations involved in accessing the physical disk platters themselves. This is in fact a major reason why modern disks have an internal buffer.

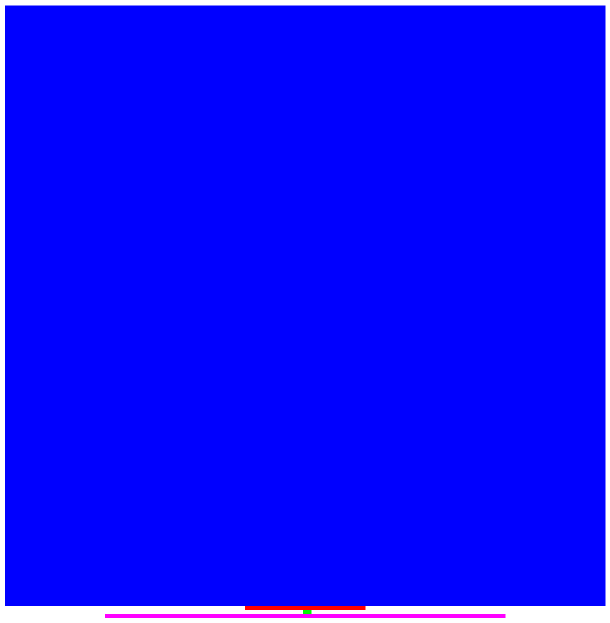
The external transfer rate is unique among hard disk specifications in that it has *almost nothing* to do with the hard disk itself. The integrated controller must have the right chip to support the interface, but that's about it. The external transfer rate is dictated primarily by the type of interface used, and the *mode* that the interface operates in. Support for a given mode has two requirements: the drive itself must support it, and the system--usually meaning the system BIOS and chipset, or add-in controller card--must support it as well. Only one or the other does absolutely no good. External transfer rate is affected by a variety of interface issues, discussed in much more detail in the section on external interface performance factors.

External transfer rate is a perennial candidate for "most overrated hard disk specification". The reason is that external transfer rate specs are usually very high and impressive; manufacturers print them in big bold letters on their retail boxes, and system makers highlight them on their spec sheets. Unfortunately, they usually have very little to do with real-world performance, because the drive's internal characteristics limit transfer performance.

As I've mentioned before, transfer consists of two steps, internal and external. For typical transfers, the net speed of the transfer cannot be any higher than the slower of these two components. Since the external transfer rate of a drive is usually much higher than its internal sustained transfer rate, that means that the STR will be the bottleneck, and thus the factor that limits performance; the high transfer rate of the interface is *mostly wasted*. As an analogy, suppose you have a 1/2" garden hose connected to a 3/4" pipe. The 1/2" segment will be what limits the flow of water; increasing the 3/4" pipe to 1" or even higher won't make a difference in how much water you get at the end of the pipe.

There is one occasion where the external transfer rate does come into play: if the data requested by the system is *already in the disk's internal cache or buffer*. In that case, the data can be sent from the buffer to the rest of the system at the full speed of the interface, whatever that happens to be. Unfortunately, these situations represent such a small percentage of total requests that the net effect of the higher interface speed on overall performance is small. Today's IDE/ATA hard disks are designed to operate with an interface speed of 100 MB/s, but their sustained transfer rates are barely pushing 40 MB/s. This means the 100 MB/s speed only applies for the occasional transfer that does not require actual access to the hard disk platters.

There is one area where the interface speed is very important to pay attention to: you do not want it to be too *low* or performance will suffer. If you take the 3/4" pipe mentioned above and reduce its diameter to 1/4", suddenly *it* becomes the bottleneck, not the 1/2" diameter hose. If the interface does not have enough speed to allow the hard disk to run at its full STR, then performance can be substantially degraded. Since interfaces are relatively inexpensive this is a situation you generally want to avoid: instead, upgrade the interface. This issue occurs only when putting a new, fast drive into a rather old, slow system.



A graphical representation of why interface transfer rates are over-rated. In the
diagram above, which is "drawn" to scale, each pixel represent 500,000 bytes.
The blue box is a 45 GB hard disk (the IBM 75GXP.) The green box (see it?) is the drive's internal 2 MB cache. The red box is the average sustained transfer
rate from the platters to the cache, and the magenta box is the 100 MB/s theoretical
interface transfer rate. As you can see, the cache is dwarfed by the disk, and the
interface transfer rate is limited by the sustained transfer rate. STR is what matters
when streaming data from the big blue box, instead of just the tiny green one.

Hard disk manufacturers always provide lots of "head room" by upping the interface standards in anticipation of advances in sustained transfer rates. In 2000 they moved from Ultra ATA/66, which was already sufficiently fast for modern drives, to the 100 MB/s Ultra ATA/100 interface. This despite there being no IDE/ATA drive available that has an STR of even half that figure. It's good to plan for the future; certainly a motherboard supporting a 100 MB/s interface will give you more "room for expansion". Just don't think it will be noticeably faster than one that "only" supports 66 MB/s, with today's drives. And also don't forget that by the time drives need that throughput, you may be using a different motherboard or PC altogether.

Note: I want to explicitly qualify my statements on this page by saying that they apply primarily to the IDE/ATA interface, as well as *single-disk* environments on SCSI. If you are running many drives on a SCSI channel--such as you would with a SCSI RAID array--the speed of the interface *does* become important very quickly, since the drives share the bandwidth of the interface. See this discussion on RAID bus bandwidth for more on this issue.

Next: Other Performance Specifications

Other Performance Specifications

The important matters of positioning and transfer performance are of course the ones that get most of the attention when considering hard disk performance specifications--and rightly so. However, they are not the only specifications that exist for hard disks. There are a few other specs that are routinely found in hard disk data sheets that are indicative of various performance characteristics of the drive, even if they aren't strictly related to the drive's ability to do random accesses or sequential transfers. I take a look at these in this section.

Next: Internal Cache (Buffer) Size

Internal Cache (Buffer) Size

All modern hard disks have an internal buffer, or cache, that is used as an intermediate repository for data being transferred between the hard disk and the PC. It is described in detail in this operation section. The size of this buffer is usually given as a standard specification on modern drives.

Having *some* cache in a drive is somewhat important to overall performance; the drive will use it to buffer recent requests and to "pre-fetch" data likely to be requested by the system in the future. If this data in the cache is in fact needed, it will be transferred to the system at the drive's external transfer rate--much faster than would be possible if there were no cache. However, the number of requests that fit into this category is relatively small. Increasing the size of the cache even by a substantial percentage doesn't change this very much, because no matter how large the cache, it will always be a very small percentage of the total capacity of the drive. Caches today, despite significant increases in size, are still far less than 0.1% of the size of the disk drives they serve.

As memory prices have fallen into the "dirt cheap" range, drive manufacturers have realized that they can increase the size of their buffers at very little cost. Certainly nothing is *lost* in doing this; extra cache won't hurt performance; but neither does it greatly improve it. As a result, if interface transfer rate is the "reigning champion" of overrated performance specifications, then cache size is probably the "prime contender". :^) Some people seem to think a 2 MiB buffer makes a drive four times as fast as one with a 512 kiB buffer! In fact, you'd be hard pressed to find even a 4% difference between them in most cases, all else being equal. Not surprisingly, both external transfer rate and cache size are overrated for the same reason: they apply to only a small percentage of transfers.

The cache size specification is of course a function of the drive's cache characteristics. Unfortunately, manufacturers rarely talk about any characteristics other than the cache's *size*.

Next: Drive Start-Up Ready Time

Drive Start-Up Ready Time

Hard disks are one of the slowest components in the PC to become ready to use when the power is applied to them. To put this in context, consider that it takes less than a second from the time you hit the power switch until the electronic components are "ready 'n rarin' to go". Hard disks, however, always have their slow mechanical components to deal with. As a result, they can take as many as 10 seconds until the platters "spin up" and the drive is ready. The time from when the power is applied until the drive is ready is called the *drive start-up ready time* or *start-up time*.

The drive start-up time is a parameter that usually is not given too much attention, and for good reason: it has no impact on the performance of the drive. The only time it generally comes into play is when the PC boots up so quickly that it tries to boot the operating system before the drive is ready. Newer drives start up quickly enough that this is usually not a problem, but you may want to check this specification if you suspect you may have an issue. Changing your BIOS settings to slow down the initial boot process may be of assistance as well.

The drive start-up ready time spec is largely a function of the drive's spindle motor, and to a lesser extent, its controller. The size and number of platters also has an impact since more mass will cause the spindle assembly to take longer to spin up to speed, all else being equal. Next: Spin-Up Time

Spin-Up Time

The amount of time that is required for the disk platters to get up to full operational speed from a stationary start is called the drive's *spin-up time*. It is usually specified in seconds.

The importance of this specification depends entirely on how you use your hard disk. If you are the type of user that starts up the computer and then leaves the disks spinning all the time, this figure holds virtually no significance. On the other hand, if you use power management to spin down the disk after a couple of minutes of idle time, then whenever you go to use the disk again after an idle period, you will have to wait for the drive to spin back up to speed again. This spec is more important for notebook PC users than those who use desktops, since power management is more important for notebooks, and is more commonly used. To be honest though, either way, a drive with slow spin-up time really represents more of an inconvenience than a major performance issue. After all, once the drive is actually running in normal use the spin-up time becomes irrelevant.

Spin-up time is of course a component of start-up ready time, since in order to get ready when the drive starts, the platters must spin up. :^) It is a function primarily of the drive's spindle characteristics and the number and size of the platters. Smaller drives tend to have lower spin-up times, in part due to lower mass and in part because they are used more for mobile applications.

Next: Power Consumption

Power Consumption

There are several reasons why power consumption is an area of concern for PC users. The first is that the amount of power needed by the hard disks must be provided for when specifying the power supply (although modern systems with one hard disk don't generally need to worry about this). The second is that the start-up power requirements of hard disks exceed their normal requirements and must be given special consideration in systems with multiple storage drives. The third is that more power consumption, all else being equal, equates to more heat dissipated by the drive. The final one is environmental: the trend is towards systems that use less power just for the sake of using less power!

The power consumption specifications provided for a drive vary from manufacturer to manufacturer. Some provide only a "typical" rating for the drive during average conditions, a start-up peak value for the +12 V voltage, and that's it. Others provide a comprehensive look at the drive's use of both +5 V and +12 V power under various conditions. For example, the table below contains the power consumption specifications for the IBM Deskstar 75GXP, four-and five platter models. Note that unlike most hard disk specifications, power consumption normally is higher for drives with more platters even within the same family--since they have more mass to move, more power is required to turn the platters. Many manufacturers just quote an average for the whole family, but IBM generally doesn't:

Operating Condition		**+5 V draw (Amps, RMS)**	**+12 V draw (Amps, RMS)**	**Power Consumption (W)**
Start-Up	**Peak**	0.81	1.81	--
Random R/W Operation	**Peak**	1.02	2.23	--
	Average	0.41	0.78	11.5
Seek	**Peak**	0.47	2.23	--
	Average	0.27	0.84	11.4
Idle	**Average**	0.24	0.57	8.1
Standby	**Average**	0.26	0.015	1.5
Sleep	**Average**	0.17	0.015	1.0

Examining these numbers reveals a number of facts about how the drive uses power. First, notice that when operating (platters spinning and actuator moving), the +12 V draw is about 0.8 A; when idle (platters spinning but actuator stationary), it is about 0.6 A; and when in standby (platters stationary), +12 V is about zero. This tells you that roughly 3/4 of the +12 V power is taken by the spindle motor and roughly 1/4 by the actuator assembly. +5 V is primarily used to drive the controller's components, which is why even in standby mode a fair percentage of the +5 V power required during operation.is needed. This is typical of modern drives. "Real-world" power consumption will generally be close to what the manufacturer specifies, but bear in mind that actual consumption will depend on a number of factors, most especially the manner in which the drive is used.

Power consumption is primarily affected by the design of the drive's spindle motor and the number and size of the spindle platters, and to a lesser extent, other components such as the actuator and controller board.

Next: Hard Disk Internal Performance Factors

Hard Disk Internal Performance Factors

There are a number of design factors and issues that affect the performance--and hence the performance specifications--of the hard disk. Of these, I refer to performance considerations that relate only or primarily to the capabilities of the hard disk drive itself as *internal performance factors*. In theory, these are not directly related to the interface or other parts of the system external to the hard disk, which means they should be reasonably consistent and even "portable" from one system to another. These are really the basis of hard disk performance, since they dictate the theoretical maximums; external factors can only further constrain the limits imposed by the design of the hard disk itself.

This section takes a look at the most important internal performance factors of the modern hard disk. They are divided into three sections, reflecting the different major areas of concern regarding internal performance considerations. First, I take a look at three major design factors related to the mechanics of the drive, which are probably the most important influence on performance. Then, I'll discuss issues related to how data is recorded on the platters. Finally, I'll describe some factors that relate to the drive's integrated controller. For each factor, I will provide a reference to the performance specifications it most directly impacts.

Note: In order to avoid duplication, I do not get into describing in detail how the hard disk's various components work here; I instead refer to the Construction and Operation section where appropriate.

Next: Mechanical Design Factors

Mechanical Design Factors

The defining performance-limiting characteristic of hard disks compared to the other main "performance" components is the fact that they operate mechanically. Consider the other components of the PC that have a fundamental impact on overall system performance: processors, motherboards, system memory, video cards. They are all solid-state--no moving parts--and as such much faster than the mechanical parts that make up hard disks. That's why memory access times are in nanoseconds, for example, while hard disk access times are in milliseconds--a million times slower!

Since mechanical parts are largely what limit hard disk performance, that makes them the most important factors affecting performance. This section takes a look at three of these key mechanical design factors in more detail.

Next: Size and Number of Platter Surfaces

Size and Number of Platter Surfaces

The data in the hard disk is stored on the platter surfaces. (The operation and characteristics of the platters and media are described in detail here, including a lot of performance-relevant detail.) The number of platters and the size of the platters themselves vary between different hard disk designs, and have an important impact on performance in several ways.

First, let's look at platter size. As discussed in much detail here, the trend is towards smaller and smaller platter sizes for a number of reasons; two of them being particularly important to performance. The first one is that smaller platters allow the data on the drive to be located physically closer together, so there is less distance for the hard disk actuator to have to move when doing random reads or writes on the disk. This directly improves positioning performance on random accesses. The second is that smaller platters have lower mass and higher rigidity, which enables them to be spun at higher speeds for a given power of spindle motor (or conversely, to use a lower-powered spindle motor for the same spin speed). The main cost of using smaller platters is reduced capacity, but with areal density constantly increasing--thus doubling capacity per square inch every year or two anyway--this is a trade-off more people than ever are willing to make.

The number of platters has a more subtle influence on performance; this is why you will sometimes see small differences in the specifications of drives of different capacity in the same model family. The first impact is a relatively simple: more platters means more weight and thus more for the spindle motor to turn. This generally means that the spin-up speed and power consumption of a drive with four platters will be a little higher than those figures for the same drive with two platters.

The other impact of the number of platters is a bit more controversial: not everyone agrees on the extent to which these effects exist. All else being equal, a drive with more platters will have slightly better positioning performance *and* a slightly higher sustained transfer rate than one with fewer platters. If you double the number of data storage surfaces, you can store the same amount of data in (roughly) half as many cylinders; this keeps the data "closer together" physically on the drive, reducing the extent to which the actuator must move when doing seeks. You also replace many cylinder switches with head switches when you have more platters; a one-platter drive will have a 1:1 ratio of head switches to cylinder switches on a sustained read; a four-platter drive will have a 7:1 ratio. Head switches are faster than cylinder switches, so this slightly improves STR, though it's certainly not a large effect. I show the difference between drives of the same family in the discussion of the sustained transfer rate specification.

The size and number of platter surfaces on the drive have an impact on seek time (and hence access time), media transfer rate and sustained transfer rate, spin-up speed, and power consumption. Of course, the basic design of the drive also matches the platter size and number to the power of the spindle motor.

Next: Actuator Characteristics

Actuator Characteristics

The hard disk's actuator--or more correctly, its actuator assembly, comprising the actuator, head arms, head sliders and read/write heads--is one of the most important performance-limiting components in the hard disk. It's also one of the least-discussed and least-understood; I discuss it in detail in this operation section.

The primary impact that the actuator assembly has is on positioning performance. Since random accesses require the heads to move over the surface of the disk to the correct location, and the actuator controls this process, the actuator assembly is the primary influence on the drive's seek time. Seek time in turn is the largest component of access time. In order to improve performance, manufacturers are constantly striving to reduce seek times by improving the speed of the actuator assembly. The first step taken was to move to voice-coil actuators from stepper-motor designs. Today, improvement is evolutionary, done by increasing the strength of the actuator itself, reducing the weight and size of the actuator arms and sliders, and tweaking other design parameters.

The characteristics of the actuator also have an impact, albeit relatively minor, on *transfer* performance. The reason is that the actuator has an impact on cylinder switch time, which is a component of sustained transfer rate. Again, the impact is relatively small compared to the impact of the other factors that influence STR.

Next: Spindle Motor Speed and Power

Spindle Motor Speed and Power

The spindle motor is one of the most important components in the hard disk, because its quality and power have a direct impact on many key performance and reliability concerns. It is discussed in detail in this section.

The drive's spindle speed affects both positioning and transfer performance and is thus one of the most important directly-quoted performance specifications unto itself; it is described in its own specification section. It affects positioning performance because it directly correlates to the latency of the drive (and in fact, is the only factor that affects the latency of regular hard disk drives). Latency, in turn, is an important component of access time, the specification that best correlates to overall positioning performance. Spindle speed affects transfer position because it is related to the drive's media transfer rate, which is the prime transfer performance specification.

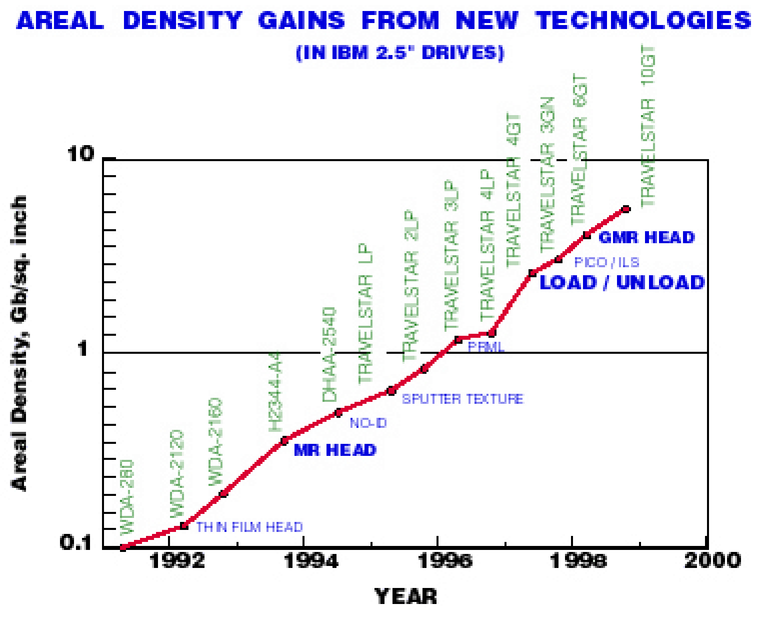
Note: While spindle speed affects the media transfer rate, it is not *proportional* to it. The reason is that it is more difficult to read and write at very high linear densities when running at very high spindle speeds. This means that in some cases a drive running at 5400 RPM will have a higher areal density than a similar drive running at 7200 RPM. The transfer rate is still normally higher for the 7200 RPM drive, because the spindle speed is 33% higher and the linear areal density is usually only smaller by a factor of 10% or less (though this could change at any time; who knows what those engineers are up to! :^))

The *power* of the spindle motor has an impact on the drive's spin-up speed, for obvious reasons. Also, since the spindle motor is the primary consumer of power in the hard disk, its design has the biggest impact on overall power consumption. In some ways, slower drives have an advantage here; it takes longer to spin anything up to 10,000 RPM than to 5400 RPM (unless you use a correspondingly larger motor in the faster drive.)

Next: Data Recording and Encoding Factors

Data Recording and Encoding Factors

One of the most amazing sets of technologies in the world is the combination of various construction and geometry characteristics of the hard disk, which allow data to be stored at such incredible capacities and retrieved with such speed and precision. The end results of the advances of recent years are capacities approaching 100 GB and incredible read/write speed, attributes that seemed unattainable in such a short period of time a decade ago. And there's nowhere to go but up from here!



This chart shows the progress of areal density over the last several years, as refleced in IBM's Travelstar series of 2.5" form factor hard disk drives. Increased areal density leads directly to larger capacity and better performance. Note that the density scale is logarithmic!

Original image © IBM Corporation
Image used with permission.

In this section I will discuss some of the key influences on performance that are related to the way that data is stored on the hard disk. This includes a look at the way data is written to the disk medium, and also how it is organized on the disk.

Next: Recording Density

Recording Density

Recording density refers to the number of bits that can be placed along a given length of track on a hard disk. Measured in bits per inch, and also sometimes called *bit density*, *linear density* or *linear areal density*--the last term being an oxymoron of sorts!--it is one of the two components of areal density. You can read about it in more detail here.

Recording density is an important performance factor because it is related to both of the "positioning plus transfer" specifications: areal density and spindle speed. Obviously, since it is a *component* of areal density, it is directly proportional to areal density. As such, it directly or indirectly affects both the access time of the drive and its sustained transfer rate, sometimes in more than one way. How does it influence spindle speed? Well, as explained here, increases in recording density can limit the ability of the drive's spindle motor to be increased--or vice-versa, a faster spindle may requires the recording density to be reduced on the drive to ensure reliability.Recording density itself is increased primarily through advances in media materials, improved platter substrates, and improved read/write head technologies.

Next: Read/Write Head Technology

Read/Write Head Technology

The read/write heads actually write and read the data to and from the hard disk, so you'd think they'd have at least *some* impact on the drive's overall performance! And in fact this is true: they do. Improvements to read/write head technologies, and related components such as the head sliders, are key to allowing increases in linear density, which in turn affects areal density and hence both positioning and transfer performance. They also allow further miniaturization of the head sliders and related components, which indirectly allows faster and more accurate positioning performance.

However, you rarely hear read/write heads mentioned in discussions of hard disk performance. The reason it isn't often discussed is not because it isn't important, but rather because it doesn't change very frequently. There have been only five general read/write head technologies used in the last 30 years! So in some ways, nobody talks about the heads because there isn't a lot to talk about. :^) Most manufacturers make the change to new head technologies at roughly the same time. Further, many of the manufacturers license head technology from each manufacturers. Today virtually every disk drive sold uses GMR heads; therefore, this is basically assumed and not really a topic for discussion. This equality in basic design generally leaves only minor tweaks to differentiate head designs between models.

Next: Encoding Method

Encoding Method

The *encoding method* of the disk refers primarily to the algorithm(s) used to change user data into a pattern of bits that can be stored on the platters. This isn't nearly as simple as it sounds; therefore, several different but related methods have been developed over time to facilitate this conversion (and of course, subsequent conversion in the other direction when the data is read back from the disk.)

Improved encoding methods have one main impact on hard disk performance: they increase recording density, and thus areal density. This improves positioning and transfer performance as described in those sections. Encoding methods are discussed in detail in this section; it contains a relevant comparison between the different methods used over the years, showing how data density has been increased as they have been improved.

Like hard disk read/write heads, encoding method is rarely discussed in hard disk performance circles because it doesn't change very often, and also because it doesn't vary greatly from one drive to the next. Mostly, its effects are bound up in the areal density specification, as well as the transfer rate specifications.

Next: Track and Sector Layout

Track and Sector Layout

There are several effects on performance that come about as a result of how the data on the surface of the platter is organized. In order to make sense of a platter surface that can store 10 GB of data or more, there has to be a way of organizing it into smaller, more manageable pieces. To accomplish this end, each surface is split into tracks, and then each track is further split into sectors, each sector holding 512 bytes of user data (normally). The track and sector layout of the hard disk, and some of the issues in how this organizing is done, are discussed here.

The most important impact of the track and sector layout is on sustained transfer rate. The various techniques used in improving the way data is organized on the hard disk all are primarily oriented around increasing the amount of data that can be stored in a given amount of space, and this mainly improves sustained transfer rate. Some advances also improve positioning speed. Here's a quick list of the performance enhancements that fall into this general category:

- **Optimal Interleaving:** All modern drives use optimal (1:1) interleaving, which cannot be changed. Thus, this factor is not really relevant for distinguishing modern drives. However, you should understand that proper interleaving does ensure that the maximum possible transfer rate is realized on a drive. Older drives that were sometimes set to the wrong interleave factor would have greatly reduced transfer rates compared to their potential maximums.

- **Zoned Bit Recording:** The use of zoned bit recording has allowed the larger outer tracks of a hard disk to be used to their full potential. It is also the reason that the media transfer rate of a disk depends on what part of the disk is being accessed; since outer tracks have more sectors, they have a higher transfer rate.
- **Cylinder and Head Skew:** Optimal cylinder and head skew factors built into the drive controller are necessary for high sustained transfer rates; they optimize cylinder switch time and head switch time respectively.
- **Sector Format:** Improved sector formats and higher sector format efficiency allow a larger percentage of a track to contain data, increasing capacity and transfer rates. Also, the "no ID" sector format improves random positioning by saving time during random seeks, and avoiding "detours" around remapped sectors.

Next: Controller and Cache Factors

Controller and Cache Factors

There are a few performance factors that are related to the hard disk's integrated controller, and its internal buffer. Since these parts of the hard disk aren't typically the most important in terms of their influence on overall performance, this discussion is relatively brief. However, these components do have an impact on performance. I also want to point out that some of the aspects of how the cache functions are more important than just its *size*, which it sometimes seems is all that most people focus on today...

Next: Controller Circuitry and Algorithms

Controller Circuitry and Algorithms

All modern drives include an *integrated controller*, also called its *logic board*, that functions as its "nerve center"; it runs the drive, controlling everything that happens within it. It also interfaces the drive to the rest of the PC system. The controller is discussed in this operation section.

The controller has an important but basically non-measurable impact on the overall performance of the hard disk. In some ways, I liken the controller of the hard disk to the umpires of a baseball game. One way of assessing how good a job the umpires are doing is as follows: if you don't notice that they are there, they are doing a good job; if you do notice them then maybe not. Similarly, the controller's job is to make sure everything happens as it should without drawing undue attention to itself.

The controller's speed in some ways affects the performance of everything in the drive simply because it *manages* everything in the drive. However, since it is much faster than the drive's mechanical components, its speed is shadowed by that of the slower components. The controller's design doesn't usually *improve* performance much, but if it is not designed to meet the needs of the other parts of the system, it can in theory harm performance. For example, when you are doing a sustained read from the drive, the controller cannot make the platters spin faster or affect the disk's recording density. It has no way to "speed up" the transfer, but if it is not designed to have enough capacity to handle the disk's maximum throughput, it can *limit* performance. Obviously, hard disk manufacturers make sure this doesn't happen.

Special algorithms within the controller, such as those used to manage multiple commands, *can* have a direct impact on performance under some types of usage.

Next: Cache Size and Type

Cache Size and Type

All modern hard disks have an internal cache, that is used as a buffer between the fast PC system and the slow mechanics of the hard disk. It is discussed in some detail here.

The *size* of the cache is a commonly-quoted (and often overhyped) performance specification. It of course has some impact on overall performance, but the exact size of the cache is not nearly as important as many people have been led to believe. The impact of the cache is on burst transfers of data over the external interface.

The *type* of the cache is rarely mentioned in specification sheets, but its speed has some small impact on performance as well. Since reads and writes depending on the cache are themselves small, differences in cache technologies don't make much real-world performance difference at all. Most disks use for cache whatever current memory technology is mainstream and inexpensive; today, that's SDRAM.

Next: Write Caching Policy

Write Caching Policy

There's a complication involved in caching *write* requests to the disk that doesn't exist when doing reads: should the write be cached (put into the disk's buffer) or forced to go through directly to the disk platters? If you don't cache the write you effectively remove the buffering that is in place to isolate the system from the slow mechanics of the drive. If you *do* cache the write you improve performance, but what happens to the write if the power is cut off to the disk before it can be put out to the platters? This is not a simple question, and doesn't have a simple answer; see here for a full discussion on the subject.

Most people won't worry about the potential risks of write caching, especially when they find out that it improves performance. :^) That's fine; but you should try to at least find your drive manufacturer's "write policy policy" when shopping for a drive; it's a good thing to know. A lot of drives use write caching without explicitly saying so.

It should be noted that write caching improves performance pretty much only in a random write environment: writing small blocks to the disk. The cache size is small relative to the size of the disk, so write caching won't improve performance much on a long sequential write. (On a long sustained write the buffer will fill up, and thus force writes to the platters to occur in order to provide space in the cache for later blocks in the sequence.)

Next: Thermal Recalibration

Thermal Recalibration

Thermal recalibration is a procedure that was at one time commonly employed to correct for shifts in the positions of tracks on drive surfaces as they heated up. It is discussed in this section.

On some older drives, thermal recalibration caused performance problems due to unexpected interruptions for the recalibration to be performed. These primarily affected transfer performance, particularly for users manipulating large files. To combat this problem, manufacturers created special (expensive) drives that did not use thermal recalibration. Fortunately today, this is no longer an issue, since recalibration is not required with today's drives the way it once was.

The most important thing to remember about thermal recalibration today in terms of performance is: don't pay extra for a drive with a big label on it that says "no thermal recalibration!" :^)

Next: Hard Disk External Performance Factors

Hard Disk External Performance Factors

Performance factors that are solely a function of the capabilities of the hard disk drive itself are the internal performance factors. However, there are also *external performance factors* to be considered, which relate to how the hard disk relates to other components in the system. Everything from the file system used on the disk, to the disk's interface speed, to the performance of the CPU and other key components, can affect disk performance.

This section discusses external factors that affect the performance of a hard disk in the "real world" of a system. Note that many of these factors are so external that they don't relate directly to the drive itself, but rather the rest of your system. As such, they often don't correspond directly to any of the hard disk performance specifications. Rather, they influence the real performance you will see when using the drive on a daily basis.

Since external factors can be different for the same drive in two different systems, this is a good reason to be careful about assuming that a given performance level on one PC will translate directly to another.

Next: Disk Interface Factors

Disk Interface Factors

While overrated in many ways, the speed of the interface *is* important in the grand scheme of hard disk performance. The interface can form a bottleneck to overall performance if it is too low for the hard disk's maximum sustained transfer rate, and can also have other effects on real-world performance. The issues discussed in this section of course primarily affect the disk's interface speed performance specification.

The two most popular hard disk interfaces used today, by far, are IDE/ATA and SCSI (and enhancements of each), so those are the ones that I focus on in this discussion. There is an entire large section devoted discussing interface issues in general, so I won't delve into too much on that subject here. I will focus primarily on how the interface affects performance, but you will still find more information on interface performance effects in the interface section. Also see the comparison of SCSI and IDE for more issues of relevance to interface performance.

Next: Interface Type, Mode and Speed

Interface Type, Mode and Speed

The nature of the interface between the hard disk and the rest of the PC system plays a role in its overall performance. The type of interface is the sole determinant of the interface transfer performance specification, and the way the interface is actually implemented has an impact on the real-world performance of the storage subsystem.

The interface *type* of almost all drives today is either IDE/ATA or SCSI. Clearly, the decision of which to use involves a much "bigger picture" analysis of your needs and overall system design, since you can't interchange the drives. (I compare the two interfaces here.) However, even within each of the two types there are various modes that control how fast the interface runs. Choosing the best mode for your drive helps to ensure optimal performance. I discuss IDE/ATA modes here, and SCSI modes in this section.

While new hard disks are generally designed to be able to run at the fastest interface speeds possible for their interface, actually obtaining this interface speed requires an appropriate controller, hardware and/or drivers on the system side. Drives will generally "fall back" to slower interface speeds if required. For example, all new IDE/ATA drives are designed to run in Ultra DMA mode 5, allowing a theoretical maximum transfer rate of 100 MB/s. These drives will function on older interfaces at 66 MB/s, 33 MB/s or 16.7 MB/s if required. Performance will be negatively affected, but only significantly eroded if the speed of the interface falls below the maximum sustained transfer rate specification of the drive.

High interface speeds also require appropriate support from the system bus upon which the interface runs.

Next: CPU Utilization

CPU Utilization

Whenever a hard disk is transferring data over the interface to the rest of the system, it uses some of the system's resources. One of the more critical of these resources is how much CPU time is required for the transfer. This is called the *CPU utilization* of the transfer. CPU utilization is important because the higher the percentage of the CPU used by the data transfer, the less power the CPU can devote to other tasks. When multitasking, too high a CPU utilization can cause slowdowns in other tasks when doing large data transfers. Of course, if you are *only* doing a large file copy or similar disk access, then CPU utilization is less important.

CPU utilization is usually highest when running an IDE/ATA hard disk using a programmed I/O mode, and lower when using a DMA mode. Most newer systems use DMA modes for hard disk data transfer, provided that the appropriate hardware and drivers are installed. SCSI drives use a dedicated controller (host adapter) and typically also have low CPU utilization.

CPU utilization is one of those performance factors that is both grossly underrated and overrated at the same time. :^) Most people have never even heard of it; it often seems though that a big percentage of those who do understand its role worry about it *way* too much. :^) Like most performance issues, sweating small differences in numbers is usually pointless; it doesn't matter much if your CPU utilization is 5% or 10%; but if it is 80% or 90% then you are going to see an impact on the usability of the system if you multitask.

Another key issue is that faster drives transfer more data, and more data--all else being equal--requires more processing time. It's totally invalid to compare the CPU utilization of drives of different generations without correcting for this very important consideration.

One hard disk utility commonly employed for testing CPU utilization is *HD Tach*. (Note that this should be considered as information, not an endorsement!)

Next: Command Overhead and Multiple Device Considerations

Command Overhead and Multiple Device Considerations

As discussed in this section, a certain amount of overhead is required to process any command to the hard disk. However, that's only *one* type of overhead, the kind within the hard disk involved in doing a random access to the platters. There are other overhead considerations as well that exist within the system itself. These include the time for the system to process the command at a high level, operating system overhead, and so on. Every "piece" of this overhead reduces overall performance by a small amount.

In comparing the SCSI and IDE/ATA interfaces, command overhead is an important consideration. SCSI is a much more intelligent and capable interface, but it is also more complex, which means more work must be done to set up a transfer. This means that SCSI can be slower than IDE/ATA in a single-user, single-tasking environment, even though it can be much faster and more capable in a machine that is supporting multiple users or multiple devices on the same bus. SCSI shines when you need to use multiple devices on a single bus, where IDE/ATA starts to become cumbersome. See here for more on the eternal IDE vs. SCSI question.

There is also another consideration: the number of devices that are sharing the interface. This is particularly a concern with SCSI, which allows for many devices on a bus (IDE/ATA and enhancements allow just two per channel). If you are using four hard disks on a SCSI bus in a server that is handling many simultaneous requests, and each drive has an internal sustained transfer rate of 18 MB/s, that 80 MB/s for Ultra2 Wide SCSI will probably, at many points in time, be in full use. On an IDE/ATA machine only one device can use any given channel at a time, so you only need to compare the speed of the interface to the speed of each drive that will use it, not the sum of their transfer rates.

Next: PC System Factors

PC System Factors

As I probably reiterate in too many places on this site, the components in a PC system are interrelated, and affect each other's performance in many ways. This makes it difficult to measure the performance of any component in isolation. Some tests are better able than others to isolate the component being tested, but it also depends on the component. Hard disks are virtually impossible to completely isolate from the rest of the system, because every access of the hard disk involves a transfer through the main processing subsystems, and thus involves almost all of the main components of the PC.

Various parts of the PC not only affect the way a hard disk benchmarks; they also affect the real-world usability of the storage subsystem as a whole. In this section I take a look at some key issues related to the PC system as a whole and how they influence hard disk performance.

Next: CPU Speed

CPU Speed

Since the system processor (CPU) is involved in everything that happens in the PC, it of course has an impact on hard disk performance. However, this impact isn't nearly as great as you might think. The reason is that the CPU is so much faster than anything related to the hard disk, particularly today, that it normally spends a great deal of time waiting for the hard disk. There's virtually no way for the CPU to really affect the performance of the storage subsystem all that much if the hard disk and system are properly set up. On older systems though, the performance of the CPU was sometimes a real factor in hard disk performance. And even today, if an interface mode is used that results in high CPU utilization by the hard disk interface, overall performance can become somewhat dependent on the performance level of the CPU itself.

The real issues with CPUs and hard disks are related to benchmarks. Every benchmark that you run on your hard disk involves instructions that are processed on the main system CPU. A faster CPU will run these benchmarks faster than a slower one, and I have found it to be quite consistent that testing the same drive on a machine with a much faster CPU, will result in higher scores. This is accentuated when using one of the transfer modes that requires intervention from the processor, such as programmed I/O.

Next: Interface Bus Speed

Interface Bus Speed

Every hard disk read or write involves a sequence of data transfers. Looking at a read: first the data is retrieved from the hard disk platters. It is transferred to the drive's internal cache by its controller. Then the data is sent over the interface cable to the interface controller on the PC system. *That* controller resides on a system bus, and uses the system bus to communicate the data to the CPU and the rest of the PC. System buses are discussed in detail in their own section.

Normally the speed of the bus used for the hard disk interface is not something that you really need to be concerned with. Virtually all systems today use the PCI bus for interfacing to their hard disks and other storage devices, which is fast enough to handle even the high interface transfer rates of modern drives. (Even if your IDE cables plug into the motherboard directly, they are still going to an IDE controller chip that logically "resides" on the PCI bus.) However, as the interface transfer rate of IDE/ATA drives (maximum 100 MB/s) now approaches the limits of the PCI bus (about 127 MB/s), at some point this will become an issue even on new systems; probably within two or three years. Hard disks continue to get faster and faster.

On older systems interface bus speed limits can become a real issue. There are still systems around that use ISA bus hard disk controllers, for example. Even if you could get one of these older cards to work with a large, modern drive, the slow speed of the ISA bus would drag it down. ISA is limited to a maximum bus bandwidth of under 16 MB/s, easily exceeded even for *sustained* transfers by most any modern drive, not to mention burst transfers!

Even on new systems, alternative means of interfacing hard disks can have a major impact on performance. The widespread adoption of the universal serial bus (USB) standard has been a boon for portability of devices and easy interchanging of hardware. Some companies are now even offering USB-based hard disks. These are convenient, but the slow speed of the USB interface--which was designed for slow items like scanners and keyboards, not hard disks--effectively cripples these drives, limiting them to a maximum transfer rate of about 1 MB/s. That may be OK for moving data between PCs, archiving seldom-needed data or doing smallish backups, but it's *very* slow for just about anything else!

Next: System BIOS Issues

System BIOS Issues

The system BIOS is the set of core routines that provides the primary interface between the main hardware of the system and the software that runs upon it. It plays a critical role in the functioning of any PC; see here for a full section covering it.

The BIOS affects hard disk performance in two distinct ways. The first is that the BIOS itself was traditionally used for access to hard disks, and thus the BIOS's routines had an impact on overall performance. Most of today's operating systems now "bypass" the BIOS to access the hard disks directly, reducing this influence greatly. See this section for a discussion of how the BIOS relates to the hard disk in a general way.

The second is related to the way most systems are designed. In a typical "regular" PC, the motherboard contains an integrated IDE/ATA system controller. Since it is part of the motherboard, it is configured and controlled using code in the system BIOS. This means that the BIOS must provide support for increased capacity when larger drives come out--to avoid BIOS capacity barrier problems--and also support for performance-enhancing features like higher-speed transfer modes, block mode, etc. If your system is a few years old, its BIOS may need to be updated or you could find the performance of newer hard drives restricted.

Next: Operating System and Controller Disk Caching

Operating System and Controller Disk Caching

The process of caching describes the use of buffers to separate operations that differ significantly in speed, so the fast one is not held up by the slower one (or at least, not as much). See here for more general details on caching and how it works. In a system there are many levels of caching that are used to allow different-speed components to run unimpeded; in the disk subsystem there are usually two levels.

The disk drive's logic board contains an integral cache. This cache is used to separate the internal mechanical read/write operations from transfers over the bus, and to hold recently accessed data. A larger cache will result in improved performance by cutting down on the required number of physical seeks and transfers on the platters themselves. Smarter caching algorithms can have the same effect.

In addition to this hardware caching, most operating systems use software *disk caching*. Since the system memory is many orders of magnitude faster than the hard disk, a small area of system memory (usually a few megabytes) is set aside to buffer requests to the hard disk. When the disk is read, the data are stored in this cache in case they are needed again in the near future (which they often are). If they are needed again, they can be supplied from the cache memory instead of requiring another read of the hard disk.

As with the disk's internal buffer, increasing the size of the cache improves performance--to a point. If you increase it too much, your operating system will run out of usable memory for programs and data, and the system will be forced to rely on much slower virtual memory. In this case your use of memory as virtual disk is causing the system to also need to use your disk as virtual memory, defeating your original intent!

In addition to the two typical caches in the hard disk subsystem, some SCSI host adapters add a third level of cache on the controller itself; these are sometimes called *caching controllers*. This cache can be several megabytes in size and logically sits between any system disk cache and any buffer on the hard disk. Again, this improves performance by reducing the number of accesses required to the disk. In this case when the system tries to read data from the hard disk, the controller will intercept the request and if it is in the cache, satisfy it from there instead of going to the hard disk. This both improves system speed greatly and also cuts down on traffic on the SCSI bus.

Next: Redundant Arrays of Inexpensive Disks (RAID)

Redundant Arrays of Inexpensive Disks (RAID)

Many higher-end systems, especially servers, now employ a technology called *redundant arrays of inexpensive disks*, or *RAID*. This concept allows for great improvements in both reliability and performance. The idea is to store data on multiple disk drives running in parallel. The primary motivation in many cases is reliability. From a performance standpoint, most RAID levels improve performance by allowing multiple accesses to happen simultaneously, and also by using algorithms that reduce seek time and latency by taking advantage of having multiple drives at their disposal. The exact performance impact depends entirely on the level of RAID used; some improve read performance at the expense of write performance, for example.

I have written an entire section on RAID that covers its issues, levels and implementation in some detail, including a discussion of its impact on storage subsystem performance. Once used almost exclusively in a corporate setting for large, expensive machines, a new crop of inexpensive RAID controllers and hard disks is bringing RAID into the "mainstream", and small RAID arrays are now commonly seen in the machines of "power users".

Next: File System Factors

File System Factors

There are several factors related to how the disk is logically structured and the file system set up and maintained, that can have a tangible effect on performance. These are basically independent of the hard disk and will have similar impacts on any hard disk. They serve almost solely to influence the real-world performance of the hard disk. Better decisions about how you manage your hard disk's file system can translate directly into better performance on *any* PC.

See the section on hard disk file systems for much more information on these issues.

Next: File System Type

File System Type

The file system refers to the structures that are used to organize data at a high level on the disk; the file system is used by the operating system to store files, directories and other relevant information on the disk for later retrieval. As such, the file system is highly operating-system-dependent. In most cases you don't generally have a "choice" between different file system types. However, in some operating systems you do, and there can be a performance impact from the choice.

Some file systems store files in packages as small as 512 bytes, while others store files in larger chunks called *allocation units* or *clusters*. Some are very simple file systems with few features and little overhead (such as the FAT file system used in DOS and Windows 9x), and others have many features but comparatively higher overhead (NTFS used in NT). Windows NT and 2000 typically give you your choice of file system; Windows 2000 supports FAT16, FAT32 and NTFS. See here for more on the different file systems used in the PC.

Which file system you use can have an effect on overall performance, but it is relatively small: typically a few percentage points. It's also difficult to predict exactly what the effect will be for a given system when selecting from one file system to another. Since the file system affects so many other usability factors of the PC, performance is usually *not* one of the primary factors for deciding between them. As an example, consider the "FAT vs. NTFS" decision, which is probably the most common "file system decision" in the PC world today. These two file systems are so different in so many ways that most people choose one or the other for reasons particular to their use, not performance. If you need the high security and advanced management features of NTFS, you are probably going to use NTFS even if FAT is a few percentage points "faster". Similarly, if you need the compatibility and simplicity of FAT, changing to NTFS for a few ticks on a benchmark is probably unwise.

One file system choice that *is* commonly made in part for performance reasons is "FAT16 vs. FAT32"; this is really a "sub-file-system" choice, since FAT16 and FAT32 are really two flavors of the same file system. The primary performance impact of changing between these has nothing to do with anything inherently different between FAT16 or FAT32, but rather the difference in cluster size that results from the choice. See here for more details on this.

Next: Partitioning and Volume Position

Partitioning and Volume Position

Partitioning is the process of dividing the hard disk into subsections, called *volumes*. It is an important initial step in preparing a hard disk for use, and is discussed in detail on this page.

The choice of how the hard disk is partitioned can have a *tangible* impact on real-world performance. This is due to several different but related effects that you should keep in mind when deciding how to partition your drive:

- **Cluster Size:** The way that the hard disk is partitioned in most cases determines the cluster size of the partition, which has a performance impact. See this section for details.
- **Zone Effects:** Modern hard disks use zoned bit recording to allow more data to be stored on the outer tracks of the hard disk than the inner ones. This directly impacts the media transfer rate of the disk when reading one zone of the disk as opposed to another; see here for details. Hard disks fill their space starting from the outer tracks and working inward. This means that if you split a hard disk into three partitions of equal size, the first partition will have the highest transfer rate, the second will be lower, and the third lower still. Therefore, you can put the more important files on the faster partitions if transfer performance is important to you.
- **Seek Confinement:** Seek times are roughly proportional to the linear distance across the face of the platter surfaces that the actuator must move the read/write heads. Using platters of smaller diameter improves seek time, all else being equal, and partitioning can have the same net effect. If you split a drive into multiple partitions, you restrict the read/write heads to a subsection of the physical disk when seeking, as long as you stay within the same partition. The tradeoff is that if you do a lot of moving data *between* partitions, or accessing multiple partitions simultaneously, you'll force the heads to "jump" back and forth between two completely different areas of the disk, reducing performance. Some who truly desire performance over all else will buy a hard disk with double the capacity that they need, partition it in two pieces and use only the first half! Or use the second half only for archiving infrequently-used data.
- **Defragmentation Time:** Larger partitions tend to become full of, well, more data, obviously. :^) A larger partition can take much longer to defragment than a smaller one. Since fragmentation reduces performance, some people prefer to partition their drives to reduce defragmentation time, enabling them to do it more frequently.

There are, of course, non-performance issues in partitioning. See this long description of partitioning issues in the file system section as well.

Next: Cluster Size

Cluster Size

The FAT file system used by DOS and Windows divides all of the file data on the hard disk into clusters comprised of multiple sectors. A cluster is normally between 2 kiB and 32 kiB in size, in powers of two, containing between 4 and 64 sectors of user data. This is done to make managing the location of data easier. Clusters and related file system structures are discussed here.

The choice of cluster size has an impact on real-world performance, though for most people it is not all *that* significant. In a nutshell, larger clusters waste more space due to slack but generally provide for slightly better performance because there will be less fragmentation and more of the file will be in consecutive blocks. This occurs because when clusters are larger, fewer of them are needed than when they are small. A 10,000 byte file would require three 4 kiB clusters but only one 16 kiB cluster. This means this file will always be in a contiguous block if stored in a 16 kiB cluster, but could be fragmented if stored in a 4 kiB cluster size partition. (The slack tradeoff is a waste of 4 kiB more storage in the case of the 16 kiB clusters.) Small cluster sizes also have a negative effect on partition because they require larger file allocation tables, to manage their much larger numbers of clusters. These tradeoffs are discussed in detail here.

Traditionally, most people have tried to use cluster sizes as small as possible in order to reduce slack and make more efficient use of disk space. This is of course a valid goal, but it has become increasingly irrelevant today as hard disks approach truly gargantuan sizes and the price per GB of storage drops to amazingly low levels. Today, the large file allocation tables resulting from enormous FAT32 partitions means that balancing slack reduction with performance effects is also important, unless you are on a very tight budget. I certainly can't recommend forcing Windows to use 4 kiB clusters on a 30 GB partition "to reduce slack" as some people do, because I personally wouldn't want to take the performance hit of having 30 MiB file allocation tables--and I wouldn't want to have to wait for that puppy to defragment either! :^)

Next: Volume Free Space

Volume Free Space

A relevant performance consideration that most people don't pay attention to is how full their hard disk is. The amount of free space on a hard disk affects the performance of the drive, for most of the same reasons that partitioning affects it:

- The more data on the volume, the more the data is spread out over the disk, reducing positioning performance;
- A disk that is more full forces new files to the inner tracks of the disk where transfer performance is reduced; and
- Drives that are full both take longer to defragment and tend to be more fragmented in the first place.

The "Peter Principle" of hard disks is that the amount of junk put on a hard disk expands to fill the available space, regardless of the size of the hard disk. Imagine what PC users 10 years ago would have thought about people with 6 GB hard disks needing an upgrade because their "disk is too full!" I had the personal experience the other day of surprisingly discovering that a 15 GB hard disk volume I had just installed was down to 2 GB free! Most people don't clean out their disks until they have to. :^)

The bottom line though is clear: the more "stuff" you put on that humongous hard disk you just bought, the more you will slow it down. :^) Don't fill your drive with clutter just because you have the space. Regularly go through your hard disk to get rid of files you don't need; if you think you will need them "at some point" then archive them to a tape, CD-R disk or other removable medium.

Another impact of this is that you cannot reliably compare performance benchmarks even on the same disk in the same system if you change the amount of data on the drive between runs. All drives will tend to show a reduction in performance as you fill them up, so benchmarking should always be done on empty drives to eliminate this variable.

Next: Fragmentation

Fragmentation

Fragmentation refers to the tendency of files stored in the FAT file system to become broken into pieces that can end up in very different places within a hard disk volume. How fragmentation occurs is explained in detail here.

A fragmented file system leads to performance degradation. Instead of a file being in one continuous "chunk" on the disk, it is split into many pieces, which can be located anywhere on the disk. Doing this introduces additional positioning tasks into what should be a sequential read operation, often greatly reducing speed. For example, consider a 100,000 byte file on a volume using 8,192 byte clusters; this file would require 13 clusters. If these clusters are contiguous then to read this file requires one positioning task and one sequential read of 100,000 bytes. If the 13 clusters are broken into four fragments, then three additional accesses are required to read the file, which could easily double the amount of time taken to get to all the data.

Defragmenting a very fragmented hard disk will often result in tangible improvements in the "feel" of the disk. To avoid excessive fragmentation, defragment on a regular basis; usually once every week or two is sufficient. See the system care guide for more.

Next: Disk Compression

Disk Compression

Compression is a technique that is used to reduce the amount of space required by files on a hard disk volume. Very popular during the late 1980s and early 1990s, compression is rarely used today on new systems. (I am speaking of *volume* compression here, not file-by-file compression such as ZIP files, which are as popular as ever!) Compression is discussed in detail in this section.

Most people have been told (repeatedly) to avoid compression because it greatly reduces performance. This is true to some extent, but the picture isn't nearly as simple as this blanket statement might suggest. Compression adds overhead to every disk access but reduces the amount data that needs to be retrieved from the platters for a given file. With modern high-speed CPUs, the overhead isn't nearly the issue it once was, while hard disks haven't improved in performance by nearly the same percentage. Therefore, there is a performance tradeoff at work here.

The proliferation of huge, cheap hard disks has made volume disk compression largely irrelevant today. I don't recommend it except for use on older systems that cannot be upgraded to a larger disk for one reason or another. The reason for this is not primarily due to performance effects, but rather because compression complicates disk usage and simply isn't needed in an era where hard disk space costs half a penny per megabyte.

Next: Hard Disk Quality and Reliability

Hard Disk Quality and Reliability

"Every drive dies; not every drive really lives."
-- Braveheart meets 21st century technology. :^)

A hard disk's performance is its most important characteristic--right up until the point where it stops working. Then, suddenly, you don't care how fast it is--or rather, *was*. You just want it to start working again. (OK, stop groaning already about the quote. :^))

Many people take their hard disk drives for granted, and don't think about their reliability much (other than worrying about their disk crashing some day). While the technology that hard disks use is very advanced, and reliability today is much better than it has ever been before, the nature of hard drives is that every one will, some day, fail. It is important to understand how drives fail and why, and how to interpret what manufacturers claims about reliability really mean.

This section takes a comprehensive look at hard disk quality and reliability. First, I explain in some detail the different hard disk quality specifications that you are likely to encounter when shopping for a hard disk. Next, I discuss issues relevant to the matter of quality and reliability. I then discuss some of the features that hard disk manufacturers are including in their drives to improve quality and reliability. Finally, I discuss warranty issues, and what to do should you have a hard drive failure.

Next: Hard Disk Quality and Reliability Specifications

Hard Disk Quality and Reliability Specifications

There are a number of different specifications used by hard disk drive manufacturers to indicate the quality and reliability of their products. Some of these, such as MTBF, are frequently discussed (but not always all that well understood). Others are obscure and typically of interest only to hard drive aficionados. All are important to those who care about hard disk quality--which should be anyone who stores data on a hard disk. :^) In this section I discuss the most important of these specifications, what they mean, and perhaps most importantly, what they *don't* mean! You'll also find some discussion of specifications in the section on quality and reliability issues, particularly temperature specifications and noise specifications.

Note: In addition to the hard-disk-specific numbers explained in this section, hard disks usually come with a number of *environmental* specifications that dictate how they should and should not be used in order to operate reliably. These are essentially the same as those provided for power supplies, so instead of repeating them I will refer you to the power supply section. The only caveat about applying the power supply environmental specifications here is that hard drives are more sensitive to *altitude* than most components and can fail when operated at altitudes over 10,000 feet; see here for the reason why.

mehedi hasan
mehedi@joinme.com

Next: Mean Time Between Failures (MTBF)

Mean Time Between Failures (MTBF)

The most common specification related to drive reliability is *mean time between failures* or *MTBF*. This value, usually measured in hours, is meant to represent the average amount of time that will pass between random failures on a drive of a given type. It is usually in the range of 300,000 to 1,200,000 hours for modern drives today (with the range increasing every few years) and is specified for almost every drive.

This number is very often misinterpreted and misused. Usually, the "analysis" goes like this: "Gee, a year contains 8,766 hours. That means my 500,000 MTBF drive should last 57 years." (I have even seen this *on the web site of a major hard disk manufacturer* that shall remain nameless to spare them the embarrassment!) After concluding that the MTBF means the drive will last for decades, amusingly, one of two *opposite* things usually happens: either the person actually thinks the drive will last half a century or longer, or the opposite: they realize this is crazy and so they write off the entire MTBF figure as "obvious exaggeration and therefore useless". The real answer of course is neither. (It is obviously impossible for any individual hard disk to be tested to anywhere near the amount of time required to provide a MTBF factor near even 100,000, never mind 500,000.)

To be interpreted properly, the MTBF figure is intended to be used in conjunction with the useful service life of the drive, the typical amount of time before the drive enters the period where failures due to component wear-out increase. MTBF only applies to the *aggregate analysis of large numbers of drives*; it says nothing about a particular unit. If the MTBF of a model is 500,000 hours and the service life is five years, this means that a drive of that type is supposed to last for five years, and that of a large group of drives operating within this timeframe, on average they will accumulate 500,000 of total run time (amongst all the drives) before the first failure of any drive. Or, you can think of it this way: if you used one of these drives and replaced it every five years with another identical one, in theory it should last 57 years before failing, on average (though I somehow doubt we'll be using 10 to 100 GB spinning-platter hard disk drives in the year 2057. :^))

There are in fact two different types of MTBF figures. When a manufacturer is introducing a new drive to the market, it obviously has not been in use in the real world, so they have no data on how the drive will perform. Still, they can't just shrug and say "who knows?", because many customers want to know what the reliability of the drive is likely to be. To this end, the companies calculate what is called a *theoretical MTBF* figure. This number is based primarily upon the analysis of historical data; for example: the historical failure rate of other drives similar to the one being placed on the market, and the failure rate of the components used in the new model. It's important to realize that these MTBF figures are estimates based on a theoretical *model* of reality, and thus are limited by the constraints of that model. There are typically assumptions made for the MTBF figure to be valid: the drive must be properly installed, it must be operating within allowable environmental limits, and so on. Theoretical MTBF figures also cannot typically account for "random" or unusual conditions such as a temporary quality problem during manufacturing a particular lot of a specific type of drive.

After a particular model of drive has been in the market for a while, say a year, the actual failures of the drive can be analyzed and a calculation made to determine the drive's *operational MTBF*. This figure is derived by analyzing field returns for a drive model and comparing them to the installed base for the model and how long the average drive in the field has been running. Operational MTBFs are typically lower than theoretical MTBFs because they include some "human element" and "unforeseeable" problems not accounted for in theoretical MTBF. Despite being arguably more accurate, operational MTBF is rarely discussed as a reliability specification because most manufacturers don't provide it as a specification, and because most people only look at the MTBFs of new drives--for which operational figures are not yet available.

The key point to remember when looking at *any* MTBF figure is that it is meant to be an *average*, based on testing done on many hard disks over a smaller period of time. Despite the theoretical numbers sometimes seeming artificially high, they do have value when put in proper perspective; a drive with a much higher MTBF figure is *probably* going to be more reliable than one with a much lower figure. As with most specifications, small differences don't account for much; given that these are theoretical numbers anyway, 350,000 is not much different than 300,000.

Overall, MTBF is what I consider a "reasonably interesting" reliability statistic--not something totally useless, but definitely something to be taken with a grain of salt. I personally view the drive's warranty length and stated service life to be more indicative of what the manufacturer *really* thinks of the drive. I personally would rather buy a hard disk with a stated service life of five years and a warranty of three years, than one with a service life of three years and warranty of two years, even if the former has an MTBF of 300,000 hours and the latter one of 500,000 hours.

In the real world, the actual amount of time between failures will depend on many factors, including the operating conditions of the drive and how it is used; this section discusses component life. Ultimately, however, luck is also a factor, so keep those backups current.

Next: Service Life

Service Life

While every product has a life expectancy, not every industry acknowledges this up front! Due to the incredibly precise nature of the technology used in hard disks (we're talking about millionths of an inch in many regards) and the fact that mechanical components are used, it is impossible to guarantee the reliability of even the highest-quality disks for more than a few years. Hard disks follow the so-called bathtub curve model of failures: they have a relatively high rate of "infant mortality" (or early failure), a period of very low failures for several years, and then a gradual reduction in reliability as they reach the end of their useful life. To ensure that those who rely on hard drives do not continue to use them as they enter the period of increased failure that results from component wear-out, every hard disk has defined as one of its specifications the manufacturer's intended *service life* for the product.

The service life of a modern hard disk is usually about three to five years. In my opinion, what the manufacturer is telling you with this number is this: "If your drive gives you good service for this length of time without any trouble, you've gotten good value from it and you should consider getting a new drive if reliability is important to you; the drive isn't going to fail the instant it hits this age, but the chances of it failing will increase significantly the longer you use it past this point". This number is in many ways more important than the "impressive" MTBF numbers that are so eagerly thrown around, and is in fact intended to be used in conjunction with MTBF to present a realistic picture of drive life and reliability.

Interestingly, the claimed service life is often longer than the warranty period for the drive. For example, the service life might be five years but the warranty period only three years. Think about what this means. ;^) Basically, it says that the manufacturer thinks the drive *should* last five years, but they aren't going to bet on it lasting more than three! I personally think the warranty period is a better indication of a drive's true useful life--to a point--because the warranty period is where the manufacturer "puts their money where their mouth is". Sometimes warranty periods are made longer for marketing reasons, but no manufacturer will warrant a product for three years if they expect to see significant problems in that time, because they will lose their shirts.

If reliability is important, you may want to make sure you get a new hard disk before your warranty period is up, or at the very least, before the drive reaches the end of its service life. Of course, at the rate technology changes, in three years you will probably want a new disk for performance reasons as well. Oh, and it goes without saying that anyone who cares about hard disk reliability should be performing regular backups.

The real world life of a hard disk is almost always going to be significantly higher than the stated service life, which tends to be very conservative. One reasons why is that even if the warranty doesn't cover the entire stated service life, most companies don't want to see their drives failing within the period of time that they say they won't--it looks bad. Another reason is because the service life only represents when the *odds* of failure increase; there is still a statistical distribution of actual failures. I have used many hard disks that have been in operation for 10 or even 12 years, and these are older technology drives that never had anywhere near the stated reliability levels of today's disks. So there certainly is no reason to believe that the day after your drive's third anniversary, you need to yank it and replace it with a new one. But just remember that the longer you go past the drive's service life, the more the chances of a failure increase. The environmental conditions, and how the drive is used, will also have an important impact on its overall life. See this section for more on component life.

Next: Start/Stop Cycles

Start / Stop Cycles

The read/write heads of the hard disk float on a cushion of air over the surface of the platters. Each time the power is applied to the drive, the platters spin up to speed, and the buildup of air under the heads causes them to lift off the data surfaces. When the power is cut, the platters spin down, the cushion of air dissipates, and the heads drop back to the surface of the drive. (Actual contact with the drive normally occurs in a dedicate landing zone, to ensure the heads don't contact parts of the disk containing user data.)

Each time the drive starts and stops a small amount of wear occurs to the heads and also to other components such as the spindle motor. For this reason, hard drives are given a specification for the minimum number of *start/stop cycles* they are designed to handle during their service life. The value for a desktop drive is typically between 30,000 and 50,000 cycles (and remember that this is not an average, but a *minimum*). Notebook drives, which are more commonly spun up and down a great deal to save battery power, usually have even higher numbers.

This specification, like MTBF and service life, provides a useful clue about the quality of the hard disk, but should not be sweated over. In the great-and-never-ending debate over whether to leave hard disks running or spin them down when idle, some look at the fact that start/stop cycles are specified as evidence that stop/start cycles are "bad" and therefore that drives should always be left running 24/7. As always, maintain perspective: if you start your hard disk in the morning and stop it at night every day for three years, that's only about 1,000 cycles. Even if you do it ten times a day, every day, you're not going to get close to the minimum specification for almost any quality drive, and for notebooks the problem is less significant because the drives are generally designed to withstand many more cycles.

IBM drives that use its head load/unload technology are often given a specification for *minimum load/unload cycles* instead of start/stop cycles. This is basically the same concept, except that the numbers are typically much higher. Some IBM notebook drives are spec'ed for 300,000 load/unload cycles! Even if you started and stopped the drive 100 times a day it would take you over eight years to get to that number.

Next: Error Rates

Error Rates

Ack! Hard disk errors! Surely, "this is the stuff that PC nightmares are made of". :^) Fortunately, true errors are rarely encountered when using hard disks. To help users understand better the rate at which errors will occur with a hard disk, manufacturers provide anywhere from one to several *error rate* specifications.

The most common error rate spec is the drive's *unrecoverable error rate*, which is usually specified as "<1 in 10^N bits", where "N" is usually between 12 and 15. "Unrecoverable" means that the drive is unable to use its error-correcting code, retries or other techniques to recover from the error in reading the disk and thus properly recreate the data. If "N" is 14, then that means this will occur every 100 trillion (100,000,000,000,000) bits read from the disk. Not too shabby. :^)

In fact, drives usually have several different error rate specifications; they are just usually not put into the short data sheets commonly distributed by hard disk manufacturers. In fairness, unrecoverable errors are the most important ones, but there are also specifications for recoverable errors, errors recovered after multiple reads and so on. To find these, you generally need to download the product manual for the drive or contact the manufacturer's technical support department. For a full explanation of these various errors and what they mean, see this full discussion of errors and error recovery.

Error rate specifications are typically used to compare drives. Within the same general class of drive there are usually relatively few differences between manufacturers. The biggest difference in error rate can be seen by comparing newer drives to older ones--newer drives are usually significantly better *despite* pushing the design envelope with higher speed and much greater capacity.

Next: Warranty Length

Warranty Length

One of the most important quality specifications, the length of the warranty is one that can only be considered *indirectly* technical, because it is more of a manufacturer policy than a technical measurement. It is however a number that "boils down" in rough terms the quality of the disk drive, for the simple reason that if a manufacturer warrants a drive for much longer than the drive actually is expected to last, they lose a *lot* of money when the warranty claims come pouring in.

Retail warranties for hard disks are usually either three years or five years in length. Three-year warranties are typical for consumer-grade (usually IDE/ATA) drives, and five-year warranties are the norm for "enterprise" class (usually SCSI) drives. I am not sure if it was by design that these two standards came to be, but they are fairly universal. It seems likely that the companies match each others' warranty length to avoid getting into a competitive battle that would stretch warranty lengths out and cost all of the manufacturers in the end.

Does the fact that a drive has a five-year warranty mean that it is higher in quality than one with a three-year warranty? The answer is "yes and no". To some extent these more expensive drives are manufactured to more exacting standards because they are sold for much higher prices to clients who plan to use them in often-critical business applications. However, at least *some* of the motivation behind longer warranties is to sell the drives as being high quality and/or to match the warranty periods of other companies. It's a bit of a "chicken-and-egg" situation. And since every drive you look at it in a given class is likely to have the same length of warranty as any other, the length of the warranty doesn't do much to help you differentiate between drives. (Although it does establish a standard for a given class of drive and give you reason to be suspicious of any drive with a warranty that is much below that standard.)

Here's something else that most people don't really think about--especially those who recommend high-end drives with five-year warranties over "cheaper" drives with "only" three-year warranties: are you really still going to be *using* that new drive in four years? Considering that most people who buy high-end drives need high performance, and considering that the performance of today's drives will be a joke compared to that of drives made in four years, the answer for many people is "no". The length of the warranty doesn't matter if the drive is sitting in a box with your leftover stereo cables and that extra printer cable. However, if you tend to buy a drive and use it for many year, that extra two years of warranty could be quite valuable.

There's a lot more to a hard disk's warranty than its length; it's quite fair to say that all warranties are not created equal. For example: what is the manufacturer's policy for replacing drives that fail during warranty? That's often at least as important as how long the warranty coverage is. I have written this separate section to discuss the various issues involved in hard disk warranties.

Next: Hard Disk Quality and Reliability Issues

Hard Disk Quality and Reliability Issues

Quality and reliability specifications are theoretical numbers that reflect the drive's quality characteristics under average--or in some cases "ideal"--conditions. In actual operation, however, the reliability of a hard disk depends as much on how the storage subsystem is implemented as it does on the characteristics of the drive itself. No implementation factors can make the drive *more* reliable than it is specified to be, but mistakes in the way the drive is used can sometimes make it *less* reliable. There are several different issues that impact upon this "real world" reliability.

In this section I will take a look at topics related to reliability such as cooling, noise and vibration, and power management. I will also discuss determining when a drive is failing, and also look at some software causes for reliability problems (which are often mistakenly blamed on hardware!)

Next: Hard Disk Quality: Luck Of The Draw?

Hard Disk Quality: Luck Of The Draw?

If you read the technical groups on USEnet or spend time on technically-oriented Web discussion groups for any reasonable length of time, you will see people post asking "what is a good brand of X?" all the time, where X is any type of component. When "X" is "hard drive", the results are usually pretty easy to predict: virtually every major manufacturer will be mentioned by at least one person as being "great" and by another as being "horrible". The end result is that you have no idea what to do. Why does this happen?

The simple fact of the matter is that most major hard drive manufacturers make very high quality products, and most hard disks provide their owners with years of very reliable service. However, all manufacturers make the occasional bad drive, and sometimes, manufacturers will have a problem with a particular product. If you happen to buy one of these, you will experience a failure, and in all likelihood you will hate that company and avoid their products from then on, perhaps with good reason. The problem is that many people will generalize this very small sample size into "Brand X sucks", when this very well may not be the case. They just may have been unlucky with what might in actuality be one of the best drives on the market. Meanwhile, they will post a frustrated-sounding message telling everyone "DON'T BUY X!" Human nature is such that there could be thousands of people using that particular model with no problems, but almost none will bother to post saying "I bought a new PC with an X hard drive in it and every day I use it, it doesn't fail!" :^)

There are occasions where manufacturers will go into "slumps" and have time where their products fall in quality compare to those of other companies. And of course, if there is a known issue with a specific model, you should avoid it. The key when doing your research is to look for *trends*. Don't over-value a single sample regardless of what it says.

There are many ways that a hard disk can fail. The one that usually comes to mind first, the infamous "head crash" is not the only way or even necessarily the most common any more. There can also be problems with bad sectors showing up on the drive, for example. Many people don't think of this, but the integrated controller can also sometimes be the culprit for a bad drive. See this discussion of failure modes, and this section in the Troubleshooter for specifics on drive failures and possible solutions.

Next: System and Drive Cooling

System and Drive Cooling

In order to ensure long life and proper reliable operation, all hard disks are designed to function only in specific temperature ranges. The user of the hard disk must keep the drive within the specifications to be sure that the drive will continue to work well, and to maintain the drive's warranty status. You can find out the temperature allowances for a particular drive model by consulting the drive's product manual or data sheet, normally available for free download at the drive manufacturer's web site.

There are in fact several different temperature limits that are specified for hard disks:

- **Non-Operating Temperature Range:** This is the range of acceptable temperatures for the drive when it is either in storage, or in a PC that is in transit or otherwise not operating. This range is normally very wide, much wider than the operating temperature limits, since the unit is much less sensitive to extremes of temperature when it is not functioning. A typical range would be -40°C (-40°F) to 70°C (158°F). Clearly few users will have a problem with these numbers.

Warning: If a drive is allowed to go below freezing, or if it is quickly exposed to a large temperature change, it must be *acclimated* before use. See here for more details.

- **Minimum Operating Temperature:** The lowest acceptable temperature for the drive when in operation. A single number is normally provided for this value, with 5°C (41°F) being typical. Again, due to the heat generated by the hard disk and the fact that almost all PCs are used indoors, this is rarely much of a concern.
- **Maximum Operating Temperatures:** The highest temperature allowed for the drive when in operation. Since the mechanical and electrical components within the hard disk--especially the spindle motor--produce heat, the biggest problem with keeping drives within operating parameters is not exceeding maximum allowable temperatures. A number of different temperature values are usually provided, depending on the drive manufacturer and model:
    - **Case Temperature:** The highest acceptable temperature allowed, measured at a specific point on the metal of the case.

- **Component Temperatures:** Some manufacturers provide a number of different maximum temperature values, measured on the surface of various components on the drive (especially, the hard disk's logic board). This method is more precise and takes into account the fact that some components run hotter than others. In practice, due to the amount of work involved, these different temperature measurements are rarely used except in special applications.
- **"Absolute" and "Reliability" Temperatures:** Some companies provide two sets of case and component temperature limits. The first set is the absolute maximum temperature(s) allowed for the drive. The second set is the maximum allowed for the drive in order to meet its reliability and MTBF specifications. Of course, the reliability temperatures are lower than the maximum temperature. Since reliability is so important for hard disks, the "reliability" temperatures are the ones to aim for.

The temperature at which the drive operates is dependent on the temperature of the system--you can have a drive run too hot in a system that is otherwise quite cool, but if the rest of the system case is hot, the drive doesn't have a chance. See this discussion of system cooling for more on this subject. Staying within temperature tolerances is usually only a problem with newer, faster drives with high spindle speeds, and in fact, heat is one of the prices that those who need the best performance must sometimes pay. In some cases active drive cooling is required.

See this full discussion of drive cooling options for more on this important subject.

Next: Noise and Vibration

Noise and Vibration

Due to their mechanical nature, there are several characteristics of hard disks that aren't relevant to most other PC components. Among these are the *noise* and *vibration* created by the disk drive. (Noise and vibration are of course two parts of the same phenomenon.) Since most devices inside the PC have no moving parts, they don't make noise. Hard disks on the other hand have very high-speed motors and actuators, both of which can make considerable noise and generate vibrations. Like cooling, noise issues have become exacerbated in recent years primarily due to the increase in spindle speeds of modern drives. Faster actuators are also part of the issue.

Hard drive noise specifications are normally provided as two numbers. The values correspond directly to the two different mechanical devices that make noise in the drive:

- **Idle Noise:** This is the noise level the drive makes whenever the drive is in operation (and not in sleep or standby mode), regardless of what it is doing; it is caused by the spindle motor and the rotation of the platters.
- **Seek Noise:** This is the noise level when the drive is performing positioning tasks, particularly random seeks. It is caused by the movement of the actuator assembly (as well as the spindle motor and platters of course, since they continue to spin during seeks!)

Hard drive noise is specified by the manufacturer in *bels* (or sometimes *decibels*; they are the same unit except that one bel equals ten decibels.) A "bel" is a logarithm of a ratio between two sound values. It is defined as follows:

Noise level (bels) = $\log_{10} (N_1 / N_0)$

Where "$\log_{10}$" means a base-10 logarithm, "N_1" is the noise level of the device being measured, and "N_0" is a *reference* noise level. (Usually, N_0 is taken as the threshold of human hearing, the quietest sound audible to the average person. The actual value of N_0 is not important as long as all the noise figures are calculated using the same reference point.) The reason for the logarithm is notational convenience; the human ear can detect sound intensities in a *tremendous* range: up to a factor of 1,000,000,000,000 from the quietest to the loudest. Logarithmic notation expresses these sounds in a range of 0 to 12 bels, saving us all those zeroes. :^)

Note: Be sure to remember, when comparing noise specifications, that the values are *base-10 logarithmic*. That means a hard disk with a noise specification of 5.0 bels is not 25% louder than one that reads 4.0 bels; it is 900% louder: *10 times louder*.

Companies always provide noise specifications, and some hard disk reviewers have started measuring noise levels with noise meters. Frankly, I don't put a lot of stock in these measurements and figures. I would even go so far as to say that they have *almost no bearing on real world noise perception* and by themselves are not useful for deciding if a drive model is going to give you a noise problem. Here are my reasons:

- Noise is a very *subjective* and personal matter. It's very possible to put two people in a room with the same system and have person A be "driven crazy" by the noise and person B not even realize the PC is turned on.
- Some types of noise bother certain people more than others, because humans have different sensitivity to sounds of various frequencies. For example, some people find certain drives have a high-pitched "whine" due to their spindle motors that others can't even detect; some find the spindle noise acceptable but the seek noise irritating, and some are the opposite.
- Hard disk noise levels depend a *great* deal on the case in which they are installed, and the location of the case relative to where the user is sitting. A small desktop case right in front of the user will make noise seem much more significant than a large tower case with the hard disk mounted in the rear, sitting on the floor next to a desk! And if a high-speed drive is installed in a server in a locked, ventilated, insulated computer room, does its noise level matter much at all?
- Absolute noise figures don't mean anything unless you take them in the context of the surroundings. If a drive has a high noise rating then it might be considered "noisy", but if you are using it in a crowded office with a lot going on, you're probably not going to hear it.
- Noise levels can sometimes vary between specific units even of the same model of drive.
- While I trust manufacturers to make accurate sound readings, some of the individual hardware reviewers using sound meters haven't been properly trained in their use. Sound measurement is not a simple thing; you must compensate for the noise of other components and other effects to ensure the readings are accurate.

So fine, look at the specifications for the drive. Clearly, if there is a big difference in the noise values, you are *more likely* to have a noise issue with the drive that has the higher specification. But in reality, it all depends on your personality, your surroundings, and your ears. If you aren't the type to care about the noise level of your PC, just don't worry about the whole issue. If you are, then you should if at all possible find a PC that has the drive you are considering installed into it, and listen to it yourself. That's the ideal trial step, but not always possible. A second choice is to look on the Internet for *a significant number of subjective comments*. If dozens of people are complaining about the noise level of a drive, then there is a good chance the drive may bother you; look for one that is commonly used and has many comments saying the drive is quiet.

Another key issue with hard disk noise is that while seek noise *specifications* just consist of the actuator noise "added" to the spindle noise, they are very different kinds of noise! If you are sitting in front of a PC for hours, your brain will tend to "tune out" the background noise of the spindle. The noise of the actuator is intermittent however, and much louder. It is this noise that often causes the most objections, and again, it's a very personal thing.

To some extent, you must realize that noise and vibration are a price you pay for increased performance. Idle noise is directly correlated to spindle speed, and seek noise is correlated to faster actuators (and hence reduced seek times). It is also true however that manufacturers are constantly improving their products, which means that the *generation* of a given technology is important. For example, the first 7200 RPM drives were rather noisy and hot; now 7200 RPM is a mature technology and the drives of this speed run *much* quieter and cooler than their ancestors.

An important final note: to minimize vibration and noise in a PC, ensure that the drive has been *properly mounted*. If the drive is loose then its inherent vibrations can become amplified by the metal that touches it, making the entire PC seem to resonate in some cases! See here fore more on hard disk mounting.

Overclocking

A popular activity undertaken by many PC hobbyists is the process of *overclocking*. This refers to running hardware above its standard and rated speed specifications in order to increase performance. I don't think overclocking makes sense for most people; I provide some background on overclocking, and my opinions on it, in this section.

Overclocking a PC inherently reduces the reliability of virtually every part of the system. Since a PC is interconnected, and overclocking usually affects key components such as the system processor and the motherboard, in some ways everything in the system is involved. When overclocking involves running just the CPU at a higher speed, it has little *direct* reliability effect on the storage subsystem. However, when overclocking the system bus, reliability concerns with hard drives often come into play. Overclocking the PCI bus upon which the IDE/ATA interface runs can cause IDE hard disks to behave spuriously. Since most SCSI host adapters also run on the PCI bus, they can also behave strangely if the PCI bus is run faster than its normal speed.

How a particular hard disk will react to overclocking is not easy to determine, and like most overclocking activities, requires trial and error. It also depends on how far you are pushing things, and especially, on the type of motherboard you are using and its integrated hard disk controller. However while it is true that "your mileage may vary", as the saying goes, it is also the case that some hard disk brands and models take to overclocking more than others. Research on any of the hundreds of overclocking sites, newsgroups and other resources can help you decide on a suitable drive if overclocking is one of your priorities.

Next: Diagnostic Software

Diagnostic Software

All hard disk manufacturers write and maintain special *diagnostic software* for their drives. The purpose of diagnostic software is simple: to test drives and diagnose potential problems with them. These programs usually come with retail drives on a floppy disk, or can be downloaded from the manufacturer's web site (at no charge). In some cases a single program will function for any of the manufacturer's drives, but in other cases a different program is needed depending on the drive model; check the manufacturer's instructions.

If you suspect a problem with your hard disk, you should always run a diagnostic program to check the status of the drive. For one thing, the technical support department of most manufacturers will require the information provided by the diagnostic tool in order to help you with any problems you experience with the drive. Some will require an error code or other status information before agreeing to declare a drive defective and issue an RMA for it.

Often, the diagnostic functions of these utilities are combined with other features for erasing the disk and are sometimes (incorrectly) called "low-level format utilities". For more information on this software, see this section.

Warning: I recommend against using any diagnostic utility or other software tool written by one manufacturer on another manufacturer's drives, unless the manufacturer of your drive instructs you to do so. While it is very unlikely that a problem will result--such software usually interrogates the drive to determine its type and will refuse to touch a drive made by another manufacturer--there is always the possibility of problems. It's best to use utilities designed specifically for your particular make and model.

Tip: The operative word when using diagnostic software is "software". Like any software program, diagnostic utilities can occasionally have bugs or problems. There was recently a version of a diagnostic utility that was incorrectly returning error codes on drives that turned out to be perfectly fine. Be sure to double-check any error reports with your drive maker's technical support department before concluding that the drive is bad.

Next: Failure Modes and Failure Determination

Failure Modes and Failure Determination

There are a large number of different ways that a hard disk can fail; these are normally called *failure modes*, and determining that a failure has occurred--and why--is called *failure determination*. The most "famous" way that a hard disk can go is via the infamous head crash. In reality, head crashes are the "airline disasters" of the hard disk world: they are dramatic, well-known and feared, but actually responsible for a small percentage of total drive problems.

A comprehensive list of failures that can cause a drive to fail would be as long as your arm, and there's really no point in compiling one. This short list will give you a flavor for the types of failures that occur:

- **Mechanical Failures:** These include component failures related to the mechanics of the disk. Problems of this type usually relate to the spindle motor or bearings, such as motor burnout, "stuck" bearings, excessive heat, or excessive noise and vibration. Actuator problems would also fit into this general category. Unsurprisingly, mechanical failures of hard disks comprise a large percentage of total problems.
- **Head and Head Assembly Failures:** The infamous head crash fits in here, as do other problems related to the heads: improper flying height, head contamination, defects in head manufacture, excessive errors on reads or writes, bad wiring between the heads and the logic board. These too comprise a large percentage of total failures.
- **Media Failure:** This class of problems relates to trouble with the platters and the magnetic media, formatting, servo operation and the like. This would include drives that fail due to read or write errors, poor handling, scratches on the media surface, errors in low-level formatting, etc. They are relatively uncommon.
- **Logic Board or Firmware Failures:** These are problems related to the drive's integrated logic board, its chips and other components, and the software routines (firmware) that runs it. Again, problems in this area of the disk are fairly uncommon compared to the other categories.

Determining that your drive has a failure isn't always the simplest thing to do. Some failures manifest themselves rather clearly and obviously; others are quite a bit more subtle. Here are some of the most common signs that your hard disk *may* be failing--not definitely, but possibly. (Remember that sometimes apparent drive failures turn out to only be problems with configuration, drivers or the like... also see the Troubleshooting Expert section on hard disks):

- **Errors:** If the drive is returning read errors or "sector not found" errors, that is an obvious sign of a problem. Sometimes errors appear at a "higher level"; for example, if you experience repeated corruption of the Windows registry, this could be due to a drive problem (though a hundred other things could cause it as well.)
- **Changed Behavior:** If the drive has suddenly changed its behavior in a dramatic way due to nothing you (believe) you have done, this may be a bearer of bad tidings. This is especially true of noise: if the drive

always made a particular noise then that noise is probably normal for the drive type (some models make certain clunks or clangs when they start or shut down). But if the drive starts making *new* noises then you should be concerned. If the drive starts vibrating more than it used to, or spins up and down a lot more than before, these are also potential warning signs.

- **Scary Noises:** The above said, there are certain noises a drive should *never* make, even when new. If you hear something scraping as the drive turns on, or the drive makes repeated clunking sounds during operation, or emits a very high-pitched whine, then there may well be a problem.
- **Drive Not Recognized:** If the drive is not recognized at all by the BIOS of a motherboard to which it is connected *where it was recognized by that BIOS in the past*, this is a sign of a failed drive. (Non-recognition of a new drive can be due to BIOS compatibility issues.)
- **SMART Alarm:** If your hard disk and system support the SMART feature and you receive a SMART alert, consider this fair warning that the drive may have a problem.

Warning: If you suspect that your hard disk is failing, *take action*. Back up your data immediately, and contact the drive manufacturer for instructions. See the section on warranty issues for more details.

Next: Software Causes for Reliability Issues

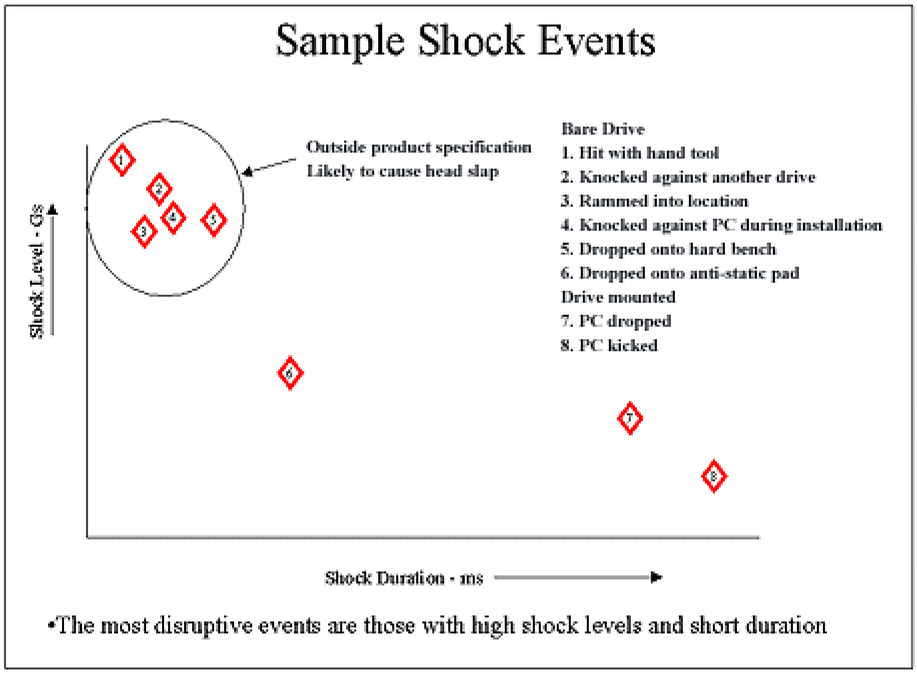
Software Causes for Reliability Issues

While most reliability problems with hard disks are due to troubles with hardware, there are some reliability concerns that are actually related more to software, or to how the hard disk is used. Software can't "break" hardware, so perhaps this is a more "loose" definition of reliability--I'm certainly not suggesting that the software you use can cause your hard disk to fail. However, the way you use your system *can* cause you to lose data or to experience system instability, and these are what most people are trying to avoid by obtaining highly reliable hard disks in the first place. Your use of the system can also sometimes create situations where it appears that there is a hard disk reliability problem but there in fact is not.

To avoid problems with your hard disk and data, you need to take care of them. I discuss data loss prevention and related issues in this section. Also, see here for a full discussion on viruses, one of the leading causes of non-hardware data loss in the PC world. Next: Hard Disk Quality and Reliability Features

Hard Disk Quality and Reliability Features

Hard disk manufacturers recognize the great importance of their customers' data, and therefore, the importance of the quality and reliability of their hard disk drives. In addition to generally improving the quality of their processes and their products, most manufacturers have added special features to their hard disks to help enhance their reliability. Most of these efforts are predicated on the principle that "an ounce of prevention equals a pound of cure". They are intended to either reduce the chances of a problem occurring, or at least let the user know if a problem is likely to occur, before data loss happens.



This chart shows several sample shock events that can affect hard disks, mapped based on their duration and intensity. Notice that the more serious shocks are those delivered to the drive by itself, while those caused to the PC as a whole are less critical to the disk. Modern drives are equipped with many reliability features that can reduce the likelihood of damage in the event of common shock occurrences, but no advances will protect adequately against outright abuse of such delicate hardware. (Not all of the more serious shock events above will necessarily cause the drive to fail or data to be lost, but they do definitely carry the potential.)

Image © Quantum Corporation
Image used with permission.

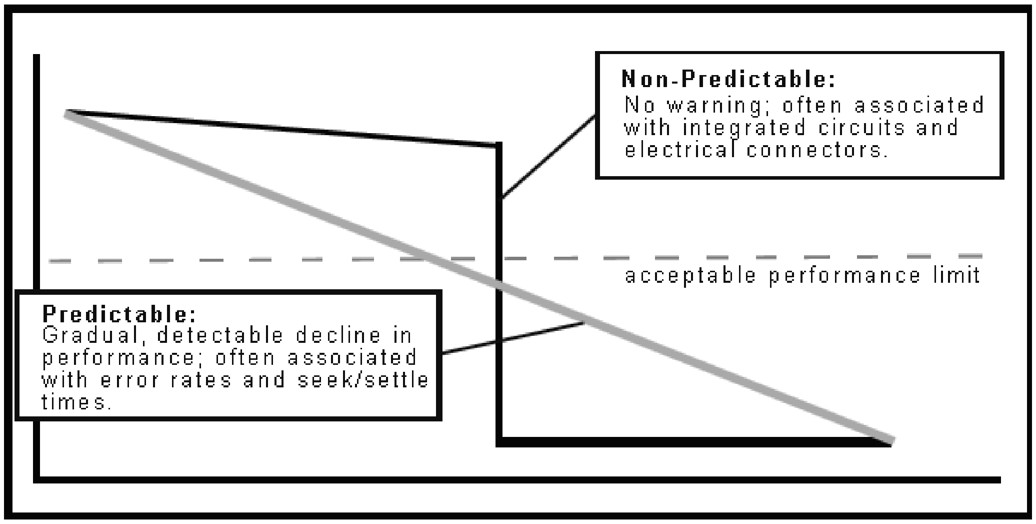
In this section I describe in detail the more common features being implemented on today's drives to enhance their quality and reliability, and to provide warning to the user if a hardware failure is imminent.

Next: Self-Monitoring Analysis and Reporting Technology (SMART)

Self-Monitoring Analysis and Reporting Technology (SMART)

In an effort to help users avoid data loss, drive manufacturers are now incorporating logic into their drives that acts as an "early warning system" for pending drive problems. This system is called *Self-Monitoring Analysis and Reporting Technology* or *SMART*. The hard disk's integrated controller works with various sensors to monitor various aspects of the drive's performance, determines from this information if the drive is behaving normally or not, and makes available status information to software that probes the drive and look at it.

The fundamental principle behind SMART is that many problems with hard disks don't occur suddenly. They result from a slow degradation of various mechanical or electronic components. SMART evolved from a technology developed by IBM called *Predictive Failure Analysis* or *PFA*. PFA divides failures into two categories: those that can be predicted and those that cannot. Predictable failures occur slowly over time, and often provide *clues* to their gradual failing that can be detected. An example of such a predictable failure is spindle motor bearing burnout: this will often occur over a long time, and can be detected by paying attention to how long the drive takes to spin up or down, by monitoring the temperature of the bearings, or by keeping track of how much current the spindle motor uses. An example of an unpredictable failure would be the burnout of a chip on the hard disk's logic board: often, this will "just happen" one day. Clearly, these sorts of unpredictable failures cannot be planned for.


Non-Predictable:
No warning; often associated with integrated circuits and electrical connectors.
acceptable performance limit
Predictable:
Gradual, detectable decline in performance; often associated with error rates and seek/settle times.

The main principle behind failure prediction is that some failures cause gradual changes in various indicators that can be tracked to detect trends that may indicate overall drive failure.

Image © Quantum Corporation
Image used with permission.

The drive manufacturer's reliability engineers analyze failed drives and various mechanical and electronic characteristics of the drive to determine various correlations: relationships between predictable failures, and values and trends in various characteristics of the drive that suggest the possibility of slow degradation of the drive. The exact characteristics monitored depend on the particular manufacturer and model. Here are some that are commonly used:

- **Head Flying Height:** A downward trend in flying height will often presage a head crash.
- **Number of Remapped Sectors:** If the drive is remapping many sectors due to internally-detected errors, this can mean the drive is starting to go.
- **ECC Use and Error Counts:** The number of errors encountered by the drive, even if corrected internally, often signal problems developing with the drive. The trend is in some cases more important than the actual count.
- **Spin-Up Time:** Changes in spin-up time can reflect problems with the spindle motor.
- **Temperature:** Increases in drive temperature often signal spindle motor problems.
- **Data Throughput:** Reduction in the transfer rate of the drive can signal various internal problems.

(Some of the quality and reliability features I am describing in this part of the site are in fact used to feed data into the SMART software.)

Using statistical analysis, the "acceptable" values of the various characteristics are programmed into the drive. If the measurements for the various attributes being monitored fall out of the acceptable range, or if the *trend* in a characteristic is showing an unacceptable decline, an alert condition is written into the drive's SMART status register to warn that a problem with the drive may be occurring.

SMART requires a hard disk that supports the feature and some sort of software to check the status of the drive. All major drive manufacturers now incorporate the SMART feature into their drives, and most newer PC systems and motherboards have BIOS routines that will check the SMART status of the drive. So do operating systems such as Windows 98. If your PC doesn't have built-in SMART support, some utility software (like Norton Utilities and similar packages) can be set up to check the SMART status of drives. This is an important point to remember: the hard disk doesn't generate SMART alerts, it just makes available status information. That status data must be checked regularly for this feature to be of any value.

Clearly, SMART is a useful tool but not one that is foolproof: it can detect some sorts of problems, but others it has no clue about. A good analogy for this feature would be to consider it like the warning lights on the dashboard of your car: something to pay attention to, but not to rely upon. You should not assume that because SMART generated an alert, there is definitely a drive problem, or conversely, that the lack of an alarm means the drive cannot possibly be having a problem. It certainly is no replacement for proper hard disk care and maintenance, or routine and current backups.

If you experience a SMART alert using your drive, you should immediately stop using it and contact your drive manufacturer's technical support department for instructions. Some companies consider a SMART alert sufficient evidence that the drive is bad, and will immediately issue an RMA for its replacement; others require other steps to be performed, such as running diagnostic software on the drive. *In no event should you ignore the alert*. Sometimes I see people asking others "how they can turn off those annoying SMART messages" on their PCs. Doing that is, well, like putting electrical tape over your car's oil pressure light so it won't bother you while you're driving! :^)

Next: Idle Time Error Checking

Idle Time Error Checking

Hard disks have extensive facilities built into them to detect, correct and avoid data errors; these are discussed in some detail here. Most of the error-recovery procedures discussed in that section are *reactive*; they are concerned with how to deal with an error situation that has occurred when data is requested by the user. As suggested in the description of SMART, sometimes problems can occur due to gradual degradation, in this case of a particular location on the media. If this is the case, it may possible to detect a "bad spot" before it gets to the point where data loss might result. Since hard disks use ECC to correct multiple-bit errors but can only do so if the number of missing or incorrect bits is below a certain threshold, detecting such errors quickly can prevent data loss that might occur if the problem were not discovered until a later time.

To exploit this principle, some manufacturers are now incorporating routines to do *proactive* error checking of the drive. During idle periods, the drive's logic board performs reads over the surface of the disk to verify that the data can be read without errors. Any errors encountered are typically treated the same way an error would be addressed during a user read: the data may be rewritten to "refresh" it, or the sector where the error is relocated may be remapped. Drives that incorporate this feature usually integrate it with SMART; if an extensive number of errors occurs during such idle checks then this can signal problems with the drive.

This is a useful reliability feature, but one that can sometimes cause confusion on the part of hard disk users. The reason is that this scanning activity can cause the drive to make "active" sounds even when the PC not doing anything involving the hard disk; the hard disk activity LED will also not be lit. (Since this scanning is purely an internal activity, it will not cause the hard disk controller or motherboard to light the LED.) If your drive is one that incorporates this feature and you see this "ghost" activity, there's nothing to be concerned about. There also should be no impact on the performance of the drive, since the scanning occurs only when the drive is idle.

Note: These scans typically run only after the drive has been idle for some number of minutes. This means you have to leave the drive on some time when you are not using it for them to activate. If your normal operating mode is to turn on the PC, use it, and then turn it off, then the drive may never become idle long enough for this idle-time scan to initiate.

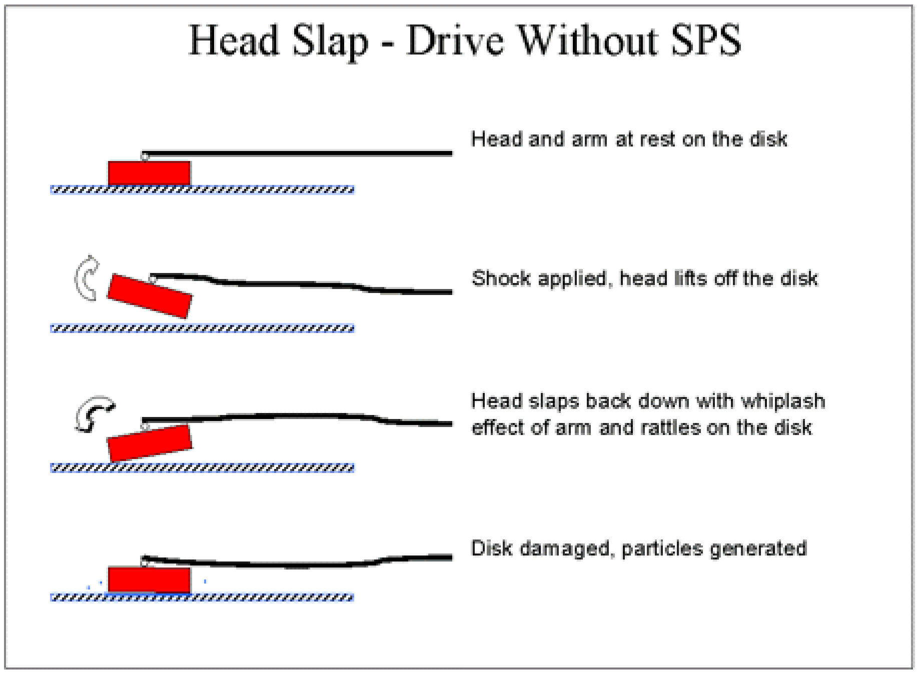
Next: Enhanced Shock Protection

Enhanced Shock Protection

Several major types of drive failures can be caused by excessive shock applied to the drive, especially when it is in operation. This is not surprising, considering the way hard disks work. When you have very precise read/write heads floating a millionth of an inch above the surface of a smooth platter spinning thousands of revolutions per minute, you shouldn't expect that you can give the device rough handling without consequences! When the heads contact the surface of the platters, this is called a *head slap* and each such event carries the potential for damage or data loss.

Despite the fragility of hard disk technology, we *do* expect to be able to treat them, well, if not roughly, then at least without kid gloves. We want to put hard disks into laptop computers and carry them around, or use removable drive trays, or incorporate tiny hard disks into devices ranging from handheld PCs to digital cameras. And even with desktop machines, we certainly don't expect the drive to fail if we bump into the case while the PC is on, or similar minor mishaps occur.

To address the increased needs for portability, and to improve reliability and reduce sensitivity to shocks in a general way, hard disk manufacturers have been hard at work improving the shock resistance of drives. One great way to do this would be to basically pack cushioning material around the drive; unfortunately, this would create other problems, ranging from needing to enlarge the drive's form factor, to making cooling more difficult. Therefore, shock protection is primarily done by increasing the *rigidity* of the most important components in the drive, such as the actuator assembly and spindle motor. To reduce head slap in particular, the head arms are made stronger. Structural support and some cushioning for the disk-head assembly is also provided to help reduce the transmission of shock energy from the outside of the drive to the platters and heads, where damage could occur.


Head Slap - Drive Without SPS
Head and arm at rest on the disk
Shock applied, head lifts off the disk
Head slaps back down with whiplash effect of arm and rattles on the disk
Disk damaged, particles generated

Anatomy of a head crash. Shock applied to the drive causes the heads to lift off the surface of the platters, then snap back causing damage. Newer drives using shock protection (which Quantum calls *SPS*) stiffen the head arms to greatly
reduce the chances of head slap, and reduce the damage caused if it occurs. (Note that the red rectangles are actually the *sliders*, not the heads...)

Image © Quantum Corporation
Image used with permission.

Enhanced shock protection is certainly a useful enhancement for drives, and it has been adopted by pretty much every manufacturer in one form or another. If you think about it, this is really a general way of improving the quality of a drive as opposed to a discrete "feature", and all drive makers strive to improve the quality of their units, even if they do it in different ways. Some manufacturers do make more of a "big deal" about this than others.

And by the way, it should go without saying that no amount of shock protection is license to treat your hard drives with anything but the utmost of care!

Next: Fly Height Detection

Fly Height Detection

The height at which the heads fly above the hard disk platters is one of the most important design parameters of any hard disk. The reason is that this gap represents a critical design trade-off: if the heads are too high above the surface of the disk the heads then data errors can occur, but if they are too low, the risk of a head crash dramatically increases. The fly height of the heads must be maintained within tight parameters to ensure the reliability of the drive.

Many manufacturers are now incorporating sensors into their drives to monitor the height at which the heads float over the surface of the platters. This is usually used as one of the data inputs to the drive's SMART feature, as any trend downward in fly height can signal the possibility of a head crash in the near future.

Next: Wear Leveling

Wear Leveling

Under normal circumstances--meaning, unless special reliability features are implemented--when a hard disk drive is idle, the head-actuator assembly just "sits there" wherever it happened to be when the last read or write occurred. In theory, there should not be any problem that results from this since the heads are floating on air and not touching the surface. However, at least one manufacturer (Western Digital) theorized that it might not be a good idea to keep the heads sitting in one spot for long periods of time even if in theory this was OK. After all, head slaps *can* occur, and under normal usage the location of the heads is not likely to be completely random: some files and areas of the disk are more often accessed than others, especially those where the file system structures are located.

To eliminate what they considered the potential for wear, Western Digital incorporated onto its drives a feature called *wear leveling*. Every fifteen seconds of idle time, the drive controller moves the heads by one track, traversing the entire disk area, so that no area of the disk has the heads sitting over it for very long. It takes about three hours for the entire surface of the disk to be traversed.

The importance of this feature seems to still be under debate. In theory it improves drive reliability, but it also causes the drive to make more noise. It also makes some people think that there is something wrong with their drive. (I did when I first encountered these strange clicking sounds on my drive even when it was idle; I later realized the noises were exactly 15 seconds apart every time, so I realized it must have been something that had been done intentionally.) The fact that other manufacturers have apparently not rushed to implement wear leveling says something, considering that most technologies that are universally considered "good ideas" end up on all manufacturers' drives within a few years of one company inventing them. It may be that the other manufacturers don't consider wear leveling useful enough to bother, or they may want to avoid the noise complaints that plagued some of WD's early efforts with this feature. Certainly, it would be hard to argue that wear leveling is a better way to spend idle time than say, error checking during idle time, which also moves the heads but does a lot more at the same time.

Tip: After some initial complaints about the noise caused by wear leveling on early implementations of the feature, Western Digital made available a firmware patch to reduce the noise caused by this feature. It can be found *here on their web site*. It is only necessary for drives 5.1 GB in size or below that employ wear leveling, as the patch is built into the firmware in later drives.

Next: Head Load/Unload Technology

Head Load/Unload Technology

Hard disks work by having the read/write heads fly over the surface of the disk platters. However, this floating action occurs only when the platters are spinning. When the platters are not moving, the air cushion dissipates, and the heads float down to contact the surfaces of the platters. This contact occurs when the drive spins down, as soon as the platters stop spinning fast enough, and occurs again when the spindle motor is restarted, until the platters get up to speed. Each time the heads contact the surface of the platters, there is the potential for damage. In addition to friction (and thus heat) on these sensitive components, dust can be created as the heads scrape off minute amounts of material from the platters.

Knowing that this will occur, manufacturers plan for it by incorporating special lubricants, and by ensuring that the heads normally contact the platters in a special landing zone away from user data areas. While obviously good planning, the entire way that starts and stops are handled--allowing contact with the platters--is not a great design. After several decades of handling this the same way, IBM engineers came up with a better solution.

Instead of letting the heads fall down to the surface of the disk when the disk's motor is stopped, the heads are lifted completely off the surface of the disk while the drive is still spinning, using special ramps. Only then are the disks allowed to spin down. When the power is reapplied to the spindle motor, the process is reversed: the disks spin up, and once they are going fast enough to let the heads fly without contacting the disk surface, the heads are moved off the "ramps" and back onto the surface of the platters. IBM calls this *load/unload technology*. In theory it should improve the reliability of the hard disk as a whole. Unfortunately, I am unaware of any other drive manufacturers using it at this time.

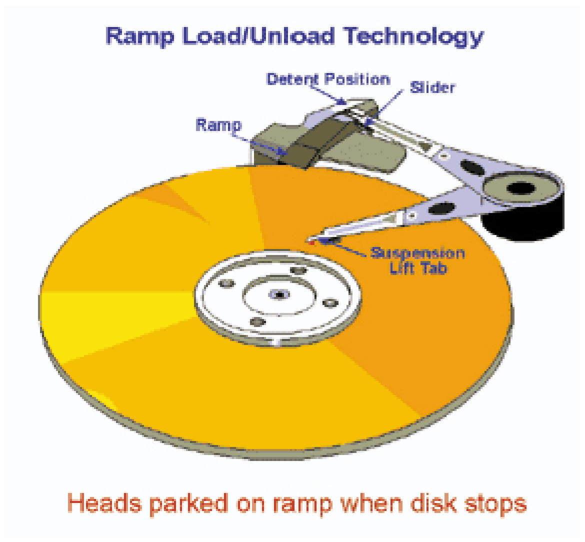

Ramp Load/Unload Technology
Detent Position
Slider
Ramp
Suspension Lift Tab
Heads parked on ramp when disk stops

Diagram showing how IBM's load/unload ramp technology functions. (One head and surface is shown; there would be a different ramp for each head and surface the disk has.)

Original image © IBM Corporation Image used with permission.

This feature appeared at about the same time that IBM went to the use of glass substrate platters. It may well be that the glass platters are more susceptible to damage from head contact than traditional aluminum platters. Regardless of the reason, load/unload technology is a great idea that I hope is adopted by other drive makers. This feature is also likely part of the reason why many IBM drives have very high start/stop cycle specifications. The only possible drawback I can see to this feature, other than possibly a bit of extra cost, is that spin-up time might be a bit slower on these drives.

Next: Temperature Monitoring

Temperature Monitoring

Since heat has become much more of a concern for newer drives, particularly high-end ones, some manufacturers have added a very good reliability feature to their drives: *thermal monitoring*. IBM calls this feature the *Drive Temperature Indicator Processor* or *Drive-TIP*. It has also been implemented by at least one other manufacturer (Western Digital).

The idea behind this feature is very simple: a temperature sensor is mounted on the drive, usually on the logic board, and it records the temperature of the drive periodically over time. One or more *trip points* are set within the drive's control logic, and status notifications are sent by the drive back to the system if they are exceeded. Normally, thermal monitoring is integrated with the drive's SMART feature for reporting. A typical setup is two trip points, one at 60°C and another at 65°C; the first trip point can be changed by the user of the drive while the second often cannot. The controller may also keep track of the highest temperature ever recorded by the drive, and may also take action on its own accord if the trip point(s) are exceeded, such as slowing down the drive's activity or even shutting it down completely. The exact implementation depends on the manufacturer.

Due to mechanical failure--for example, a case cooling fan that malfunctions--overheating is possible even in a system properly designed to avoid such problems under normal circumstances. This makes temperature monitoring a very useful and important feature for systems were reliability is essential. It is more often found on high-end SCSI units than consumer-grade drives, but it seems likely to me that in the future it will become standard on most drives.

Next: Expanded Remapping and Spare Sectoring

Expanded Remapping and Spare Sectoring

When any hard disk detects a problem with a particular sector on the surface of the drive, it will remap that sector and mark it as "bad" so that it will not be written to again in the future. This ensures that repeat problems with this bad spot will not recur. This process is called *remapping and spare sectoring* and is described in this section on hard disk formatting.

Some drives go a step beyond ordinary remapping. Instead of just remapping the sector where an error is encountered, or where a number of retries were necessary to get the data from the disk, the controller remaps a zone of sectors around the defective location. The logic behind this feature is that if an area of the disk is damaged badly enough to create a bad sector, the problem might not be limited to just that sector, even if the errors are only showing up there *right now*. After all, a single sector on a hard disk is *very* small; it stands to reason that there is a much higher chance that a sector near the defective one will go bad than a random sector somewhere else on the disk.

The "buffer area" that is remapped can include sectors before and after the bad sector, as well as adjacent tracks, thus covering a two-dimensional space centered on the bad disk location. This process occurs transparently to the user. It makes use of spare sectors allocated on the drive for the purpose of remapping.

Western Digital calls this feature *defect margining*. As with the wear leveling feature, defect margining does not appear to have been universally adopted by other hard disk manufacturers. *Unlike* wear leveling, there is not to my knowledge any major drawback to this feature, and it seems in theory to be a very good idea. (I suppose it might require a few extra spare sectors to be placed on the drive, but capacity is very cheap these days.) I do not know why the other drive makers are not also doing this--maybe they *are* doing it but just not publicizing it.

Next: Redundant Arrays of Inexpensive Disks (RAID)

Redundant Arrays of Inexpensive Disks (RAID)

Most of the reliability features and issues discussed in this part of the site relate to making drives themselves more reliable. However, there is only so much you can do to improve the reliability of a single drive without the cost becoming exorbitant. Furthermore, since most people aren't willing to pay for ultra-reliable drives, manufacturers have little incentive to develop them. For those applications where reliability is *paramount*, the quality of no single-drive solution is sufficient. For these situations, many businesses and power users are increasingly turning to the use of multiple drives in a redundant or partially-redundant array configuration. The common term that refers to this technology is *Redundant Arrays of Inexpensive* (or *Independent*) *Disks*, abbreviated *RAID*.

The principle behind RAID is "belt and suspenders": if you store redundant information across multiple disks, then you insulate yourself from disaster in the event that one of the disks fails. If done properly, you also improve performance--sometimes in a substantial way--by allowing the drives to be accessed in parallel. And you can make it so bad drives can be replaced without the system even being taken down.

RAID is a big topic unto itself; there are many different ways that RAID can be implemented; various hardware and software considerations; and many tradeoffs to be considered when implementing a system. I have therefore created a separate area that discusses RAID in detail. Check it out if the subject interests you. RAID is rapidly increasing in popularity, and I believe it will only be a few years before it starts showing up even in high-end home systems.

Next: Hard Disk Warranty and Disaster Recovery Issues

Hard Disk Warranty and Disaster Recovery Issues

If you use computers long enough and often enough, you will eventually will have to deal with a failed drive. It really is only a matter of time and luck. When this occurs with a drive that is reasonably new, you'll want to have it repaired or replaced under warranty. Unfortunately, most people couldn't tell you anything about their hard disk's warranty other than its length--and in some cases they later find out that they were mistaken about even that! The vast majority of hard disk users don't read the "fine print", and many manufacturers don't exactly make it easy to find even if you *want* to read it.

There are a lot of issues related to warranty coverage on hard disks that you need to understand to be sure of your warranty status and what will happen in the event that you need to exercise your warranty rights. Hard disk manufacturers differ drastically in terms of their warranty and service procedures, which is something most people don't think of at the time they buy. But if reliability is important to you, you *must* understand how various manufacturers approach warranty issues before making your purchase. For some people, this matter is more important than all the other quality and performance issues put together.

In this section I take a close look at the warranty issues related to hard drives. This includes a discussion of manufacturer warranty policies, a look at the important matter of OEM and retail drives, and a look at issues involved in returning a drive for repair or replacement. I also discuss data recovery and data security, essential topics to those with very important or confidential data.

Tip: You may also want to check out this related general discussion of warranty and service issues.

Next: Warranty Coverage: Retail, OEM and Stolen Drives

Warranty Coverage: Retail, OEM and Stolen Drives

You can't spend more than a few weeks on technical discussion forums without running into someone who has had The Rude Awakening [tm]. It usually goes something like this:

"I have a drive made by manufacturer 'M' that I bought from retailer 'R'. It died, and since it was within the 3-year warranty period that 'M' offers on their drives, I called them for a replacement. They told me immediately that this was an OEM drive and they could therefore not provide any warranty coverage for it. So now I am stuck, and I have to go buy a new drive even though this one was only 15 months old. I am *really* ticked off!"

(And that's a sanitized version. :^)) The same thing can just as easily happen with a drive in a "PC bought from company 'C'". Welcome to the hard disk "warranty caste system". As far as warranty coverage is concerned, all drives are not always created equal, even if the drives are physically identical.

The reason that this happens is that manufacturers sell drives in two distinct ways: retail and OEM. "OEM" stands for *original equipment manufacturer* and refers (in this context) to a company that builds PCs. Hard drive makers package some of their drives for sale directly to the public, and some for sale in large lots to big companies that make thousands of PCs a month. Due to the very different needs of these two types of customers, the packages are very different. If you are not familiar with the difference between retail and OEM packaging, read this page before continuing.

Aside from the contents of the drive package, the other big difference between OEM and retail hard drives *for some manufacturers* is the warranty provided on the drive. Most drive makers with big OEM contracts sell those drives at a lower price to the OEM because they are *specifically excluding warranty coverage from them*. Nothing comes free in this world, including warranty support. The OEMs save money by not buying warranties on the hard disks they use in their systems. The condition of the deal is that the OEM will provide warranty coverage for the drive. It's important to understand that this is commonly done not just for hard disk drives but also for many other components used in PCs: optical drives, memory, video cards and even CPUs. If OEMs did not do this, the people who buy PCs would be paying for all these individual warranties. Instead, the PC maker provides a warranty that covers the entire system, and the buyer of the system is supposed to go to the OEM for service.

The problem is that many of these parts being packaged for OEMs with no warranty are ending up on the retail market; they are sometimes called *gray market* components. Buyers who do not understand that OEM parts often do not come with warranty coverage purchase them thinking they have the same coverage as retail drives bought in their local computer superstore. They *think* they just got the deal of the century and wonder why other people pay extra for those retail-boxed drives. These are the people who are most susceptible to The Rude Awakening [tm].

The same thing often happens with PC buyers, because of another little problem: the warranties most PC vendors provide for their systems are one or two years, while most drive warranties are three or five years. PC buyers think they can call the drive manufacturer to get the drive replaced if it fails after the PC warranty expires but before the drive warranty does, or if the PC maker goes out of business or is just a pain in the butt to deal with. Well, sometimes you can and sometimes you can't.

While similar in so many ways, the big hard disk manufacturers are very different in their policies towards OEM drive warranty coverage. Some companies intentionally provide a "no questions asked" warranty policy: if they made the drive and it is within the warranty period, the drive is covered. Others will refuse to cover any drives not specifically sold with warranty coverage included. Warranty policies can change, so *you must check before making your purchase to see what the warranty policy is of the manufacturer you are selecting*.

It seems strange that the hard drive makers, who are so competitive and are in a market with relatively low margins, could be so different on this issue. How can the companies that cover all their drives afford to provide warranty coverage on OEM drives when other companies don't do this? Well, there's a simple answer: they can't. Their coverage of all OEM drives means they can't cut their prices on OEM drives nearly to the extent that the companies that refuse to cover OEM drives can. This means *in general* that they are not cost-competitive with those companies and forego some of the big OEM business in order to have simpler warranty policies and to engender the goodwill of the end user base.

As for the companies that refuse to provide end-user warranty coverage on OEM drives: while the frustration of being surprised on this matter is understandable, I think these companies get just a *bit* too much flak on the subject. Is it really reasonable to expect a company to sell a drive at a lower price because no warranty is included, and then expect them still to provide warranty coverage? Most of the people who expect hard disk makers to do this don't expect it from the manufacturers of most other products. (I will say this however: it is in my opinion inexcusable that companies such as IBM do not provide any way for users to tell easily by looking at their drives whether or not they come with warranty coverage. Surely putting some clear notation on the drive would not be that difficult to do. At least, an online serial number warranty status check facility would give hard disk buyers a fighting chance. Unfortunately, the party line on this issue usually seems to be "if you want to be sure you have a warranty, buy a retail drive from an authorized dealer". That's true, but they know that users are buying their drives OEM and they should make it easier to check warranty status. Instead, they are turning a blind eye to this problem.)

OK, now you understand what the issue is. The question then becomes: how do you avoid making a mistake in this regard? Well, if you are buying a drive separately, the simplest way is to either buy a retail boxed drive, or buy a drive from one of the manufacturers that has a "no questions asked" warranty coverage policy. If you are buying a PC, the situation becomes a bit more complicated. If it's a large brand-name PC, the chances are close to 100% that the drive it contains is only covered by the PC maker's warranty--plan for this. However, if you buy a PC from a small local shop, it's possible that they will be using drives bought in small lots that do come with a full retail warranty. *You have to ask*. Even if buying from a big mail-order company, you can sometimes ask to have a drive from a "no questions asked" vendor substituted for a small charge, but again, verify the status of the warranty.

Another issue related to warranty coverage has to do with *stolen drives*. Since hard disks are so small and so expensive, large lots of drives are often the target of thieves who steal the drives and then dump them back on the market for resale. Manufacturers who lose drives this way will often refuse to honor the warranty on such drives. The only protection from this is the first step you should always take anyway: *buy from a reputable dealer with a good reputation*. If you are unsure about a drive you have just bought, contact the drive maker's technical support department and they will probably be able to check the drive's serial number against their database and tell you its status.

Next: Third Party Warranty Support

Third Party Warranty Support

As if the matter of warranty coverage weren't confusing enough, some companies have added to the brew by outsourcing support for some of their drives. This is usually done for older product lines that the company does not want to continue having to provide support for. Checking with the technical support department will usually let you find out if a drive you own has support provided for it by a third party. I am not aware of any companies presently doing this for newer drives; it is usually for obsolete hardware (which I suppose wouldn't be covered by warranty any more, making this more of a general service and support issue; it's still a good thing to know about if you plan to hold on to your drive past the warranty period.)

Another case where this issue comes into play is if a company goes out of business. Hard disk manufacturers do go under from time to time. Usually they don't just fold and "disappear" but rather are bought out by another company. When this happens, the buying company normally inherits the warranty and support load for the drives made by the company they are buying. However, sometimes these are outsourced to still another company. Also, a buyout can result in any number of changes relative to the warranty status and policies of the old company, so be sure to check into the matter if this happens to a company whose drives you are using.

Next: Warranty Replacement Policies

Warranty Replacement Policies

If companies vary in their willingness to provide warranty coverage to different types of drives, they vary almost as much in the way they handle warranty claims. Like coverage issues, these often come as a surprise to hard disk users who only find out about them after disaster strikes. The way to protect yourself is to be informed: read the fine print. Before purchasing a hard disk, or selecting a hard disk brand for inclusion in a new system, do your research. Visit the web sites of the hard drive manufacturers that you are interested in and read about their warranty policies. If there's anything you don't understand, call the company or write them an email and ask. They will usually be happy to explain these policies (particularly if they understand that you are making a pre-sale call!)

When it comes to warranty replacement there are two main policies to be considered. The first is the matter of *what* the company will be replacing the defective drive with. The second is *how* the company will be doing the replacement.

When you send a hard disk in on a warranty claim, the drive you get back is never the one you sent in. (This is a good thing, because if you got back the same drive, you'd be twiddling your thumbs for weeks waiting for it to be repaired.) In fact, many defective drives are in fact never repaired at all, since it is so expensive to do so. Instead, the company sends you a replacement drive, and this is one of the policy issues that distinguishes companies: some send *new* drives and others send *refurbished* drives. Some send both (not to the same person, of course. :^)) Many people have good luck with refurbished drives, but some others don't. Some people don't feel *confident* about refurbished drives, which makes it hard for them to use these units even if they are perfectly fine. All in all, it seems pretty obvious which is the more optimal choice here.

Tip: If you have an older drive that fails near the end of its warranty period, resist the temptation to "not bother" getting it replaced because it's "too small and too slow anyway". Two and a half years after your drive is made, the manufacturer sometimes won't have any of the same type of drive to send as a replacement on a warranty claim, and you might get a newer or faster drive as a replacement. This happens more than you might think!

The second issue is how the replacement drive will get to you; most companies offer a choice of two options. The first is standard replacement, where you send the drive back to the manufacturer, and then they send a replacement back to you. This is simple, but time-consuming; you could be without a drive for a week or more. Recognizing that many of their customers could not tolerate being down for such a long period of time, most manufacturers now offer an *advanced replacement* option. The company will send out the replacement drive--usually by second-day air delivery--before receiving back the defective drive. You have to give them a credit card number as surety that you will in fact send back the defective drive.

If reliability and up-time are important to you *at all*, I would strongly advise against buying a hard disk from any manufacturer that will not offer advanced replacement. I have read testimonials from people who had to go out and buy a new drive when their current one failed because even though it was under warranty, they would have been waiting too long for a warranty replacement and simply couldn't afford the down time. If your application or business are such that you cannot afford to have your system down even for a day or two, then your needs exceed what hard disk manufacturer warranties can provide. You must look beyond a single-drive solution and consider a fault-tolerant storage subsystem such as a RAID array. If RAID is too expensive, then a cheap but much less optimal solution is to just buy a spare drive to keep around in case the one you are using fails, so you can restore from your backup immediately. If even that is too expensive then, well, avoiding downtime is simply not a top priority. Everything has its costs.

Next: Obtaining Warranty Service

Obtaining Warranty Service

If you suspect that the drive you are using has a problem, then you may need to contact the company to obtain warranty service on the drive. Instructions for doing this are easily found on the manufacturer's web site under a heading typically called "Support". You will usually be asked to call the company's technical support department for instructions and to obtain an RMA (return of materials authorization) number. Some companies may let you get an RMA number using an automated system.

Warning: Never send a drive, or in fact *anything*, back to any company without an RMA number (unless you are instructed to do so, which is rare). At best, the item will be refused and sent back to you. At worst, it will get lost in their system because they will have no way of tracking it. Never underestimate the odds of unidentified hardware getting lost in a large company!

Companies vary in their policies regarding failure determination, and the conditions you must satisfy before they will accept a drive back under warranty. Some will make you use their diagnostic software to check out the drive first; others will take your word on a claim that the drive is misbehaving (reasoning that nobody would put themselves through the hassle of sending back a drive without good cause!) Policies regarding paperwork and timing of various actions also vary greatly from one drive maker to another.

For more general information on repairs and returns, see this section.

Next: Data Security

Data Security

Most people have personal information on their hard disks, and some worry about what happens to this data when they send a drive in to a manufacturer for warranty replacement (or to a data recovery company for that matter). If your hard disk fails, most of the time almost all of the data is still intact on the platters. A burned-out spindle motor or flaky actuator has no impact on the data stored on the disk. Even a head crash or other defect related to the data surfaces will result in little of the actual data being removed from the drive.

If you are worried about the data on a disk that is totally dead, there is really nothing you can do to get the data off the drive that won't also void the warranty of the drive. If you are worried about the data on a drive that is acting "funny" but has not totally failed, you can wipe the data off the drive before sending it back to the manufacturer by using a zero-fill utility. This will for all intents and purposes eliminate all data from the drive. There have been "James Bond" like reports of experts retrieving data after a zero-fill has been performed, due to trace magnetic fields on the disk; even if true, someone would have to be *remarkably* motivated to even bother with such an exercise. Unless you're a master spy, the contents of your hard disk probably aren't nearly interesting enough for anyone with the skill to do something like this to even be bothered spending their time. And most of the *real* "cloak and dagger" stories about genuises retrieving data from disks that have had their data overwritten with various data patterns a dozen times or been blasted with shotguns are probably apocryphal. It is true that data recovery companies can do amazing things, but they aren't miracle workers. :^)

You should keep in mind that hard disk technicians are professionals. A hard disk company cannot afford to have a reputation for snooping around in their customers' files--and a data recovery company can afford such a black eye even less. I personally have never heard of even one instance of a hard disk manufacturer or data recovery company going through someone's "stuff" and causing problems, so either it isn't happening, or nobody is talking about it. Remember that most of the data that is so important to you, is important *only* to you. This is especially true of most PCs used for personal purposes, as opposed to businesses.

All of the above said, the bottom line of data security is that it is only truly secure if it never leaves your hands. If you're really worried about data security, if you truly can't afford *any* chance of the data on the drive falling into the wrong hands because it contains the entire company's payroll information or your almost-complete plans for a cold fusion reactor, then you have an alternative: don't send it in for warranty service. Just buy a new drive. After all, hard disks only cost a few hundred dollars, and if the data is *that* important, it's worth eating the cost of a hard disk and just buying a new one. This is in fact exactly what some businesses do. The old drive should be destroyed or just put in the back of a safe somewhere. If you have access to a smelter, well, that even worked on the Terminator. :^)

Next: Data Recovery

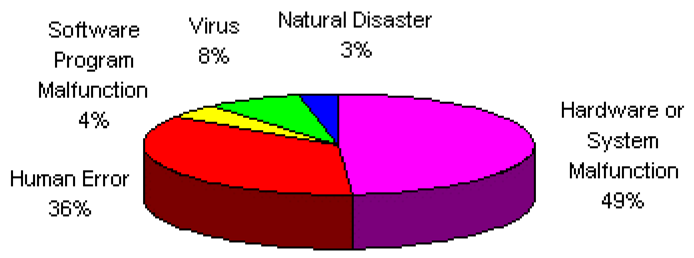
Data Recovery

Hard disks differ from virtually every other component in the PC in one very critical respect: when you lose a hard disk you lose more than just hardware--you lose software and data as well. The hard disk can be replaced, but the data on it often cannot (and it certainly isn't covered by your warranty). This is why annoying people like me always harp on users to back up their data.

If for some reason you fail to heed my sage advice :^) and do not back up your important data, it is still sometimes possible to recover it in the event of a hard disk failure. There are data recovery services that specialize in restoring lost data from hard disks that have either failed or been corrupted due to software problems (accidental formatting, viruses, etc.)

Some of these services are pretty amazing--many of them have a very good success rate and can resurrect data even when it seems like it would be impossible to the casual observer. They use specialized equipment and software, and typically even have mini clean rooms that enable them to take the cover off the drive and access the platters directly. All this wizardry comes at a cost, of course. A full recovery usually *starts* at a few hundred dollars and proceeds from there. The cost is high for three reasons: the equipment is expensive, the people are highly skilled, and the company knows how valuable the data is or you wouldn't be talking to them. Compared to recreating a year's worth of data, $2,000 for a recovery is a bargain. Compared to doing routine backups, it's a colossal waste of money.

One company well-known for performing data recovery is *Ontrack*.



This graph shows the most common causes of data loss requiring the use of data recovery services, based on actual Ontrack recoveries in the 1995-1996 timeframe.

Note: The use of RAID arrays will reduce the likelihood that data recovery will be required, but can complicate matters if for some reason it *is* required. For information on the implications of RAID on data recovery, see this section.

Note: Even though drives have labels and stickers on them saying that if the drive is opened the warranty will be voided, this does not apply to professional data recovery. These companies have arrangements with the major hard disk manufacturers to allow them to work on the drives without affecting the drive's warranty status. In some cases the recovery company may be required to send a letter to the drive manufacturer certifying that your particular drive had recovery performed upon it.

Next: Redundant Arrays of Inexpensive Disks (RAID)

Redundant Arrays of Inexpensive Disks (RAID)

"KILLS - BUGS - DEAD!"
-- TV commercial for RAID bug spray

There are many applications, particularly in a business environment, where there are needs beyond what can be fulfilled by a single hard disk, regardless of its size, performance or quality level. Many businesses can't afford to have their systems go down for even an hour in the event of a disk failure; they need large storage subsystems with capacities in the terabytes; and they want to be able to insulate themselves from hardware failures to any extent possible. Some people working with multimedia files need fast data transfer exceeding what current drives can deliver, without spending a fortune on specialty drives. These situations require that the traditional "one hard disk per system" model be set aside and a new system employed. This technique is called *Redundant Arrays of Inexpensive Disks* or *RAID*. ("Inexpensive" is sometimes replaced with "Independent", but the former term is the one that was used when the term "RAID" was first coined by the researchers at the University of California at Berkeley, who first investigated the use of multiple-drive arrays in 1987.)

The fundamental principle behind RAID is the use of multiple hard disk drives in an array that behaves in most respects like a single large, fast one. There are a number of ways that this can be done, depending on the needs of the application, but in every case the use of multiple drives allows the resulting storage subsystem to exceed the capacity, data security, and performance of the drives that make up the system, to one extent or another. The tradeoffs--remember, there's no free lunch--are usually in cost and complexity.

Originally, RAID was almost exclusively the province of high-end business applications, due to the high cost of the hardware required. This has changed in recent years, and as "power users" of all sorts clamor for improved performance and better up-time, RAID is making its way from the "upper echelons" down to the mainstream. The recent proliferation of inexpensive RAID controllers that work with consumer-grade IDE/ATA drives--as opposed to expensive SCSI units--has increased interest in RAID dramatically. This trend will probably continue. I predict that more and more motherboard manufacturers will begin offering support for the feature on their boards, and within a couple of years PC builders will start to offer systems with inexpensive RAID setups as standard configurations. This interest, combined with my long-time interest in this technology, is the reason for my recent expansion of the RAID coverage on this site from one page to 80. :^)

Unfortunately, RAID in the computer context doesn't really kill bugs dead. It can, if properly implemented, "kill down-time dead", which is still pretty good. :^)

Next: Why Use RAID? Benefits and Costs, Tradeoffs and Limitations

Why Use RAID? Benefits and Costs, Tradeoffs and Limitations

RAID offers many advantages over the use of single hard disks, but it is clearly not for everyone. The potential for increased capacity, performance and reliability are attractive, but they come with real costs. Nothing in life is free. In this section I take an overview look at RAID, to help explain its benefits, costs, tradeoffs and limitations. This should give you a better idea if RAID is for you, and help you to understand what RAID can do--and what it can't do.

As you read on, it's essential to keep in mind that with RAID, it's definitely the case that "the devil is in the details". Most common blanket statements made about RAID like "RAID improves availability" or "RAID is for companies that need fast database service" or "RAID level 5 is better than RAID level 0" are only true at best part of the time. In almost every case, *it depends*. Usually, what RAID is and what it does for you depends on what type you choose and how you implement and manage it. For example, for some applications RAID 5 *is* better than RAID 0; for others, RAID 0 is vastly superior to RAID 5! There are situations where a RAID design, hardware and software that would normally result in high reliability could result instead in disaster if they are not properly controlled.

Next: RAID Benefits

RAID Benefits

Alright, let's take a look at the good stuff first. :^) RAID really does offer a wealth of significant advantages that would be attractive to almost any serious PC user. (Unfortunately, there are still those pesky costs, tradeoffs and limitations to be dealt with... :^)) The degree that you realize the various benefits below does depend on the exact type of RAID that is set up and how you do it, but you are always going to get some combination of the following:

- **Higher Data Security:** Through the use of redundancy, most RAID levels provide protection for the data stored on the array. This means that the data on the array can withstand even the complete failure of one hard disk (or sometimes more) without any data loss, and without requiring any data to be restored from backup. This security feature is a key benefit of RAID and probably the aspect that drives the creation of more RAID arrays than any other. All RAID levels provide some degree of data protection, depending on the exact implementation, *except* RAID level 0.
- **Fault Tolerance:** RAID implementations that include redundancy provide a much more reliable overall storage subsystem than can be achieved by a single disk. This means there is a lower chance of the storage subsystem as a whole failing due to hardware failures. (At the same time though, the added hardware used in RAID means the chances of having a hardware problem of some sort *with an individual component*, even if it doesn't take down the storage subsystem, is increased; see this full discussion of RAID reliability for more.)
- **Improved Availability:** Availability refers to access to data. Good RAID systems improve availability both by providing fault tolerance and by providing special features that allow for recovery from hardware faults without disruption. See the discussion of RAID reliability and also this discussion of advanced RAID features.
- **Increased, Integrated Capacity:** By turning a number of smaller drives into a larger array, you add their capacity together (though a percentage of total capacity is lost to overhead or redundancy in most implementations). This facilitates applications that require large amounts of contiguous disk space, and also makes disk space management simpler. Let's suppose you need 300 GB of space for a large database. Unfortunately, no hard disk manufacturer makes a drive nearly that large. You could put five 72 GB drives into the system, but then you'd have to find some way to split the database into five pieces, and you'd be stuck with trying to remember what was were. Instead, you could set up a RAID 0 array containing those five 72 GB hard disks; this will appear to the operating system as a single, 360 GB hard disk! All RAID implementations provide this "combining" benefit, though the ones that include redundancy of course "waste" some of the space on that redundant information.
- **Improved Performance:** Last, but certainly not least, RAID systems improve performance by allowing the controller to exploit the capabilities of multiple hard disks to get around performance-limiting mechanical issues that plague individual hard disks. Different RAID implementations improve performance in different ways and to

different degrees, but all improve it in some way. See this full discussion of RAID performance issues for more.

Next: RAID Costs

RAID Costs

The many benefits of RAID do not come without some costs. (This is self-evident simply by the fact that most PCs do not use RAID arrays.) Most of the attention when it comes to RAID costs is paid to the hardware, but there are other costs that can dwarf the dollars (pounds, lira, shekels) paid for hardware. It's impossible to say exactly what a RAID implementation will cost because they can vary from simple setups costing only a couple of hundred dollars, to enormous arrays that cost as much as a small house.

When considering if RAID is right for you, don't forget to add in the costs in all of these major categories, where relevant:

- **Planning and Design:** For a decent-sized RAID system, you must allow some resources for planning what type of RAID will be implemented, deciding on array size, choosing hardware, and so on.
- **Hardware:** Hardware costs include hard disks, enclosures, power supplies, power protection and possibly a hardware RAID controller. While "high-end" RAID requires all of the above, "economy" RAID implementations can be done within a regular PC case, using the existing PC power supply and protection, and either an inexpensive hardware controller or no controller. Most people think immediately of the cost of the hard disks in a RAID system when they think about expenses associated with RAID. This is only true to a point, and it depends on what RAID level is being implemented. Remember that if you need "X" amount of space for your files, you're going to have to buy "X" worth of hard disks regardless. RAID only costs extra in this regard to the extent that additional drives must be purchased for redundancy. If you are implementing RAID 0, there is no additional disk drive cost unless you are buying extra-large drives and you don't really need the full capacity of the array.
- **Software:** Most hardware RAID solutions come with all the software you need to operate them. If you are doing software RAID however, you need an operating system such as Windows NT or Windows 2000 that provides this functionality. Of course, you may already be planning to use an operating system with software RAID support anyway...
- **Setup and Training:** Simple RAID systems in individual PCs don't really require training, and are easy to set up. Larger RAID systems can require many hours to set up and configure, and training of IS professionals may be required.
- **Maintenance:** Enterprise-class RAID systems require ongoing maintenance if they are to continue to provide the organization with high availability and performance.

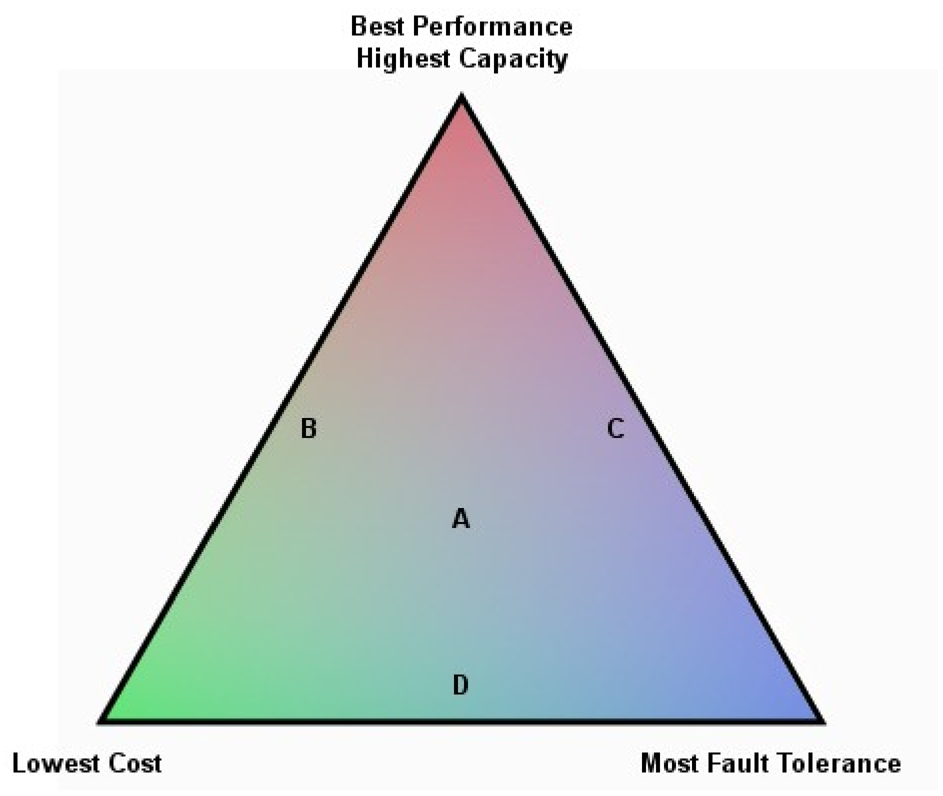
Obviously, the costs are highest for businesses implementing large arrays. For these companies, however, these costs must be compared to the costs of data loss, data recovery and interruption of availability that would result if RAID were not used. For many companies, the *entire* cost of a RAID setup pays for itself the first time it prevents their enterprise system from having to be taken down for half a day to deal with a hardware failure.

Next: RAID Tradeoffs

RAID Tradeoffs

"Fast, cheap, good: choose two"

The phrase above is a famous one in the world of computing, and elsewhere: it's a good rule of thumb to keep in mind when making many purchase decisions. It describes well what I call the "essential tradeoff triangle" that applies to thousands of different technologies and devices, from PCs to race car engines to kitchen blenders: you can easily get something that is inexpensive and fast, but at the cost of quality; or something high quality and fast, but it won't be cheap; or something cheap and high quality, but of lower performance. In RAID, the "good" attribute refers specifically to reliability concerns, and in particular, fault tolerance. The "fast" attribute refers to either performance or capacity (it's not speed, but it does trade off against fault tolerance and cost). In general, for a given price point, the performance improvement of a RAID array trades off to some extent with the array's redundancy and data security potential. If you want to improve both, you have to pay more. The same applies to capacity: high capacity trades off against economy and fault tolerance. (These are of course rules of thumb: the attributes don't *strictly* trade off against one another in a linear way; I'm just talking about general trends.)



This triangle shows how performance (or capacity), cost and fault tolerance trade off in the RAID world. At each of the corners, one of the attributes is maximized at the expense of the other two. Point "A" represents a balance between the three: not great in any of them, not poor in any either. Points "B",
"C" and "D" represent doing well in two of the attributes at the expense of the third: these are the "choose two" points that were mentioned earlier.

What it all really comes down to is: what are your priorities, and what are you willing to spend? If high data availability and fault tolerance are not important, you can get a high-performance RAID array for relatively little money; if peak performance is not all that important, you can get a very reliable system without spending a fortune. It's only when you need both that things get expensive. The tradeoffs between performance, cost and reliability are most readily seen when contrasting the different RAID levels. Some emphasize performance over fault tolerance, others do the opposite, while some try to balance the attributes. Some cost more than others. Some offer very high fault tolerance but relatively low capacity. In no case, however, is there a "free lunch". Cheap RAID solutions are limited in their ability to protect your data or improve performance; high-end RAID implementations that provide very high performance and very high data reliability are also quite expensive.

Next: RAID Limitations

RAID Limitations

I think it's important to point out that while RAID can do many things to improve the reliability and performance of the storage subsystem of a PC, there are many things it *can't* do. One very *dangerous* phenomenon that is sometimes exhibited by those who are using RAID systems is that they develop an "invulnerability complex". Much like those strange people who think being behind the wheel of an SUV means they can drive up the side of Mt. Everest and therefore zip around in bad weather at 60 mph, some PC users who have RAID systems with redundancy seem to think that the data protection means they are "all covered". They stop worrying as much about maintenance and backups, just like those SUV owners think that four-wheel drive has obviated safe and cautious driving.

While the redundancy that is built into RAID definitely gives you some very important protection, it doesn't turn your PC into Superman (er... "SuperPC"). There are still sources of failure that can and *do* strike RAID systems. If you examine this list of risks to your data, one thing will become clear immediately: RAID not only doesn't protect against *all* of them, it doesn't even protect against *most* of them! RAID won't help you if a virus wipes out your system, or if a disgruntled employee decides to delete all your files, or if lightning hits your building and causes a fire in your computer room. And even in the area of hardware failure, RAID is not foolproof. There have been cases where more than one drive has failed at the same time, causing the entire array to fail and necessitating expensive data recovery. It's not common, but then, neither are lightning strikes. This is one reason why backups remain critical even when RAID is used.

And of course, a final note on RAID 0, which has become very popular amongst those who strive for improved performance and "don't care" about enhanced reliability: RAID 0 should really be called "AID", because there is no redundancy at all, and therefore, you have no fault tolerance, and no protection against data loss. In fact, the reliability of a RAID 0 system is *much worse* than that of a regular hard disk: see here. Also, recovery from a failure of any RAID 0 hard disk is extremely difficult. Keep your backups current, and remember the risks you take if you go with RAID 0--most don't until disaster strikes.

Next: Should You Use RAID?

Should You Use RAID?

I don't know. You tell me. :^)

How easy it is to answer this question depends on who you are and what you are trying to do. The only way to answer the question is to fully explore the issue, weigh the costs against the benefits, compare the costs to your budget and decide what your priorities are. Do this honestly and the question will answer itself.

That said, I won't cop out completely. :^) Here are some very broad guidelines:

- **Business Servers:** In this author's opinion, all but the smallest businesses should be running their critical data on some sort of RAID system. Data is so important, and interruptions can be so crippling to most businesses, that the costs are usually worth it. Even an inexpensive, small RAID setup is better than nothing, and if budget is very tight, not all of the company's data has to reside on the array.
- **Workstations:** For those individuals who are doing intensive work such as video file editing, graphical design, CAD/CAM, and the like should consider a RAID array. RAID 0 will provide the improved performance needed in many of these applications. (RAID 10 is superior due to its redundancy, but the requirement for four drives makes it expensive and space-consuming; if the RAID 0 array is backed up each night then that's usually quite acceptable for a workstation.)
- **Regular PCs:** Most "regular PC users" do not need RAID, and the extra cost of one or more additional hard drives is usually not justified. Most individuals who set up RAID on regular PCs cannot afford hardware RAID and SCSI drives, so they use software RAID or inexpensive IDE/ATA RAID controllers. They are typically setting up RAID solely for performance reasons, and choose RAID 0. Unfortunately, RAID 0 just doesn't improve performance all that much for the way typical PCs are used; I often see gamers setting up RAID 0 systems when most games will take little advantage of it. Meanwhile, the RAID 0 array puts all of the user's data in jeopardy.

Next: RAID Concepts and Issues

RAID Concepts and Issues

Most publications discussing RAID start with describing the different RAID levels. As you probably know, I like to be different. :^) Well, not just for the *sake* of being different, but where it makes sense to do so. In this case, I think it makes sense to understand the different concepts and issues that underlie RAID before we go to discussing the different levels. I hope that doing this will not only give you the foundation necessary to understand various design issues and concepts in RAID applications, but will frame the various issues that differentiate RAID levels and make them easier to understand as well.

In this section I start off by describing the general concepts that define what RAID is about. Then, since performance and reliability are the two "classes" of reasons why PC users choose RAID, I discuss issues related to those two subjects in some detail.

Next: General RAID Concepts

General RAID Concepts

In this section I will start "at the beginning", describing the key concepts that define the fundamental characteristics of RAID. These are the concepts that describe how RAID works, how arrays are set up, and how different RAID levels work to improve reliability and performance. Understanding these concepts provides the foundation for our subsequent discussions of performance issues, reliability concerns, and the various RAID levels.

Next: Physical and Logical Arrays and Drives

Physical and Logical Arrays and Drives

The fundamental structure of RAID is the *array*. An array is a collection of drives that is configured, formatted and managed in a particular way. The number of drives in the array, and the way that data is split between them, is what determines the RAID level, the capacity of the array, and its overall performance and data protection characteristics. Deciding what types of arrays to set up, and how to configure them, is the first thing you do when setting up a RAID implementation.

Understanding arrays and drives can get awfully confusing because high-end RAID systems allow for such complexity in how they are arranged. To get a better handle on this, let's look at the "hierarchy" in which data in a RAID system is organized:

1. **Physical Drives:** The physical, actual hard disks that comprise the array are the "building blocks" of all data storage under RAID.
2. **Physical Arrays:** One or more physical drives are collected together to form a *physical array*. Most simple RAID setups use just one physical array, but some complex ones can have two or more physical arrays.
3. **Logical Arrays:** Logical arrays are formed by splitting or combining physical arrays. Typically, one logical array corresponds to one physical array. However, it is possible to set up a logical array that includes multiple physical arrays (typically used to allow multiple RAID levels). It is also possible to set up two entirely different logical arrays from a single physical, as described here.
4. **Logical Drives:** One or more logical drives are formed from one logical array (much the same way they would normally be formed from one physical drive in a non-RAID system). These appear to the operating system as if they were regular disk volumes, and are treated

accordingly, with the RAID controller managing the array(s) that underlie them.

Notice how we start with drives, go to arrays, and then end up back talking about drives. Isn't that nice and confusing? :^) This occurs so that the array management is "hidden" from the operating system. (Under software RAID a special part of the operating system does this management, hiding the arrays from the rest of the software.) The *really* confusing thing is that the terms above, which are similar-sounding enough, are often used *loosely*. Since most RAID setups consist of one physical array made into one logical array made into one logical drive, the term "logical array" is sometimes used interchangeably with "physical array" or "logical drive". Ugh.

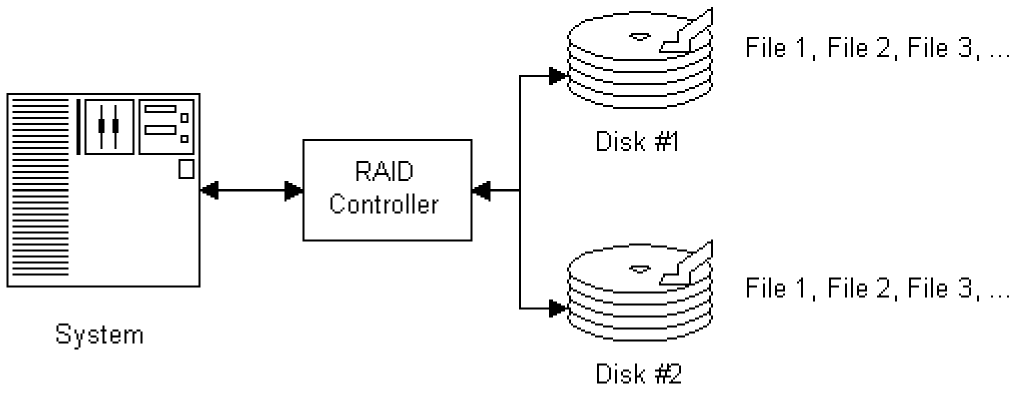
As you can see, better RAID controllers give you all the "rope" you need to hang yourself when it comes to defining arrays. How many logical arrays and drives you should define depends entirely on the RAID level(s) you are going to be used. But in general, when defining arrays and drives, you should always keep in mind the "KISS" rule--keep it simple. For most reasonably simple applications, creating a single logical array for each physical array is the best way to go. If you need multiple logical arrays for special applications, using separate physical arrays for them may make management simpler than having two arrays sharing physical arrays and therefore, sharing physical drives.

Next: Mirroring

Mirroring

Mirroring is one of the two data redundancy techniques used in RAID (the other being parity). In a RAID system using mirroring, all data in the system is written simultaneously to *two* hard disks instead of one; thus the "mirror" concept. The principle behind mirroring is that this 100% data redundancy provides full protection against the failure of either of the disks containing the duplicated data. Mirroring setups always require an even number of drives for obvious reasons.

The chief advantage of mirroring is that it provides not only complete redundancy of data, but also reasonably fast recovery from a disk failure. Since all the data is on the second drive, it is ready to use if the first one fails. Mirroring also improves some forms of read performance (though it actually hurts write performance.) The chief disadvantage of RAID 1 is expense: that data duplication means half the space in the RAID is "wasted" so you must buy twice the capacity that you want to end up with in the array. Performance is also not as good as some RAID levels.



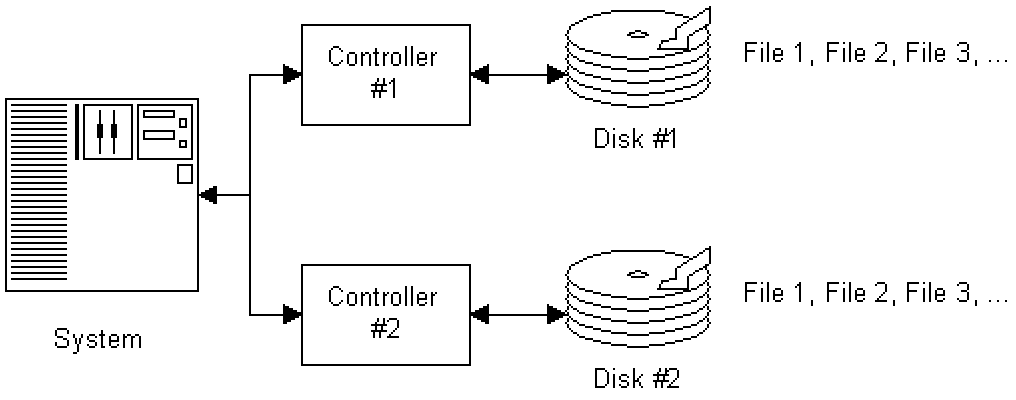
Block diagram of a RAID mirroring configuration. The RAID controller duplicates the same information onto each of two hard disks. Note that the RAID controller is represented as a "logical black box" since its functions can be implemented in software, or several different types of hardware (integrated controller, bus-based add-in card, stand-alone RAID hardware.)

Mirroring is used in RAID 1, as well as multiple-level RAID involving RAID 1 (RAID 01 or RAID 10). It is related in concept to duplexing. Very high-end mirroring solutions even include such fancy technologies as *remote mirroring*, where data is configured in a RAID 1 array with the pairs split between physical locations to protect against physical disaster! You won't typically find support for anything that fancy in a PC RAID card. :^)

Next: Duplexing

Duplexing

Duplexing is an extension of mirroring that is based on the same principle as that technique. Like in mirroring, all data is duplicated onto two distinct physical hard drives. Duplexing goes one step beyond mirroring, however, in that it also duplicates the hardware that controls the two hard drives (or sets of hard drives). So if you were doing mirroring on two hard disks, they would both be connected to a single host adapter or RAID controller. If you were doing duplexing, one of the drives would be connected to one adapter and the other to a second adapter.



Block diagram of a RAID duplexing configuration. Two controllers are used to send the same information to two different hard disks. The controllers are often regular host adapters or disk controllers with the mirroring done by the system. Contrast this diagram with the one for straight mirroring.

Duplexing is superior to mirroring in terms of availability because it provides the same protection against drive failure that mirroring does, but also protects against the failure of either of the controllers. It also costs more than mirroring because you are duplicating more hardware. Duplexing is one option when implementing RAID 1 (though most people think of RAID 1 as only being mirroring, and certainly, mirroring is much more commonly implemented than duplexing.)

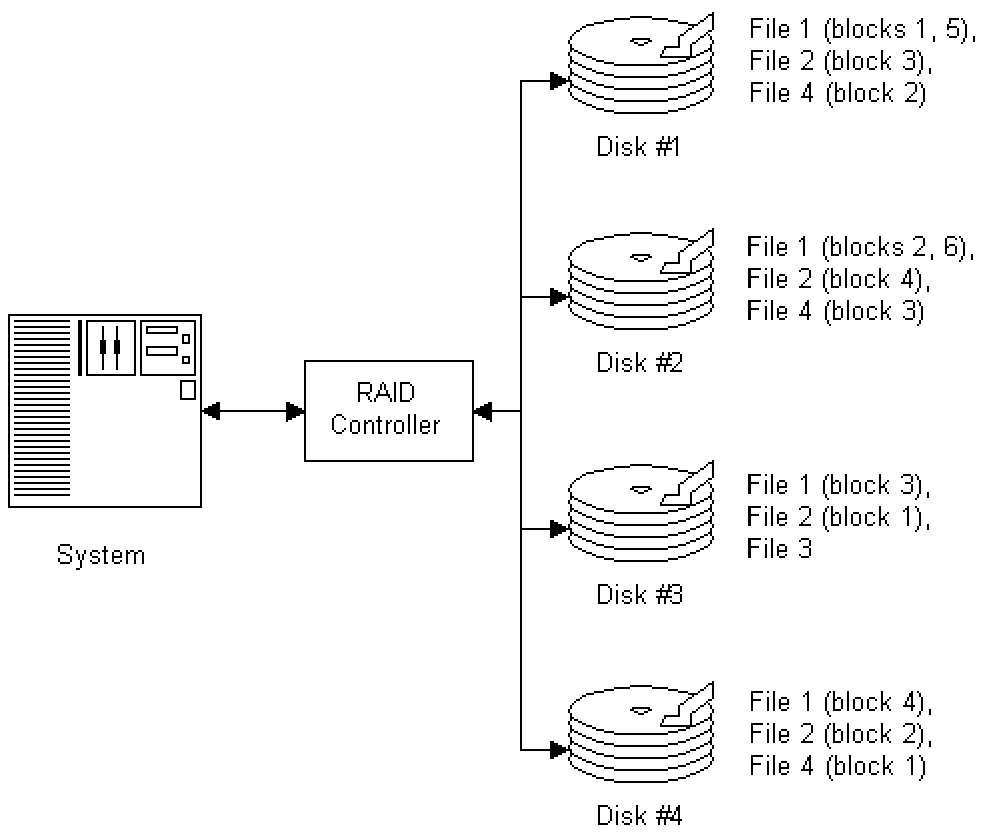
Since hardware RAID is typically set up under the assumption that the RAID controller will handle all the drives in the array, duplexing is not supported as an option in most PC hardware RAID solutions--even fairly high-end ones. Duplexing is more often found in software RAID solutions managed by the operating system, where the operating system is running the RAID implementation at a high level and can easily split the data between the host adapters. (There are hardware RAID duplexing solutions but usually only on very expensive external RAID boxes.)

Next: Striping

Striping

The main performance-limiting issues with disk storage relate to the slow mechanical components that are used for positioning and transferring data. Since a RAID array has many drives in it, an opportunity presents itself to improve performance by using the hardware in all these drives *in parallel*. For example, if we need to read a large file, instead of pulling it all from a single hard disk, it is much faster to chop it up into pieces, store some of the pieces on each of the drives in an array, and then use *all* the disks to read back the file when needed. This technique is called *striping*, after the pattern that might be visible if you could see these "chopped up pieces" on the various drives with a different color used for each file. It is similar in concept to the memory performance-enhancing technique called interleaving.

Striping can be done at the byte level, or in blocks. Byte-level striping means that the file is broken into "byte-sized pieces" (hee hee, sorry about that, I just couldn't resist. ;^)) The first byte of the file is sent to the first drive, then the second to the second drive, and so on. (See the discussion of RAID level 3 for more on byte-level striping.) Sometimes byte-level striping is done as a sector of 512 bytes. Block-level striping means that each file is split into blocks of a certain size and those are distributed to the various drives. The size of the blocks used is also called the *stripe size* (or *block size*, or several other names), and can be selected from a variety of choices when the array is set up; see here for more details.



Block diagram of a RAID striping configuration. One controller (which again can be hardware or software) splits files into blocks or bytes and distributes them across several hard disks. The block size determines how many "pieces" files
will be split into. In this example, the first block of file 1 is sent to disk #1, then the
second block to disk #2, etc. When all four disks have one block of file 1, the fifth block goes back to disk #1, and this continues until the file is completed. Note
that file 3 is only on one disk; this means it was smaller than the block size in this case.

Striping is used in the implementation of most of the basic, single RAID levels (and by extension, any multiple RAID levels that use those single RAID levels). However, the actual way striping is set up, and how it is used, varies greatly from level to level. RAID 0 uses block-level striping without parity; RAID 3 and RAID 7 use byte-level striping with parity; and RAID 4, RAID 5 and RAID 6 use block-level striping with parity. Note the distinction between striping with and without parity: striping by itself involves no redundancy, and therefore, *provides no data protection*. Also see the discussion of RAID 2 for a look at the oddball bit-level striping with ECC defined by that RAID type.

Note: Some companies use the term "spanning" when they really mean striping. Spanning really normally refers to JBOD.

Next: Parity

Parity

Mirroring is a data redundancy technique used by some RAID levels, in particular RAID level 1, to provide data protection on a RAID array. While mirroring has some advantages and is well-suited for certain RAID implementations, it also has some limitations. It has a high overhead cost, because fully 50% of the drives in the array are reserved for duplicate data; and it doesn't improve performance as much as data striping does for many applications. For this reason, a different way of protecting data is provided as an alternate to mirroring. It involves the use of *parity information*, which is redundancy information calculated from the actual data values.

You may have heard the term "parity" before, used in the context of system memory error detection; in fact, the parity used in RAID is very similar in concept to parity RAM. The principle behind parity is simple: take "N" pieces of data, and from them, compute an extra piece of data. Take the "N+1" pieces of data and store them on "N+1" drives. If you lose any *one* of the "N+1" pieces of data, you can recreate it from the "N" that remain, regardless of which piece is lost. Parity protection is used with striping, and the "N" pieces of data are typically the blocks or bytes distributed across the drives in the array. The parity information can either be stored on a separate, dedicated drive, or be mixed with the data across all the drives in the array.

The parity calculation is typically performed using a logical operation called "exclusive OR" or "XOR". As you may know, the "OR" logical operator is "true" (1) if either of its operands is true, and false (0) if neither is true. The exclusive OR operator is "true" if *and only if* one of its operands is true; it differs from "OR" in that if both operands are true, "XOR" is false. This truth table for the two operators will illustrate:

Input		**Output**	
#1	**#2**	**"OR"**	**"XOR"**
0	0	**0**	**0**
0	1	**1**	**1**
1	0	**1**	**1**
1	1	**1**	**0**

Uh huh. So what, right? Well, the interesting thing about "XOR" is that it is a logical operation that if performed twice in a row, "undoes itself". If you calculate "A XOR B" and then take that result and do another "XOR B" on it, you get back A, the value you started with. That is to say, "A XOR B XOR B = A". This property is exploited for parity calculation under RAID. If we have four data elements, D1, D2, D3 and D4, we can calculate the parity data, "DP" as "D1 XOR D2 XOR D3 XOR D4". Then, if we know any four of D1, D2, D3, D4 and DP, we can XOR those four together and it will yield the missing element.

Let's take an example to show how this works; you can do this yourself easily on a sheet of paper. Suppose we have the following four bytes of data: D1=10100101, D2=11110000, D3=00111100, and D4=10111001. We can "XOR" them together as follows, one step at a time:

D1 XOR D2 XOR D3 XOR D4
= ((D1 XOR D2) XOR D3) XOR D4
= ((10100101 XOR 11110000) XOR 00111100) XOR 10111001
= (01010101.XOR 00111100) XOR 10111001
= 01101001 XOR 10111001
= 11010000

So "11010000" becomes the parity byte, DP. Now let's say we store these five values on five hard disks, and hard disk #3, containing value "00111100", goes el-muncho. We can retrieve the missing byte simply by XOR'ing together the other three original data pieces, and the parity byte we calculated earlier, as so:

D1 XOR D2 XOR D4 XOR DP
= ((D1 XOR D2) XOR D4) XOR DP
= ((10100101 XOR 11110000) XOR 10111001) XOR 11010000
= (01010101 XOR 10111001) XOR 11010000
= 11101100 XOR 11010000
= 00111100

Which is D3, the missing value. Pretty neat, huh? :^) This operation can be done on any number of bits, incidentally; I just used eight bits for simplicity. It's also a very simple binary calculation--which is a good thing, because it has to be done for every bit stored in a parity-enabled RAID array.

Compared to mirroring, parity (used with striping) has some advantages and disadvantages. The most obvious advantage is that parity protects data against any single drive in the array failing without requiring the 50% "waste" of mirroring; only one of the "N+1" drives contains redundancy information. (The overhead of parity is equal to (100/N)% where N is the total number of drives in the array.) Striping with parity also allows you to take advantage of the performance advantages of striping. The chief disadvantages of striping with parity relate to complexity: all those parity bytes have to be computed--millions of them per second!--and that takes computing power. This means a hardware controller that performs these calculations is required for high performance--if you do *software* RAID with striping and parity the system CPU will be dragged down doing all these computations. Also, while you can recover from a lost drive under parity, the missing data all has to be rebuilt, which has its own complications; recovering from a lost mirrored drive is comparatively simple.

All of the RAID levels from RAID 3 to RAID 7 use parity; the most popular of these today is RAID 5. RAID 2 uses a concept similar to parity but not exactly the same.

Next: RAID Performance Issues

RAID Performance Issues

RAID was originally developed as a way of protecting data by providing fault tolerance; that's the reason for the "R" at the front of the acronym. Today, while matters of reliability, availability and fault tolerance continue to be essential to many of those who use RAID, performance issues are being given about as much attention. There are in fact whole classes of implementers who build RAID arrays solely for performance considerations, with no redundancy or data protection at all. Even those who do employ redundancy obviously care about getting the most from their array hardware.

The key to performance increases under RAID is *parallelism*. The ability to access multiple disks simultaneously allows for data to be written to or read from a RAID array faster than would be possible with a single drive. In fact, RAID is in some ways responsible for the demise of esoteric high-performance hard disk designs, such as drives with multiple actuators. A multiple-drive array essentially has "multiple actuators" without requiring special engineering; it's a win-win solution for both manufacturers (which hate low-volume specialty products) and consumers (which hate the price tags that come with low-volume specialty products).

There's no possible way to discuss every factor that affects RAID performance in a separate section like this one--and there really isn't any point in doing so anyway. As you read about RAID levels, and RAID implementation and configuration, many issues that are related to performance will come up. In this section I want to explore some of the fundamentals though, the basic concepts that impact overall array performance. One of my goals is to try to define better what exactly is meant by performance in a RAID context. Most people who know something about RAID would say "RAID improves performance", but some types improve it better than others, and in different *ways* than others. Understanding this will help you differentiate between the different RAID levels on a performance basis.

Next: Read and Write Performance

Read and Write Performance

Hard disks perform two distinct functions: writing data, and then reading it back. In most ways, the electronic and mechanical processes involved in these two operations are very similar. However, even within a single hard disk, read and write performance are often different in small but important ways. This is discussed in more detail here. When it comes to RAID, the differences between read and write performance are magnified. Because of the different ways that disks can be arranged in arrays, and the different ways data can be stored, in some cases there can be large discrepancies in how "method A" compares to "method B" for read performance, as opposed to write performance.

The fundamental difference between reading and writing under RAID is this: when you write data in a redundant environment, you must access every place where that data is stored; when you read the data back, you only need to read the minimum amount of data necessary to retrieve the actual data--the redundant information does not need to be accessed on a read. OK, this isn't as complicated as I probably just made it sound. :^) Let's see how various storage techniques used in RAID differ in this regard:

- **Mirroring:** Read performance under mirroring is far superior to write performance. Let's suppose you are mirroring two drives under RAID 1. Every piece of data is duplicated, stored on both drives. This means that every byte of data stored must be written to *both drives*, making write performance under RAID 1 actually a bit *slower* than just using a single disk; even if it were as fast as a single disk, both drives are tied up during the write. But when you go to read back the data? There's absolutely no reason to access both drives; the controller, if intelligently programmed, will only ask one of the drives for the data--the other drive can be used to satisfy a different request. This makes RAID significantly faster than a single drive for reads, under most conditions.
- **Striping Without Parity:** A RAID 0 array has about equal read and write performance (or more accurately, roughly the same ratio of read to write performance that a single hard disk would have.) The reason is that the "chopping up" of the data without parity calculation means you must access the same number of drives for reads as you do for writes.
- **Striping With Parity:** As with mirroring, write performance when striping with parity (RAID levels 3 through 6) is worse than read performance, but unlike mirroring, the "hit" taken on a write when doing striping with parity is much more significant. Here's how the different accesses fare:
    - For *reads*, striping with parity can actually be *faster* than striping without parity. The parity information is not needed on reads, and this makes the array behave during reads in a way similar to a RAID 0 array, except that the data is spread across one extra drive, slightly improving parallelism.
    - For sequential writes, there is the dual overhead of parity calculations as well as having to write to an additional disk to store the parity information. This makes sequential writes slower than striping without parity.
    - The biggest discrepancy under this technique is between random reads and random writes. Random reads that only require parts of a stripe from one or two disks can be processed in parallel with other random reads that only need parts of stripes on different disks. In theory, random writes would be the same, except for one problem: every time you change any block in a stripe, you have to recalculate the parity for that stripe, which requires *two* writes *plus* reading back all the other pieces of the stripe! Consider a RAID 5 array made from five

disks, and a particular stripe across those disks that happens to have data on drives #3, #4, #5 and #1, and its parity block on drive #2. You want to do a small "random write" that changes just the block in this stripe on drive #3. Without the parity, the controller could just write to drive #3 and it would be done. With parity though, the change to drive #3 affects the parity information for the entire stripe. So this single write turns into a read of drives #4, #5 and #1, a parity calculation, and then a write to drive #3 (the data) and drive #2 (the newly-recalculated parity information). This is why striping with parity stinks for random write performance. (This is also why RAID 5 implementations in software are not recommended if you are interested in performance.)

- Another hit to write performance comes from the dedicated parity drive used in certain striping with parity implementations (in particular, RAID levels 3 and 4). Since only one drive contains parity information, *every* write must write to this drive, turning it into a performance bottleneck. Under implementations with distributed parity, like RAID 5, all drives contain data and parity information, so there is no single bottleneck drive; the overheads mentioned just above still apply though.

Note: As if the performance hit for writes under striping with parity weren't bad enough, there is even one more piece of overhead! The controller has to make sure that when it changes data and its associated parity, all the changes happen simultaneously; if the process were interrupted in the middle, say, after the data were changed and not the parity, the integrity of the array would be compromised. To prevent this, a special process must be used, sometimes called a *two-phase commit*. This is similar to the techniques used in database operations, for example, to make sure that when you transfer money from your checking account to your savings account, it doesn't get subtracted from one without being certain that it was added to the other (or vice-versa). More overhead, more performance slowdown.

The bottom line that results from the difference between read and write performance is that many RAID levels, especially ones involving striping with parity, provide far better net performance improvement based on the ratio of reads to writes in the intended application. Some applications have a relatively low number of writes as a percentage of total accesses; for example, a web server. For these applications, the very popular RAID 5 solution may be an ideal choice. Other applications have a much higher percentage of writes; for example, an interactive database or development environment. These applications may be better off with a RAID 01 or 10 solution, even if it does cost a bit more to set up.

Note: Some controllers employ write caching to improve performance during writes; see here for more on this advanced feature.

Next: Positioning and Transfer Performance

Positioning and Transfer Performance

Much the way different RAID implementations vary in the degree to which they improve read and write performance, they also differ in how much they affect different types of accesses to data, both read *and* write. Data access to a single hard disk involves two discrete steps: first, getting the heads of the hard disk to exactly where the data is located, which I call *positioning*; and second, transporting the data to or from the hard disk platters, which I call *transfer*. Hard disk performance depends on both of these; the importance of one relative to the other depends entirely on the types of files being used and how the data is organized on the disk. For more on positioning and transfer, and related issues, see this section (if you don't understand these two concepts reasonably well, what follows may not mean as much to you as it should...)

The use of multiple hard disks in a RAID environment means that in some cases positioning or transfer performance can be improved, and sometimes both. RAID levels are often differentiated in part by the degree to which they improve either positioning or transfer performance compared to other levels; this becomes another factor to be weighed in matching the choice of RAID level to a particular application. It's important also to realize that the relative performance of positioning or transfer accesses depends also on whether you are doing reads or writes; often writes have overheads that can greatly reduce the performance of random writes by requiring additional reads and/or writes to disks that normally wouldn't be involved in a random write. See here for more details.

In general terms, here's how the basic storage techniques used in RAID vary with respect to positioning and transfer performance, for both writes and reads:

- **Mirroring:** During a read, only one drive is typically accessed, but the controller can use both drives to perform two independent accesses. Thus, mirroring improves positioning performance. However, once the data is found, it will be read off one drive; therefore, mirroring will not really improve sequential performance. During a write, both hard disks are used, and performance is generally worse than it would be for a single drive.
- **Striping:** Large files that are split into enough blocks to span every drive in the array require each drive to position to a particular spot, so positioning performance is not improved; once the heads are all in place however, data is read from all the drives at once, *greatly* improving transfer performance. On *reads*, small files that don't require reading from all the disks in the array can allow a smart controller to actually run two or more accesses in parallel (and if the files are in the same stripe block, then it will be even faster). This improves both positioning and transfer performance, though the increase in transfer performance is relatively small. Performance improvement in a striping environment also depends on stripe size and stripe width. Random writes are often degraded by the need to read all the data in a stripe to recalculate parity; this can essentially turn a

random write into enough operations to make it more resemble a sequential operation.

Which is more important: positioning or transfer? This is a controversial subject in the world of PCs, and one I am not going to be able to answer for you. This discussion of ranking performance specifications is one place to start, but really, you have to look at your particular application. A simple (but often *too* simple) rule of thumb is that the larger the files you are working with, the more important transfer performance is; the smaller the files, the more important positioning is. I would also say that too many people overvalue the importance of greatly increasing sequential transfer rates through the use of striping. A lot of people seem to think that implementing RAID 0 is an easy way to storage nirvana, but for "average use" it may not help performance nearly as much as you might think.

Next: Stripe Width and Stripe Size

Stripe Width and Stripe Size

RAID arrays that use striping improve performance by splitting up files into small pieces and distributing them to multiple hard disks. Most striping implementations allow the creator of the array control over two critical parameters that define the way that the data is broken into chunks and sent to the various disks. Each of these factors has an important impact on the performance of a striped array.

The first key parameter is the *stripe width* of the array. Stripe width refers to the number of parallel stripes that can be written to or read from simultaneously. This is of course equal to the number of disks in the array. So a four-disk striped array would have a stripe width of four. Read and write performance of a striped array increases as stripe width increases, all else being equal. The reason is that adding more drives to the array increases the *parallelism* of the array, allowing access to more drives simultaneously. You will generally have superior transfer performance from an array of eight 18 GB drives than from an array of four 36 GB of the same drive family, all else being equal. Of course, the cost of eight 18 GB drives is higher than that of four 36 GB drives, and there are other concerns such as power supply to be dealt with.

The second important parameter is the *stripe size* of the array, sometimes also referred to by terms such as *block size*, *chunk size*, *stripe length* or *granularity*. This term refers to the size of the stripes written to each disk. RAID arrays that stripe in blocks typically allow the selection of block sizes in kiB ranging from 2 kiB to 512 kiB (or even higher) in powers of two (meaning 2 kiB, 4 kiB, 8 kiB and so on.) Byte-level striping (as in RAID 3) uses a stripe size of one byte or perhaps a small number like 512, usually not selectable by the user.

Warning: Watch out for sloppy tech writers and marketing droids who use the term "stripe width" when they really mean "stripe size". Since stripe size is a user-defined parameter that can be changed easily--and about which there is lots of argument :^)--it is far more often discussed than stripe width (which, once an array has been set up, is really a static value unless you add hardware.) Also, watch out for people who refer to stripe size as being the *combined* size of all the blocks in a single stripe. Normally, an 8 kiB stripe size means that each block of each stripe on each disk is 8 kiB. Some people, however, will refer to a four-drive array as having a stripe size of 8 kiB, and mean that each drive has a *2 kiB* block, with the total making up 8 kiB. This latter meaning is not commonly used.

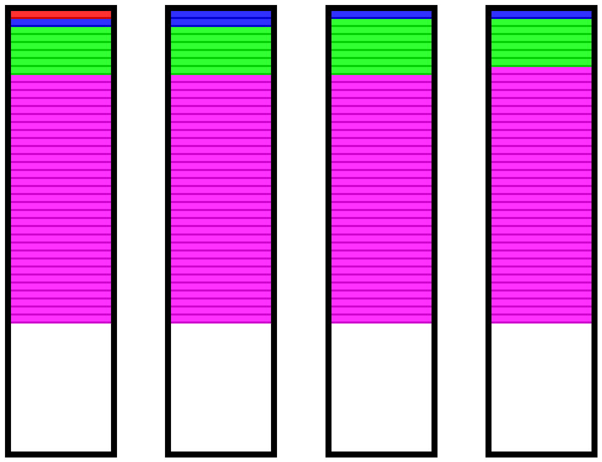
The impact of stripe size upon performance is more difficult to quantify than the effect of stripe width:

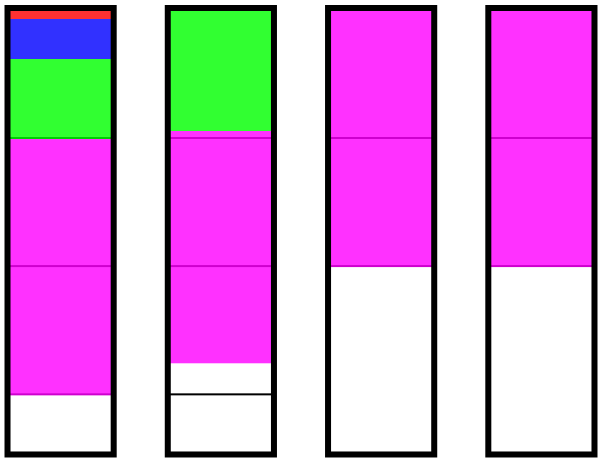
- **Decreasing Stripe Size:** As stripe size is decreased, files are broken into smaller and smaller pieces. This increases the number of drives that an average file will use to hold all the blocks containing the data of that file, theoretically increasing transfer performance, but decreasing positioning performance.

- **Increasing Stripe Size:** Increasing the stripe size of the array does the opposite of decreasing it, of course. Fewer drives are required to store files of a given size, so transfer performance decreases. However, if the controller is optimized to allow it, the requirement for fewer drives allows the drives not needed for a particular access to be used for another one, improving positioning performance.

Tip: For a graphical illustration showing how different stripe sizes work, see the discussion of RAID 0.

Obviously, there is no "optimal stripe size" for everyone; it depends on your performance needs, the types of applications you run, and in fact, even the characteristics of your drives to some extent. (That's why controller manufacturers reserve it as a user-definable value!) There are many "rules of thumb" that are thrown around to tell people how they should choose stripe size, but unfortunately they are all, at best, oversimplified. For example, some say to match the stripe size to the cluster size of FAT file system logical volumes. The theory is that by doing this you can fit an entire cluster in one stripe. Nice theory, but there's no practical way to ensure that each stripe contains exactly one cluster. Even if you could, this optimization only makes sense if you value positioning performance over transfer performance; many people do striping specifically for transfer performance.





A comparison of different stripe sizes. On the left, a four-disk RAID 0 array with a stripe size of 4 kiB; on the right, the same array with the same data, but using a 64 kiB stripe size. In these diagrams four files are shown color-coded: the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB. They are shown *drawn to scale* to illustrate how much space they take up in relative terms in the array (one vertical pixel represents 1 kiB.)
You can see how dramatically differently "medium-sized" files are treated as stripe size changes. The 4 kiB file takes only one block on one disk in both arrays, and the 500 kiB file is spread across all four disks in both arrays. But when the large stripe size is used, the blue file appears on only one disk instead of all four, and the green file is on two instead of four. This improves random positioning to these files. In both cases the stripe *width* is of course four. For a view of the same array with an "in-between" stripe size of 16 kiB, see the page on RAID 0.

So what should you use for a stripe size? The best way to find out is to try different values: empirical evidence is the best for this particular problem. Also, as with most "performance optimizing endeavors", don't overestimate the difference in performance between different stripe sizes; it can be significant, particularly if contrasting values from opposite ends of the spectrum like 4 kiB and 256 kiB, but the difference often isn't all that large between similar values. And if you *must* have a rule of thumb, I'd say this: transactional environments where you have large numbers of small reads and writes are probably better off with larger stripe sizes (but only to a point); applications where smaller numbers of larger files need to be read quickly will likely prefer smaller stripes. Obviously, if you need to balance these requirements, choose something in the middle. :^)

Note: The improvement in positioning performance that results from increasing stripe size to allow multiple parallel accesses to different disks in the array depends entirely on the controller's smarts (as do a lot of other things in RAID). For example, some controllers are designed to not do any writes to a striped array until they have enough data to fill an entire stripe across *all* the disks in the array. Clearly, this controller will not improve positioning performance as much as one that doesn't have this limitation. Also, striping with parity often requires extra reads and writes to maintain the integrity of the parity information, as described here.

Next: Degraded Operation and Rebuilding

Degraded Operation and Rebuilding

All of the discussions concerning RAID performance--both here and elsewhere--are based on the assumption that the array is operating normally, with all drives functioning. Redundancy-enabled RAID solutions provide the ability for the system to continue even one of the drives in the array has failed. However, when this occurs, performance is negatively affected; the array is said to be operating in a *degraded state* when this happens (some manufacturers may call this a *critical state* or use another synonym). The impact on performance depends on the type of RAID used by the array, and also how the RAID controller reacts to the drive failure.

When an array enters a degraded state, performance is reduced for two main reasons. The first is that one of the drives is no longer available, and the array must compensate for this loss of hardware. In a two-drive mirrored array, you are left with an "array of one drive", and therefore, performance becomes the same as it would be for a single drive. In a striped array with parity, performance is degraded due to the loss of a drive and the need to regenerate its lost information from the parity data, on the fly, as data is read back from the array.

The second reason for degraded operation after a drive failure is that after the toasted drive is replaced, the data that was removed from the array with its departure must be regenerated on the new disk. This process is called *rebuilding*. A mirrored array must copy the contents of the good drive over to the replacement drive. A striped array with parity must have the entire contents of the replacement drive replaced by determining new parity information (and/or replacement data calculated from parity information) for *all* the data on the good drives. Clearly, these procedures are going to be time-consuming and also relatively slow--they can take several hours. During this time, the array will function properly, but its performance will be greatly diminished. The impact on performance of rebuilding depends *entirely* on the RAID level and the nature of the controller, but it usually affects it significantly. Hardware RAID will generally do a faster job of rebuilding than software RAID. Fortunately, rebuilding doesn't happen often (or at least, it shouldn't!)

Many RAID systems give the administrator control over whether the system does *automatic* or *manual* rebuilds. In an automatic configuration, the array will detect the replacement of the dead drive and begin rebuilding automatically on the new one--or it may start the rebuild as soon as the bad drive fails if the array is equipped with hot spares. In manual mode, the administrator must tell the system when to do the rebuild. Manual is not necessarily "worse" than automatic, because if the system is not one that runs 24 hours a day, 7 days a week, the administrator will often prefer to wait until after hours to rebuild the array, thereby avoiding the performance hit associated with the rebuild. However, take the following warning into account as well...

Warning: Most regular RAID arrays using mirroring or striping with parity are in a vulnerable state when they are running in degraded mode. Until the offending drive is replaced and rebuilt, they provide *no data protection*. Do not excessively procrastinate rebuilding a degraded array; even if you are just waiting for the end of the day, recognize that you are taking risks in doing so.

Note: None of this section applies when you are doing striping without parity (RAID 0). When a drive in a RAID 0 array fails, performance doesn't degrade, it comes to a screeching halt. :^) The reason is that RAID 0 includes no redundancy information, so the failure of any drive in the array means all the data in the array is lost, short of heroics. See here for more details.

Next: RAID Reliability Issues

RAID Reliability Issues

While the performance improvements that result from the use of RAID are of course important, the primary goal of the use of RAID for most businesses is the protection of both their critical data and their employees' productivity. In a properly-implemented RAID system, down-time due to hardware problems can be dramatically reduced, and data loss due to drive failures all but eliminated.

In order to understand how RAID really affects important reliability considerations, it is necessary to first understand the various issues that are related to that somewhat nebulous term. For example, a commonly-heard phrase is that "RAID improves hard disk reliability", but that's not an accurate statement. The truth depends to some extent on how you define reliability: do you mean the reliability of the individual drives, or the whole system? Are you talking about the data, or the hardware itself? And of course, not all RAID implementations improve reliability in *any* way.

In this section, I take a closer look at the key issues related to the general topic of reliability under RAID. This includes a more thorough look at the concepts of fault tolerance, reliability and availability, a discussion of the reliability of other system components that affect the reliability of the system as a whole, and a discussion of backups and data recovery.

Next: Reliability

Reliability

If you ask an IT professional to list the reasons why he or she set up a RAID array, one of the answers likely to be mentioned is "increased reliability". They probably don't really mean it though. ;^) As I have implied in many other areas of the site's coverage of RAID, "reliability" is a vague word when it comes to redundant disk arrays. The answer of increased reliability is both true and not true at the same time.

The reliability of an individual component refers to how likely the component is to remain working with a failure being encountered, typically measured over some period of time. The reliability of a component is a combination of factors: general factors related to the design and manufacture of the particular make and model, and specific factors relevant to the way that particular component was built, shipped, installed and maintained.

The reliability of a *system* is a function of the reliability of its components. The more components you put into a system, the worse the reliability is of the system as a whole. That's the reason why compex machines typically break down more frequently than simple ones. While oversimplified, the number used most often to express the reliability of many components, including hard disks, is mean time between failures (MTBF). If the MTBF values of the components in a system are designated as $MTBF_1$, $MTBF_2$, and so on up $MTBF_N$, the reliability of the system can be calculated as follows:

System MTBF = $1 / (1/MTBF_1 + 1/MTBF_2 + ... + 1/MTBF_N)$

If the MTBF values of all the components are equal (i.e., $MTBF_1 = MTBF_2 = ... = MTBF_N$) then the formula simplifies to:

System MTBF = Component MTBF / N

The implications of this are clear. If you create a RAID array with four drives, each of which has an MTBF figure of 500,000 hours, the MTBF of the *array* is only 125,000 hours! In fact, it's usually worse than that, because if you are using hardware RAID, you must also include the MTBF of the *controller*, which without the RAID functionality, wouldn't be needed. For sake of illustration, let's say the MTBF of the controller card is 300,000 hours. The MTBF of the storage subsystem then would be:

System MTBF = $1 / (1/MTBF_1 + 1/MTBF_2 + ... + 1/MTBF_N)$
$= 1 / (1/500000 + 1/500000 + 1/500000 + 1/500000 + 1/300000)$
$= 88,235$

So in creating our array, our "reliability" has actually decreased 82%. Is that right? Why then do people bother with RAID at all? Well, that's the other side of the reliability coin. While the reliability of the array hardware goes down, when you include redundancy information through mirroring or parity, you provide *fault tolerance*, the ability to withstand and recover from a failure. This allows the decreased reliability of the array to allow failures to occur without the array or its data being disrupted, and that's how RAID provides data protection. Fault tolerance is discussed here. The reason that most people say RAID improves reliability is that when they are using the term "reliability" they are including in that the fault tolerance of RAID; they are not really talking about the reliability of the hardware.

What happens if you don't include redundancy? Well, then you have a ticking time-bomb: and that's exactly what striping without parity, RAID 0, is. A striped array without redundancy has substantially *lower* reliability than a single drive and no fault tolerance. That's why I do not recommend its use unless its performance is absolutely required, and it is supplemented with very thorough backup procedures.

Next: Fault Tolerance

Fault Tolerance

When most people talk about the "enhanced reliability of RAID", what they really are referring to is the *fault tolerance* of most RAID implementations. Reliability and fault tolerance are not the same thing at all, as described in the discussion of RAID array reliability. Fault tolerance refers to the ability of a RAID array to withstand the loss of some of its hardware without the loss of data or availability. When a fault occurs, the array enters a degraded state and the failed drive must be replaced and rebuilt.

The capability of an array to tolerate hard disk faults depends entirely on the RAID level implemented. RAID 0, which has no redundant information, has no fault tolerance for that reason; if any drive fails the array goes down. RAID levels 1, 2, 3, 4, 5 and 7 can tolerate the loss of one hard disk; RAID 6 can tolerate the loss of two. The multiple RAID levels can often tolerate the loss of multiple hard disks depending on which ones they are and how the array is configured. For example, if you have a RAID 10 array consisting of two pairs of mirrored drives striped together, the array can tolerate the simultaneous loss of two of the four drives, as long as they are *not in the same pair*.

Ultimately, fault tolerance of a RAID-equipped PC depends not only on the reliability of the drives, but also the other hardware in the system. For example, most RAID implementations are dependent upon the RAID controller not failing; if it goes down then your data is probably fine, but availability is not--this is one reason why duplexing is sometimes used, since it can tolerate the loss of a drive *and* a controller. Since items such as power supplies have a high failure rate and ultimately can bring down any array, fault tolerance in a PC equipped with RAID is also often used to describe features implemented on the system as a whole to prevent faults in this support equipment; see here for more details.

Next: Availability

Availability

Another issue related to reliability and fault tolerance but not the same as either of them is *availability*. This term refers simply to the ability of users to continue to get access to their data. If people are using the array to do their work, then it is "available"; if everyone's gathered around the water cooler talking about how much money some guy won last night on "Who Wants To Be A Millionaire?" then it is "not available". :^)

For many businesses that implement RAID, availability is the "bottom line" reason why they decided to spend the money for that technology. (Data protection is usually just as important, but you can get pretty good data protection through backups, saving the cost of all the RAID hardware, and you still need the backups anyway.) For some businesses, the cost of losing access to their data for an hour can exceed the price of the entire RAID system.

The availability of an array depends on several factors and features:

- **Hardware Reliability:** The higher the reliability of the hardware in the array, the less the chances of there being a hardware problem in the first place; this is the best way to keep availability high, of course. :^) The reliability of support hardware is just as important as that of the drives in the array.
- **Fault Tolerance:** RAID arrays and implementations that have higher fault tolerance provide higher availability. Again here, the fault tolerance of support hardware is also important.
- **Hot Swapping:** RAID arrays that include the ability to hot swap drives eliminate the need to shut down the system in the event of a hardware failure.
- **Automatic Rebuilding:** If the system runs continuously, the ability to automatically rebuild the drive while the array continues to operate is essential.
- **Service:** If an array goes down, availability is gone until the hardware fault is corrected. Though this is uncommon in a properly-configured RAID system it does occur, and when it does, availability rapidly boils down to how fast the hardware can be serviced.

If high availability is important then it can be achieved--at a cost. There are in fact companies that design special, highly-fault-tolerant systems that can withstand just about anything not involving the use of explosives. :^) Very high-end solutions for availability-critical applications go beyond RAID to the use of complete redundant systems, often in different locations--these can even handle explosives!

Next: Reliability of Other System Components

Reliability of Other System Components

As described in this section, the reliability of a system is a function of the reliability of the various components that comprise it. The more components in a system, the less reliable a system will be. Furthermore, in terms of reliability, the chain is truly only as strong as its weakest link. When dealing with a PC, there are a number of critical components without which the system will not function; if one of these hardware pieces fails then your array will go down, regardless of the number of disks you have or how well they are manufactured. This is an important point that too few people consider carefully enough when setting up a RAID box.

One unreliable component can severely drag down the overall reliability of a system, because the MTBF of a system will always be lower than the MTBF of the *least* reliable component. Recall the formula for reliability of a system:

System MTBF = 1 / ($1/MTBF_1 + 1/MTBF_2 + ... + 1/MTBF_N$)

Also recall that if the MTBF values of all the components are equal (i.e., $MTBF_1 = MTBF_2 = ... = MTBF_N$) then this boils down to:

System MTBF = Component MTBF / N

This means that if we have four components with an MTBF of 1,000,000 hours each, the MTBF of the system is 250,000 hours. But if we have four components, of which three have an MTBF of 1,000,000 hours and the fourth has an MTBF of 100,000 hours? In this case, the MTBF of the system drops to only about 77,000 hours, one-third of the previous value.

What this all means is that you can have the greatest hard disks in the world, and use multiple RAID levels to protect against drive failure, but if you put it all in a system with lousy support components, you're not going to have a reliable, high-availability system. It's as simple as that, but in fact, it's actually *worse* than that. While RAID reduces reliability, it improves fault tolerance; however, most of the other components in the system have no fault tolerance. This means that the failure of any one of them will bring down the PC. Of particular concern are components that affect all the drives in a system, and which generally have a reputation for problems or relatively low reliability.

To increase the reliability of the PC as a whole, systems using RAID are usually designed to use high-quality components. Many systems go beyond this, however, by introducing fault tolerance into other key components in the system that often fail. Since many of the most common problems with PCs are related to power, and since without the power supply *nothing* in the PC will operate, many high-end RAID-capable systems come equipped with redundant power supplies. These supplies are essentially a pair of supplies in one, either of which can operate the PC. If one fails then the other can handle the entire load of the PC. Most also allow hot swapping, just like hard disk hot swapping in a RAID array. See this section for more.

Another critical issue regarding support hardware relates to power protection--your PC is completely dependent on the supply of electricity to the power supply unless you use a UPS. In my opinion, any application important enough to warrant the implementation of a fault-tolerant RAID system is also important enough to justify the cost of a UPS, which you can think of as "fault tolerance for the electric grid". In addition to allowing the system to weather short interruptions in utility power, the UPS also protects the system against being shut down abruptly while in the middle of an operation. This is especially important for arrays using complex techniques such as striping with parity; having the PC go down in the middle of writing striped information to the array can cause a real mess. Even if the battery doesn't last through a power failure, a UPS lets you shut the PC down gracefully in the event of a prolonged outage.

How about components like motherboards, CPUs, system memory and the like? They certainly are critical to the operation of the system: a multiple-CPU system can handle the loss of a processor (though that's a pretty rare event in any case) but no PC around will function if it has a motherboard failure. Most systems running RAID do not provide any protection against failure of these sorts of components. The usual reason is that there is no practical way to protect against the failure of something like a motherboard without going to (very expensive) specialty, proprietary designs. If you require protection against the failure of *any* of the components in a system, then you need to look beyond fault-tolerance within the PC and configure a system of redundant machines.

Next: The Continued Importance of Backups

The Continued Importance of Backups

I suppose I should apologize for the fact that I tend to hound my readers on the subject of backups. Hmm. Nah. :^) Well, I *do* apologize to those of you who already know to do backups and actually do them, but unfortunately, you by far represent the minority of PC users. Lack of proper backup procedures is risky at any time, but with some types of RAID, especially RAID 0, it becomes riskier than ever.

Most RAID levels use striping, which causes most of the files on the system to be "chopped up" into small pieces and distributed to various drives in the array. This improves performance but poses a real risk to the integrity of your data. If a drive goes on a striped array, "some part" of almost every file in the system is gone, and there's really no way to figure out what is gone. A fault-tolerant array that includes parity protection eliminates much of this risk by allowing a single drive to fail, but a RAID 0 array does not. Since arrays have much *lower* hardware reliability than individual drives, this means the odds of a catastrophic, near-unrecoverable failure on a RAID 0 array are higher than most people realize. (Data recovery companies can sometimes perform miracles; frankly, if your data is important enough to warrant spending what a recovery service would charge to reconstruct a RAID 0 array, it's important enough that it shouldn't be *on* a RAID 0 array. An ounce of prevention and all that...)

If you are using mirroring, either by itself or in combination with another technique, your odds of losing your data due specifically to hardware failure are greatly reduced, but they are far from zero. Regardless, there are a host of other problems that could wipe out your data (for more on these risks, see here):

- **Unexpected Hard Disk Failures:** It's not unheard of for two mirrored drives to fail simultaneously, especially if they are the exact same make and model and came from the same lot. Sometimes manufacturers have a "bad run" and the failure rate of a particular batch of drives is very high.

- **Failures of Support Hardware:** The other hardware that runs the entire array can fail. Power supplies fail, or a motherboard develops a problem due to overheating, or memory goes bad. These can cause either outright failure of the array, or even worse, gradual corruption in some circumstances.
- **Physical Damage:** Due to natural disasters, carelessness or sabotage, your hardware could be physically damaged. This will affect all or most of the drives in the array.
- **Software Problems:** Software has bugs. This includes all operating systems, although some are much better than others. Bugs can lead to data loss.
- **Viruses:** A virus can wipe out individual files or entire disk volumes at the system level; no form of RAID will help you.
- **Human Error:** This is one of the leading causes of data loss; again, RAID will not protect you against an accidental file deletion, formatting the wrong partition, etc.

That's not a comprehensive list, but it's good enough to make my point: even a robust RAID implementation does not replace proper backup procedures. The more you can afford to put together a good RAID system, the more important your data is, and the more you should spend on a good backup system.

Next: Data Recovery

Data Recovery

Under normal circumstances, if a hard disk fails, its data is no longer accessible. Most failures, especially mechanical ones, do not actually cause the data on the platter surfaces to be erased, however. This means that with special procedures, it is possible to retrieve most, if not all, of the data on a hard disk--if it is important enough to justify the expense. This process is called *data recovery* and is discussed in general terms in this section.

RAID, particularly RAID that uses striping, complicates data recovery significantly. The data is no longer placed in relatively simple fashion on a single hard disk, but rather, distributed across many drives. Despite this complication, it *is* possible to do data recovery in most cases; it just costs more due to the complexity of the array. For example, there have been cases of two drives failing on a RAID 5 array and the data being successfully recovered.

That said, it should be remembered that compared to a single drive, a RAID array that includes redundancy provides fault tolerance that means in most cases a drive failure won't result in any lost data at all. If a fault occurs, and it is dealt with quickly and competently, in most cases the need for data recovery never arises. In fact, some companies refer to rebuilding a replaced failed drive in a RAID array as "real-time data recovery", which I suppose it is, in a way, though I personally find that term a bit confusing.

Next: RAID Levels

RAID Levels

There are many different ways to implement a RAID array, using some combination of mirroring, striping, duplexing and parity technologies. Several standardized methods were defined in the 1988 Berkeley research publication that is credited with starting the RAID phenomenon; for some (unfortunate) reason, the researchers decided to call these different techniques *levels*. This was a poor choice of words in my opinion because the word "level" implies hierarchy or revision, or that the different RAID levels are somehow "built upon each other", when in fact, that is not the case. The word "level" implies to some people that "RAID level N+1" is *better* somehow than "RAID level N". In fact, this isn't really true--the various levels are independent and different, and no strict hierarchy should be inferred from the specific number attached to a RAID level. A given RAID level that is "better" for one person may be "worse" for another.

The original 1988 paper defined RAID levels 1 through 5; since then, single RAID levels 0 and 6 have been added to the mix, and other extensions such as the proprietary RAID 7 have shown up as well. Beyond these single-level RAID designs, a number of *multiple* RAID levels have been defined, which use two or more of the single RAID levels in combination to create new array types with new capabilities (and limitations). Most of these different RAID levels are in use today in different systems, a testament to the different needs of various RAID users. Some have largely disappeared from the market as experience over time has shown them to be inferior to other levels without advantages to compensate.

In this section I take a detailed look at RAID levels. I start with a discussion of some of the key technical factors that differentiate RAID levels; these are then used to frame the coverage of the RAID levels themselves. I discuss the eight single-level RAID designs, and take a look at several common multiple-level RAID types as well. Each RAID level is discussed in detail and information is provided about over a dozen of its various characteristics, with general recommendations provided for typical uses of each level. Finally, I show a summary comparison table that contrasts the different levels in terms of their benefits and costs.

Tip: Carefully consider *all* the factors and variables when comparing different RAID levels; sometimes, things are not what they seem. Pay careful attention to the various performance attributes, to help differentiate levels based on how you are most likely to use the array; sometimes the "common wisdom" about different RAID levels will not apply to your needs.

Warning: As I am noticing increasingly these days in all areas of computing, the RAID levels are sometimes not used consistently by manufacturers. For example, I have encountered a significant RAID controller maker that provides support for what they call "RAID 3"; when you examine the details, however, you find that this RAID level is actually implemented as block striping with dedicated parity, which is RAID *4*, not 3. Why they did this, I have *no* idea. An organization called the *RAID Advisory Board* or *RAB* does maintain RAID standards and certifies hardware that meets "official" RAID level definitions, so you can look for their "seals of approval". Even so, it's still best to ask for specific technical details about any RAID system if you aren't certain of what the manufacturer has implemented.

Next: Technical Factors Differentiating RAID Levels

Technical Factors Differentiating RAID Levels

The reason why there are so many different RAID levels is that there are many different ways to configure a bunch of hard disks, and many different needs of RAID users. Distinguishing between different levels isn't easy at times, because many RAID levels are similar to others in various ways. Worse, sometimes the differences between levels seem subtle, but these small discrepancies can have a huge impact on the characteristics of the array and the applications that make sense for it.

To completely and accurately portray each single and multiple RAID level, I describe each in terms of its most important characteristics, including those related to fault tolerance, capacity, performance, cost and other attributes. To avoid duplication, I have provided this section that describes what each of these technical factors or attributes are about. For each one I explain briefly what the attribute means, how it is defined, and how and why it helps differentiate between various RAID levels. In the last sub-section, I also discuss the reasons why some implementers may wish to consider creating more than one array for a system if the needs of that system can't be met by one array type.

Note that in addition to the descriptions in this section, you may also want to take a look at the more general discussions of performance and reliability issues in the section covering general concepts and issues. Some of the pages that follow in this area will refer back to those or other pages in the site's coverage of RAID.

Next: Redundancy Technique: Mirroring vs. Parity

Redundancy Technique: Mirroring vs. Parity

The technique (or techniques) used to provide redundancy in a RAID array is a primary differentiator between levels. Redundancy is provided in most RAID levels through the use of mirroring or parity (which is implemented with striping):

- **Mirroring:** Single RAID level 1, and multiple RAID levels 0+1 and 1+0 ("RAID 10"), employ mirroring for redundancy. One variant of RAID 1 includes mirroring of the hard disk controller as well as the disk, called duplexing.
- **Striping with Parity:** Single RAID levels 2 through 7, and multiple RAID levels 0+3 (aka "53"), 3+0, 0+5 and 5+0, use parity with striping for data redundancy.
- **Neither Mirroring nor Parity:** RAID level 0 is striping without parity; it provides no redundancy
- **Both Mirroring *and* Striping with Parity:** Multiple RAID levels 1+5 and 5+1 have the "best of both worlds", both forms of redundancy protection.

The exact way that striping with parity is implemented depends on the particulars of the level as well. Some levels involve striping of individual bytes or sectors, while others use larger blocks; in the latter case, the size of the blocks is controlled by the stripe size of the array. RAID 2 uses a special form of striping with parity that is actually more like the ECC encoding used to protect data on various computer devices.

The choice of method for attaining redundancy affects almost every other characteristic of an array, which is why I consider it a primary differentiator. All aspects of performance, fault tolerance, impact of rebuilding, storage efficiency, cost and more: they all depend on the choice of redundancy technique. For more information on how mirroring works and its general advantages over parity; see here. For more on parity and its advantages over mirroring, see here.

Next: Controller Requirements

Controller Requirements

RAID levels differ in terms of the controller required to implement them. In general, simple controllers implement simple RAID levels, while more complex RAID levels require more sophisticated (read: expensive) controllers. Some levels don't require a dedicated controller at all, functioning acceptably using operating system or other software to manage the array (albeit at a performance price).

The simplest RAID levels, often supported by software RAID or low-end controllers, include RAID 0, RAID 1, and RAID 0+1 or 1+0 (though many low-end controllers only support one or the other, not both.) Some lower-end controllers include support for the popular RAID 5 level, and so do some software RAID implementations, but at a rather severe penalty in terms of performance and advanced features. RAID levels 3, 4, 6 and 7, and the other more esoteric multiple RAID levels such as RAID 53 and RAID 30, usually require more capable controllers. RAID 2 is complex enough that it requires special, proprietary hardware; it has for this reason all but disappeared from the market.

Next: Hard Disk Requirements

Hard Disk Requirements

Different RAID levels have varying requirements for the hard disks used in the array. The most important difference between levels is related to the minimum number of drives in the array, which depends entirely on how the RAID level implements mirroring, striping, and parity. Simple striping (RAID 0) requires two or more drives; mirroring (RAID 1) requires two drives; and striping with parity requires at least three drives (two or more for data stripes and one for parity, whether it is dedicated to a single drive or distributed). Striping with double parity (RAID 6) requires at least four drives. Multiple RAID levels generally require a number of drives equal to the product of the minimum number of drives of the single levels that comprise them. For example, RAID 10 requires a minimum of four drives (and must consist of an even number of drives) and RAID 50 requires at least six!

The maximum number of drives is typically limited by the RAID controller, not by anything inherent in the definition of the RAID level. The exception to this rule is RAID 1, which works with two drives only. This limitation is one reason why RAID 1 is rarely used in larger arrays (RAID 0+1 or 1+0 are used where the advantages of RAID 1 are required in combination with larger capacity.)

Finally, all RAID levels work best when fitted with identical drives of identical capacity. Some RAID levels can tolerate differences in performance between drives in the array better than others (typically the simpler levels). All RAID arrays make best use of the space on the disk when the drives are the same size; see here for more details.

For more on hard disk requirements for RAID and selecting hard disks, see this section.

Next: Array Capacity and Storage Efficiency

Array Capacity and Storage Efficiency

The *capacity* of a RAID array is determined by multiplying the capacity of the drives in the array by the number of non-redundant drives, i.e., drives used for data as opposited to parity or mirrored data. The more redundancy in the array, the lower the capacity of the array for a given number of drives. For example, six 40 GB hard disks in a RAID 0 array would yield a 240 GB array; in a RAID 5 configuration that same array would have 200 GB of usable capacity; in a RAID 1+0 setup, only 120 GB. The total capacity of the array depends on both the RAID level and the size and number of drives used.

The *storage efficiency* of the array is the percentage of the total size of the array that can be used for user data; it is more useful for comparing RAID levels because it does not depend on the size of the drives used in the array. It can be calculated simply by dividing the total usable capacity of the drive by the sum of the capacities of all the drives that underlie the array. For some RAID levels it is a constant, while for others it depends on the number of drives in the array. For example, the storage efficiency of RAID 0 is always 100%, and for RAID 1 and RAID 0+1, 50%. For levels that use striping with parity, storage efficiency increases as the number of drives is increased, because the number of parity drives is a constant and therefore diminishes as a percentage of the total.

While common parlance for calculating storage says to multiply the number of data-containing drives by the capacity of the drives, bear in mind that this assumes that all the drives are of the same size (which is the ideal way to set up a RAID array). In fact, the capacity of any RAID array is constrained by the size of the smallest drive in the array; it is that size that should be used when calculating capacity. So, the capacity of a RAID 5 array containing four drives that are 20 GB, 30 GB, 40 GB and 45 GB in size is 60 GB (calculated as 4-1 times 20 GB, the size of the smallest drive.) If you were to actually set up an array using such drives, 55 GB of space would be lost in what is sometimes called *drive waste*. If an array has drive waste due to unequal drive sizes, this will reduce the storage efficiency of the array as well, of course: the RAID 5 array given as an example here has a storage efficiency of (60 / 135) = 44%, instead of the 75% that would result if all the drives were the same size. Some controllers will let you use such "waste" as regular, non-RAID storage. See this discussion of drive size for related issues.

Next: Fault Tolerance and Availability

Fault Tolerance and Availability

Of course, RAID levels vary a great deal in terms of their ability to handle faults (usually meaning drive failures) and keep data availability high. The need for fault tolerance dictates the choice of RAID level probably more than any other requirement. In "boiled down" terms, the more redundancy in the array, the more fault tolerant it is and the higher the availability of its data, but the way the drives are connected is also very important. Each level has the particulars of its ability to deal with these important reliability-related issues discussed in detail in its own section. For a more thorough, more general look at these matters, check out the section on reliability issues.

Note: While RAID levels obviously vary in the degree to which they can tolerate faults, availability is often almost as much a function of implementation as it is design. High availability in particular is dependent upon the use of advanced features such as drive swapping and hot spares.

Next: Degradation and Rebuilding

Degradation and Rebuilding

A fault-tolerant (redundant) array can continue operating after experiencing a failure of one (or sometimes more) of its drives. When this happens the array enters a *degraded state* until the faulty drive is replaced and its contents are *rebuilt*. Running in a degraded state has an important impact on both the performance and subsequent fault tolerance of the array; see this full discussion for more.

Some RAID levels handle degradation more "elegantly" than others; some can in fact tolerate the loss of more than one drive, while others cannot handle the loss of even one. While in a degraded mode, some RAID levels suffer much more performance loss than others; generally, the more complex the RAID level, the more its performance decreases when the array is degraded (simple mirroring shows little loss of performance, while striping with parity shows much more). Also, alternate ways of implementing multiple RAID levels can be surprisingly different in terms of the number of lost drives they can tolerate, and also in the impact on performance when rebuilding after a fault.

Next: Performance

Performance

Like fault tolerance, performance is one of the main reasons that many people implement RAID arrays; and like fault tolerance, it is an important way that RAID levels differ greatly from one to the next. In fact, some RAID levels differ *only* in terms of performance.

Not only does performance vary between RAID levels, different *types* of performance vary between RAID levels. It is not meaningful to boil things down to just "performance"; rather, we must look at the issues of read and write performance; and positioning and transfer performance. RAID level "A" may be better than level "B" at one aspect of performance but much worse than "B" in another.

To make things more clear in explaining and contrasting the different RAID levels, I have combined these two two-element variables--read vs. write and positioning (random access) vs. transfer (sequential access)--into a "matrix" of four specific performance categories:

- **Random Read Performance:** How the RAID level performs on random access reads of files in the array. Typically, this is most important for transactional environments with smallish files, especially ones with a high ratio of reads to writes.
- **Random Write Performance:** How the RAID level performs when writing small files in random places on the array. Again, this is most relevant to transaction-processing environments, however, it is even more important to applications where a large number of writes are done, because write performance is much worse than read performance for many popular RAID levels.

- **Sequential Read Performance:** The performance of the RAID level when reading large files sequentially from the array. This is of greatest concern in applications where there are many more reads than writes, for example, a server containing many large graphics files.
- **Sequential Write Performance:** The RAID level's general performance when writing large files. This is sometimes less important than sequential read performance, but is critical for situations where large files are written often, such as video or audio editing.

Notes about performance assessments and comparisons (i.e., "very good", "fair", "poor", etc.) of RAID levels:

- All performance ratings are based on the assumption that hardware RAID is being used. Software RAID will often be substantially slower.
- Keep in mind that they are just rough generalizations; exact performance depends on disk type, controllers, stripe size, implementation details and a dozen other factors.
- Rankings are relative to other RAID levels. A "poor" rating means that RAID level does poorly on that type of transfer, even though it may still be faster than a single drive at that task.
- Write performance is generally worse than read performance, "scores" for write performance are generally rated against write performance of other RAID arrays.

Next: Multiple Array Considerations

Multiple Array Considerations

When reading about the various RAID levels you will quickly come to the (correct) conclusion that choosing a level is largely a matter of tradeoffs and compromises. This is especially true when it comes to dealing with performance considerations. If you have to serve two very different applications, you may not be able to easily find a single RAID level that satisfies all of your requirements. If that is the case, you may be better off creating two separate arrays, using different RAID levels for each.

Let's suppose that you are setting up an enterprise server for a business of 100 people; you can afford a reasonable amount of hardware but you do *not* have an unlimited budget. The primary application is a business system that requires fast performance and some degree of fault tolerance. However, you also have a design and engineering department, dealing with huge files, and they need "speed, speed and more speed"; and an accounting department that *insists* upon a mirrored setup for their data because they believe that provides the highest integrity. You could just set up a RAID 5 array and make everyone use it. It will provide fault tolerance and good performance for most uses, but the engineering and accounting folks probably won't be too thrilled. Instead, you could set up both a RAID 5 array and either a RAID 0 or RAID 1 array to satisfy at least one of the "specialty" groups. You could also set up three arrays--RAID 5 for general use, RAID 0 for the design group and RAID 1 for accounting--but realize that you are then getting into more of an administrative headache.

An excellent way to consider doing multiple arrays is to use different machines. This not only spreads the load around, it removes the machine containing the arrays as a single point of failure, improving the overall fault tolerance of the organization. So to continue the example above, perhaps the best solution is to fit a main server with a larger RAID 5 array for general use and a smaller RAID 1 array for accounting, and configure a second, departmental server for design and engineering that runs RAID 0. The RAID 0 array can be set up with an inexpensive controller or even software RAID, to avoid duplicating the controller costs. Just make sure it gets completely backed up every night! :^)

If you do set up two (or more) arrays on the same machine, you will have to decide exactly how to do this. Most better controllers will support multiple arrays without too much trouble. Some will let you set up more than one logical array based on a single physical array, but I personally recommend using different drives for each array. This makes things conceptually simpler, and you also may be able to avoid degrading both arrays in the event of a drive failure.

Next: Single RAID Levels

Single RAID Levels

In this section I take a look at the "single" RAID levels--meaning, the "regular" RAID levels, as opposed to multiple or nested RAID levels. Single RAID levels are by far the most commonly used since they are simpler and less expensive to implement, and satisfy the needs of most RAID users. Generally, only very high-end or specialty applications require the use of multiple RAID levels.

There are eight "regular" RAID levels, which are used to varying degrees in the "real world" today. A few levels, especially RAID 0, RAID 1 and RAID 5, are extremely popular, while a couple are rarely if ever seen in modern systems. For each level, I provide a comprehensive discussion of its attributes and characteristics in the areas of capacity, performance, fault tolerance, cost and more.

Note: For an explanation of any of the categories used in describing the RAID levels, see the section discussing technical factors differentiating RAID levels.

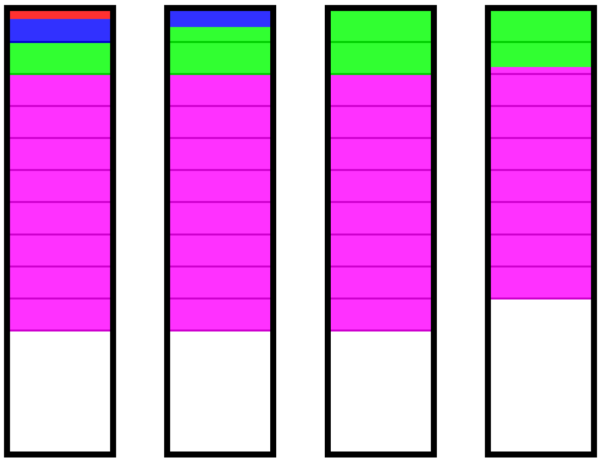
Next: RAID Level 0

RAID Level 0

Common Name(s): RAID 0. (Note that the term "RAID 0" is sometimes used to mean not only the conventional striping technique described here but also other "non-redundant" ways of setting up disk arrays. Sometimes it is (probably incorrectly) used just to describe a collection of disks that doesn't use redundancy.)

Technique(s) Used: Striping (without parity)

Description: The simplest RAID level, RAID 0 should really be called "AID", since it involves no redundancy. Files are broken into stripes of a size dictated by the user-defined stripe size of the array, and stripes are sent to each disk in the array. Giving up redundancy allows this RAID level the best overall performance characteristics of the single RAID levels, especially for its cost. For this reason, it is becoming increasingly popular by performance-seekers, especially in the lower end of the marketplace.



This illustration shows how files of different sizes are distributed between the drives on a four-disk, 16 kiB stripe size RAID 0 array. The red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB. They are shown *drawn to scale* to illustrate how much space they take up in relative terms in the array--one vertical pixel represents 1 kiB. (To see the impact that increasing or decreasing stripe size has on the way the data is stored in the array, see the 4 kiB and 64 kiB stripe size versions of this illustration on the page discussing stripe size issues.)

Controller Requirements: Supported by all hardware controllers, both SCSI and IDE/ATA, and also most software RAID solutions.

Hard Disk Requirements: Minimum of two hard disks (some may support one drive, the point of which escapes me); maximum set by controller. Any type may be used, but they should be of identical type and size for best performance and to eliminate "waste".

Array Capacity: (Size of Smallest Drive * Number of Drives).

Storage Efficiency: 100% if identical drives are used.

Fault Tolerance: None. Failure of any drive results in loss of all data, short of specialized data recovery.

Availability: Lowest of any RAID level. Lack of fault tolerance means no rapid recovery from failures. Failure of any drive results in array being lost and immediate downtime until array can be rebuilt and data restored from backup.

Degradation and Rebuilding: Not applicable.

Random Read Performance: Very good; better if using larger stripe sizes if the controller supports independent reads to different disks in the array.

Random Write Performance: Very good; again, best if using a larger stripe size and a controller supporting independent writes.

Sequential Read Performance: Very good to excellent.

Sequential Write Performance: Very good.

Cost: Lowest of all RAID levels.

Special Considerations: Using a RAID 0 array without backing up any changes made to its data at least daily is a loud statement that that data is not important to you.

Recommended Uses: Non-critical data (or data that changes infrequently and is backed up regularly) requiring high speed, particularly write speed, and low cost of implementation. Audio and video streaming and editing; web servers; graphic design; high-end gaming or hobbyist systems; temporary or "scratch" disks on larger machines.

Next: RAID Level 1

RAID Level 1

Common Name(s): RAID 1; RAID 1 with Duplexing.

Technique(s) Used: Mirroring or Duplexing

Description: RAID 1 is usually implemented as mirroring; a drive has its data duplicated on two different drives using either a hardware RAID controller or software (generally via the operating system). If either drive fails, the other continues to function as a single drive until the failed drive is replaced. Conceptually simple, RAID 1 is popular for those who require fault tolerance and don't need top-notch read performance. A variant of RAID 1 is duplexing, which duplicates the controller card as well as the drive, providing tolerance against failures of either a drive or a controller. It is much less commonly seen than straight mirroring.

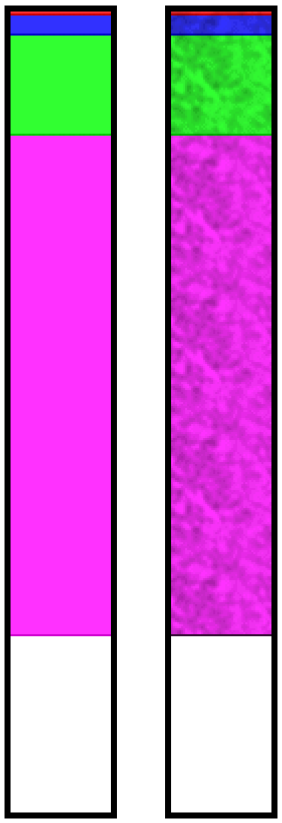


Illustration of a pair of mirrored hard disks, showing how the files are duplicated on both drives. (The files are the same as those in the RAID 0 illustration, except that to save space I have reduced the scale here so one vertical pixel represents 2 kiB.)

Controller Requirements: Supported by all hardware controllers, both SCSI and IDE/ATA, and also most software RAID solutions.

Hard Disk Requirements: Exactly two hard disks. Any type may be used but they should ideally be identical.

Array Capacity: Size of Smaller Drive.

Storage Efficiency: 50% if drives of the same size are used, otherwise (Size of Smaller Drive / (Size of Smaller Drive + Size of Larger Drive))

Fault Tolerance: Very good; duplexing even better.

Availability: Very good. Most RAID controllers, even low-end ones, will support hot sparing and automatic rebuilding of RAID 1 arrays.

Degradation and Rebuilding: Slight degradation of read performance; write performance will actually improve. Rebuilding is relatively fast.

Random Read Performance: Good. Better than a single drive but worse than many other RAID levels.

Random Write Performance: Good. Worse than a single drive, but better than many other RAID levels. :^)

Sequential Read Performance: Fair; about the same as a single drive.

Sequential Write Performance: Good; again, better than many other RAID levels.

Cost: Relatively high due to redundant drives; lowest storage efficiency of the single RAID levels. Duplexing is still more expensive due to redundant controllers. On the other hand, no expensive controller is required, and large consumer-grade drives are rather inexpensive these days, making RAID 1 a viable choice for an individual system.

Special Considerations: RAID 1 arrays are limited to the size of the drives used in the array. Multiple RAID 1 arrays can be set up if additional storage is required, but RAID 1+0 begins to look more attractive in that circumstance. Performance may be reduced if implemented using software instead of a hardware controller; duplexing may require software RAID and thus may show lower performance than mirroring.

Recommended Uses: Applications requiring high fault tolerance at a low cost, without heavy emphasis on large amounts of storage capacity or top performance. Especially useful in situations where the perception is that having a duplicated set of data is more secure than using parity. For this reason, RAID 1 is popular for accounting and other financial data. It is also commonly used for small database systems, enterprise servers, and for individual users requiring fault tolerance with a minimum of hassle and cost (since redundancy using parity generally requires more expensive hardware.)

Next: RAID Level 2

RAID Level 2

Common Name(s): RAID 2.

Technique(s) Used: Bit-level striping with Hamming code ECC.

Description: Level 2 is the "black sheep" of the RAID family, because it is the only RAID level that does not use one or more of the "standard" techniques of mirroring, striping and/or parity. RAID 2 uses something *similar* to striping with parity, but not the same as what is used by RAID levels 3 to 7. It is implemented by splitting data at the *bit* level and spreading it over a number of data disks and a number of redundancy disks. The redundant bits are calculated using Hamming codes, a form of error correcting code (ECC). Each time something is to be written to the array these codes are calculated and written along side the data to dedicated ECC disks; when the data is read back these ECC codes are read as well to confirm that no errors have occurred since the data was written. If a single-bit error occurs, it can be corrected "on the fly". If this sounds similar to the way that ECC is used *within* hard disks today, that's for a good reason: it's pretty much exactly the same. It's also the same concept used for ECC protection of system memory.

Level 2 is the only RAID level of the ones defined by the original Berkeley document that is not used today, for a variety of reasons. It is expensive and often requires many drives--see below for some surprisingly large numbers. The controller required was complex, specialized and expensive. The performance of RAID 2 is also rather substandard in transactional environments due to the bit-level striping. But most of all, level 2 was obviated by the use of ECC within a hard disk; essentially, much of what RAID 2 provides you now get for "free" within each hard disk, with other RAID levels providing protection above and *beyond* ECC.

Due to its cost and complexity, level 2 never really "caught on". Therefore, much of the information below is based upon theoretical analysis, not empirical evidence.

Controller Requirements: Specialized controller hardware required.

Hard Disk Requirements: Depends on exact implementation, but a typical setup required 10 data disks and 4 ECC disks for a total of 14, or 32 data disks and 7 ECC disks for a total of 39! The disks were spindle-synchronized to run in tandem.

Array Capacity: Depends on exact implementation but would be rather large if built today using modern drives.

Storage Efficiency: Depends on the number of data and ECC disks; for the 10+4 configuration, about 71%; for the 32+7 setup, about 82%.

Fault Tolerance: Only fair; for all the redundant drives included, you don't get much tolerance: only one drive can fail in this setup and be recoverable "on the fly".

Availability: Very good, due to "on the fly" error correction.

Degradation and Rebuilding: In theory, there would be little degradation due to failure of a single drive.

Random Read Performance: Fair. Bit-level striping makes multiple accesses impossible.

Random Write Performance: Poor, due to bit-level striping and ECC calculation overhead.

Sequential Read Performance: Very good, due to parallelism of many drives.

Sequential Write Performance: Fair to good.

Cost: Very expensive.

Special Considerations: Not used in modern systems.

Recommended Uses: Not used in modern systems.

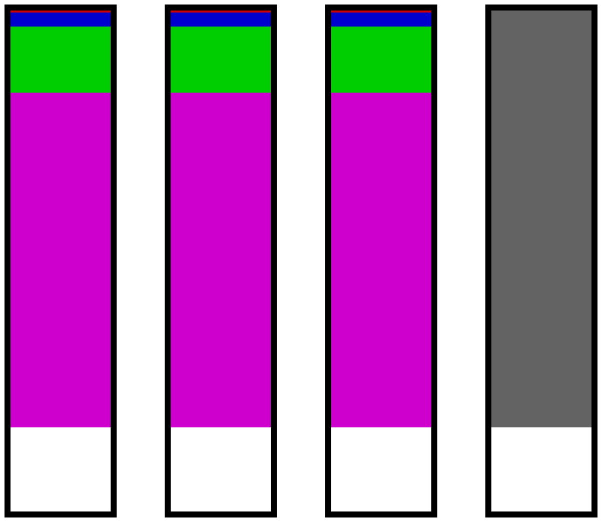
Next: RAID Level 3

RAID Level 3

Common Name(s): RAID 3. (Watch out for some companies that say their products implement RAID 3 when they are really RAID 4.)

Technique(s) Used: Byte-level striping with dedicated parity.

Description: Under RAID 3, data is striped across multiple disks at a byte level; the exact number of bytes sent in each stripe varies but is typically under 1024. The parity information is sent to a dedicated parity disk, but the failure of any disk in the array can be tolerated (i.e., the dedicated parity disk doesn't represent a single point of failure in the array.) The dedicated parity disk *does* generally serve as a performance bottleneck, especially for random writes, because it must be accessed any time anything is sent to the array; this is contrasted to distributed-parity levels such as RAID 5 which improve write performance by using distributed parity (though they still suffer from large overheads on writes, as described here). RAID 3 differs from RAID 4 only in the size of the stripes sent to the various disks.



This illustration shows how files of different sizes are distributed between the drives on a four-disk, byte-striped RAID 3 array. As with the RAID 0 illustration, the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB, with each vertical pixel representing 1 kiB of space. Notice that the files are evenly spread between three drives, with the fourth containing parity information (shown in dark gray). Since the blocks are so tiny in RAID 3, the individual boundaries between stripes can't be seen. You may want to compare this illustration to the one for RAID 4.

Controller Requirements: Generally requires a medium-to-high-end hardware RAID card.

Hard Disk Requirements: Minimum of three standard hard disks; maximum set by controller. Should be of identical size and type.

Array Capacity: (Size of Smallest Drive) * (Number of Drives - 1)

Storage Efficiency: If all drives are the same size, ((Number of Drives - 1) / Number of Drives).

Fault Tolerance: Good. Can tolerate loss of one drive.

Availability: Very good. Hot sparing and automatic rebuild are usually supported by controllers that implement RAID 3.

Degradation and Rebuilding: Relatively little degrading of performance if a drive fails. Rebuilds can take many hours.

Random Read Performance: Good, but not great, due to byte-level striping.

Random Write Performance: Poor, due to byte-level striping, parity calculation overhead, and the bottleneck of the dedicated parity drive.

Sequential Read Performance: Very good.

Sequential Write Performance: Fair to good.

Cost: Moderate. A hardware controller is usually required, as well as at least three drives.

Special Considerations: Not as popular as many of the other commonly-implemented RAID levels. For transactional environments, RAID 5 is usually a better choice.

Recommended Uses: Applications working with large files that require high transfer performance with redundancy, especially serving or editing large files: multimedia, publishing and so on. RAID 3 is often used for the same sorts of applications that would typically see the use of RAID 0, where the lack of fault tolerance of RAID 0 makes it unacceptable.

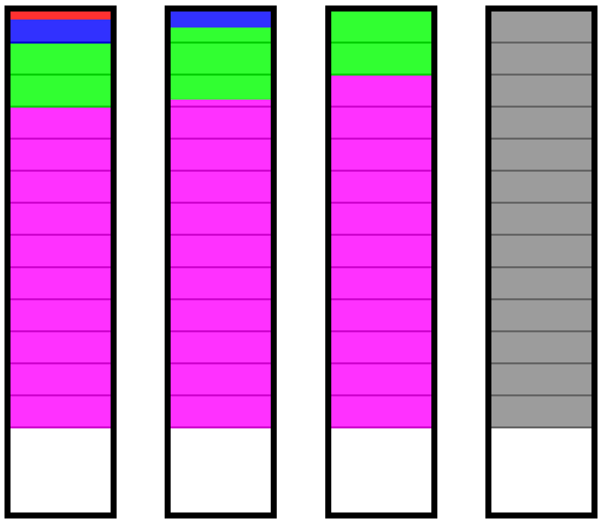
Next: RAID Level 4

RAID Level 4

Common Name(s): RAID 4 (sometimes called RAID 3 by the confused).

Technique(s) Used: Block-level striping with dedicated parity.

Description: RAID 4 improves performance by striping data across many disks in blocks, and provides fault tolerance through a dedicated parity disk. This makes it in some ways the "middle sibling" in a family of close relatives, RAID levels 3, 4 and 5. It is like RAID 3 except that it uses blocks instead of bytes for striping, and like RAID 5 except that it uses dedicated parity instead of distributed parity. Going from byte to block striping improves random access performance compared to RAID 3, but the dedicated parity disk remains a bottleneck, especially for random write performance. Fault tolerance, format efficiency and many other attributes are the same as for RAID 3 and RAID 5.



This illustration shows how files of different sizes are distributed between the drives on a four-disk RAID 4 array using a 16 kiB stripe size. As with the RAID 0 illustration, the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB, with each vertical pixel representing 1 kiB of space. Notice that as with RAID 3, the files are evenly spread between
three drives, with the fourth containing parity information (shown in gray). You may want to contrast this illustration to the one for RAID 3 (which is very similar except that the blocks are so tiny you can't see them) and the one for RAID 5 (which distributes the parity blocks across all four drives.)

Controller Requirements: Generally requires a medium-to-high-end hardware RAID card.

Hard Disk Requirements: Minimum of three standard hard disks; maximum set by controller. Should be of identical size and type.

Array Capacity: (Size of Smallest Drive) * (Number of Drives - 1).

Storage Efficiency: If all drives are the same size, ((Number of Drives - 1) / Number of Drives).

Fault Tolerance: Good. Can tolerate loss of one drive.

Availability: Very good. Hot sparing and automatic rebuild are usually supported..

Degradation and Rebuilding: Moderate degrading if a drive fails; potentially lengthy rebuilds.

Random Read Performance: Very good.

Random Write Performance: Poor to fair, due to parity calculation overhead and the bottleneck of the dedicated parity drive.

Sequential Read Performance: Good to very good.

Sequential Write Performance: Fair to good.

Cost: Moderate. A hardware controller is usually required, as well as at least three drives.

Special Considerations: Performance will depend to some extent upon the stripe size chosen.

Recommended Uses: Jack of all trades and master of none, RAID 4 is not as commonly used as RAID 3 and RAID 5, because it is in some ways a "compromise" between them that doesn't have a target market as well defined as either of those two levels. It is sometimes used by applications commonly seen using RAID 3 or RAID 5, running the gamut from databases and enterprise planning systems to serving large multimedia files.

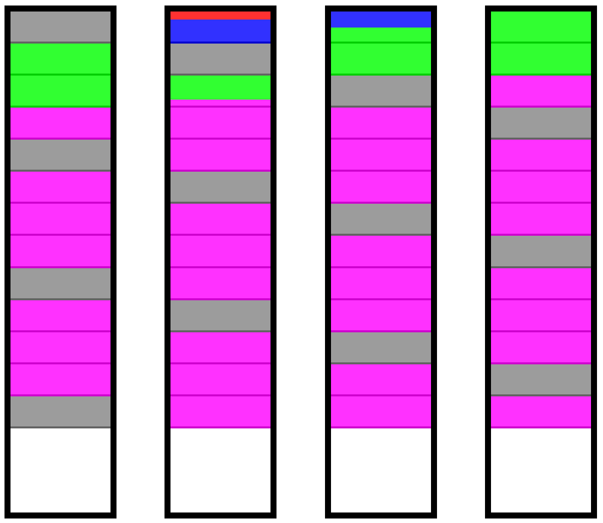
Next: RAID Level 5

RAID Level 5

Common Name(s): RAID 5.

Technique(s) Used: Block-level striping with distributed parity.

Description: One of the most popular RAID levels, RAID 5 stripes both data and parity information across three or more drives. It is similar to RAID 4 except that it exchanges the dedicated parity drive for a distributed parity algorithm, writing data and parity blocks across all the drives in the array. This removes the "bottleneck" that the dedicated parity drive represents, improving write performance slightly and allowing somewhat better parallelism in a multiple-transaction environment, though the overhead necessary in dealing with the parity continues to bog down writes. Fault tolerance is maintained by ensuring that the parity information for any given block of data is placed on a drive separate from those used to store the data itself. The performance of a RAID 5 array can be "adjusted" by trying different stripe sizes until one is found that is well-matched to the application being used.



This illustration shows how files of different sizes are distributed between the drives on a four-disk RAID 5 array using a 16 kiB stripe size. As with the RAID 0 illustration, the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB, with each vertical pixel representing 1 kiB of space. Contrast this diagram to the one for RAID 4, which is identical except that the data is only on three drives and the parity (shown in gray) is exclusively on the fourth.drive.

Controller Requirements: Requires a moderately high-end card for hardware RAID; supported by some operating systems for software RAID, but at a substantial performance penalty.

Hard Disk Requirements: Minimum of three standard hard disks; maximum set by controller. Should be of identical size and type.

Array Capacity: (Size of Smallest Drive) * (Number of Drives - 1).

Storage Efficiency: If all drives are the same size, ((Number of Drives - 1) / Number of Drives).

Fault Tolerance: Good. Can tolerate loss of one drive.

Availability: Good to very good. Hot sparing and automatic rebuild are usually featured on hardware RAID controllers supporting RAID 5 (software RAID 5 will require down-time).

Degradation and Rebuilding: Due to distributed parity, degradation can be substantial after a failure and during rebuilding.

Random Read Performance: Very good to excellent; generally better for larger stripe sizes. Can be better than RAID 0 since the data is distributed over one additional drive, and the parity information is not required during normal reads.

Random Write Performance: Only fair, due to parity overhead; this is improved over RAID 3 and RAID 4 due to eliminating the dedicated parity drive, but the overhead is still substantial.

Sequential Read Performance: Good to very good; generally better for smaller stripe sizes.

Sequential Write Performance: Fair to good.

Cost: Moderate, but often less than that of RAID 3 or RAID 4 due to its greater popularity, and especially if software RAID is used.

Special Considerations: Due to the amount of parity calculating required, software RAID 5 can *seriously* slow down a system. Performance will depend to some extent upon the stripe size chosen.

Recommended Uses: RAID 5 is seen by many as the ideal combination of good performance, good fault tolerance and high capacity and storage efficiency. It is best suited for transaction processing and is often used for "general purpose" service, as well as for relational database applications, enterprise resource planning and other business systems. For write-intensive applications, RAID 1 or RAID 1+0 are probably better choices (albeit higher in terms of hardware cost), as the performance of RAID 5 will begin to substantially decrease in a write-heavy environment.

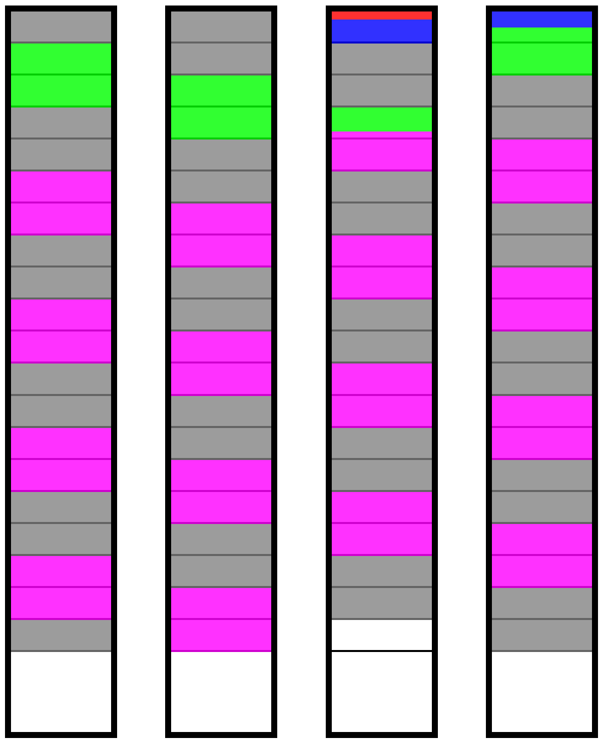
Next: RAID Level 6

RAID Level 6

Common Name(s): RAID 6. Some companies use the term "RAID 6" to refer to proprietary extensions of RAID 5; these are not discussed here.

Technique(s) Used: Block-level striping with *dual* distributed parity.

Description: RAID 6 can be thought of as "RAID 5, but more". It stripes blocks of data and parity across an array of drives like RAID 5, except that it calculates *two* sets of parity information for each parcel of data. The goal of this duplication is solely to improve fault tolerance; RAID 6 can handle the failure of any two drives in the array while other single RAID levels can handle at most one fault. Performance-wise, RAID 6 is generally slightly worse than RAID 5 in terms of writes due to the added overhead of more parity calculations, but may be slightly faster in random reads due to spreading of data over one more disk. As with RAID levels 4 and 5, performance can be adjusted by experimenting with different stripe sizes.



This illustration shows how files of different sizes are distributed between the drives on a four-disk RAID 6 array using a 16 kiB stripe size. As with the RAID 0 illustration, the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB, with each vertical pixel representing 1 kiB of space. This diagram is the same as the RAID 5 one, except that you'll notice that there is now twice as much gray parity information, and as a result, more space taken up on the four drives to contain the same data than the other levels that use striping.

Controller Requirements: Requires a specialized (usually meaning expensive) hardware controller.

Hard Disk Requirements: Minimum of four hard disks; maximum set by controller. Should be of identical size and type.

Array Capacity: (Size of Smallest Drive) * (Number of Drives - 2).

Storage Efficiency: If all drives are the same size, ((Number of Drives - 2) / Number of Drives).

Fault Tolerance: Very good to excellent. Can tolerate the simultaneous loss of any two drives in the array.

Availability: Excellent.

Degradation and Rebuilding: Due to the complexity of dual distributed parity, degradation can be substantial after a failure and during rebuilding. Dual redundancy may allow rebuilding to be delayed to avoid performance hit.

Random Read Performance: Very good to excellent; generally better for larger stripe sizes.

Random Write Performance: Poor, due to dual parity overhead and complexity.

Sequential Read Performance: Good to very good; generally better for smaller stripe sizes.

Sequential Write Performance: Fair.

Cost: High.

Special Considerations: Requires special implementation; not widely available.

Recommended Uses: In theory, RAID 6 is ideally suited to the same sorts of applications as RAID 5, but in situations where additional fault tolerance is required. In practice, RAID 6 has never really caught on because few companies are willing to pay for the extra cost to insure against a relatively rare event--it's unusual for two drives to fail simultaneously (unless something happens that takes out the entire array, in which case RAID 6 won't help anyway). On the lower end of the RAID 5 market, the rise of hot swapping and automatic rebuild features for RAID 5 have made RAID 6 even less desirable, since with these advanced features a RAID 5 array can recover from a single drive failure in a matter of hours (where without them, RAID 5 would require downtime for rebuilding, giving RAID 6 a substantial advantage.) On the higher end of the RAID 5 market, RAID 6 usually loses out to multiple RAID solutions such as RAID 10 that provide some degree of multiple-drive fault tolerance while offering improved performance as well.

Next: RAID Level 7

RAID Level 7

Common Name(s): RAID 7.

Technique(s) Used: Asynchronous, cached striping with dedicated parity.

Description: Unlike the other RAID levels, RAID 7 isn't an open industry standard; it is really a trademarked marketing term of Storage Computer Corporation, used to describe their proprietary RAID design. (I debated giving it a page alongside the other RAID levels, but since it *is* used in the market, it deserves to be explained; that said, information about it appears to be limited.) RAID 7 is based on concepts used in RAID levels 3 and 4, but greatly enhanced to address some of the limitations of those levels. Of particular note is the inclusion of a great deal of cache arranged into multiple levels, and a specialized real-time processor for managing the array asynchronously. This hardware support--especially the cache--allow the array to handle many simultaneous operations, greatly improving performance of all sorts while maintaining fault tolerance. In particular, RAID 7 offers much improved random read and write performance over RAID 3 or RAID 4 because the dependence on the dedicated parity disk is greatly reduced through the added hardware. The increased performance of RAID 7 of course comes at a cost. This is an expensive solution, made and supported by only one company.

Controller Requirements: Requires a specialized, expensive, proprietary controller.

Hard Disk Requirements: Depends on implementation.

Array Capacity: Depends on implementation.

Storage Efficiency: Depends on implementation.

Fault Tolerance: Very good.

Availability: Excellent, due to use of multiple hot spares.

Degradation and Rebuilding: Better than many RAID levels due to hardware support for parity calculation operations and multiple cache levels.

Random Read Performance: Very good to excellent. The extra cache can often supply the results of the read without needing to access the array drives.

Random Write Performance: Very good; substantially better than other single RAID levels doing striping with parity.

Sequential Read Performance: Very good to excellent.

Sequential Write Performance: Very good.

Cost: Very high.

Special Considerations: RAID 7 is a proprietary product of a single company; if it is of interest then you should contact Storage Computer Corporation for more details on the specifics of implementing it. All the caching creates potential vulnerabilities in the event of power failure, making the use of one or more UPS units mandatory.

Recommended Uses: Specialized high-end applications requiring absolutely top performance and willing to live with the limitations of a proprietary, expensive solution. For most users, a multiple RAID level solution like RAID 1+0 will probably yield comparable performance improvements over single RAID levels, at lower cost.

Next: Multiple (Nested) RAID Levels

Multiple (Nested) RAID Levels

The single RAID levels have distinct advantages and disadvantages, which is why most of them are used in various parts of the market to address different application requirements. It wasn't long after RAID began to be implemented that engineers looked at these RAID levels and began to wonder if it might be possible to get some of the advantages of more than one RAID level by designing arrays that use a combination of techniques. These RAID levels are called variously *multiple*, *nested*, or *multi*-RAID levels. They are also sometimes called *two-dimensional*, in reference to the two-dimensional schematics that are used to represent the application of two RAID levels to a set of disks, as you shall see.

Multiple RAID levels are most commonly used to improve performance, and they do this well. Nested RAID levels typically provide better performance characteristics than either of the single RAID levels that comprise them. The most commonly combined level is RAID 0, which is often mixed with redundant RAID levels such as 1, 3 or 5 to provide fault tolerance while exploiting the performance advantages of RAID 0. There is never a "free lunch", and so with multiple RAID levels what you pay is a cost in complexity: many drives are required, management and maintenance are more involved, and for some implementations a high-end RAID controller is required.

Not all combinations of RAID levels exist (which is good, because I'd get really bored of describing them all! :^)) Typically, the most popular multiple RAID levels are those that combine single RAID levels that complement each other with different strengths and weaknesses. Making a multiple RAID array marrying RAID 4 to RAID 5 wouldn't be the best idea, since they are so similar to begin with.

In this section I take a look at some of the more common multiple RAID levels. Note that some of the multiple RAID levels discussed here are frequently used, but others are rarely implemented. In particular, for completeness I describe both the "X+Y" and "Y+X" configurations of each multiple level, when in some cases only one or the other is commonly made into product. For example, I know that RAID 50 (5+0) is an option in commercial RAID controllers, but I am not sure if anyone makes a RAID 05 solution. There may also be other combinations of RAID levels that I am not aware of.

Next: RAID X+Y vs. RAID Y+X

RAID X+Y vs. RAID Y+X

Before looking at the specific multiple RAID levels, I have to explain a few things about the way multiple RAID levels are constructed. A multiple RAID level is generally created by taking a number of disks and dividing them into sets. Within each set a single RAID level is applied to form a number of arrays. Then, the second RAID level is applied to the arrays to create a higher-level array. This is why these are sometimes called *nested* arrays.

Since there are two levels, there are two ways they can be combined. The choice of which level is applied first and which second has an impact on some important array characteristics. Let's take as an example multiple RAID employing RAID 0 and RAID 1 to create an array of ten disks. Much as we can define 10 to be 2*5 or 5*2, we can create our multiple RAID array two ways:

- **RAID 0, then RAID 1:** Divide the ten disks into two sets of five. Turn each set into a RAID 0 array containing five disks, then mirror the two arrays. (Sometimes called a "mirror of stripes".)
- **RAID 1, then RAID 0:** Divide the ten disks into five sets of two. Turn each set into a RAID 1 array, then stripe across the five mirrored sets. (A "stripe of mirrors").

Naming conventions for multiple RAID levels are *just horrible*. The standard that *most* of the industry seems to use is that if RAID level X is applied first and then RAID level Y is applied over top of it, that is RAID "X+Y", also sometimes seen as "RAID XY" or "RAID X/Y". This would mean that alternative number 1 above would be called "RAID 0+1" or "RAID 1+0", and that's in fact the terminology that most companies use. Unfortunately, other companies reverse the terms! They might call the RAID 0 and then RAID 1 technique "RAID 1/0" or "RAID 10" (perhaps out of fear that people would think "RAID 01" and "RAID 1" were the same thing). Some designers use the terms "RAID 01" and "RAID 10" interchangeably. The result of all this confusion is that you *must investigate* to determine what exactly a company is implementing when you look at multiple RAID. Don't trust the label.

Of course, I haven't even explained why you should care about the distinctions, so I suppose I should do that. :^) After all, if you have ten marbles, why would you care if they are arranged in five columns and two rows, or two columns and five rows? Same here: aren't ten disks in an array ten disks in an array? Clearly I am setting you up, so the answer is obviously "no". :^)

In many respects, there *is* no difference between them: there is no impact on drive requirements, capacity, storage efficiency, and importantly, not much impact on performance. The big difference comes into play when we look at *fault tolerance*. Most controllers implement multiple level RAID by forming a "super array" comprised of "sub-arrays" underneath it. In many cases the arrays that comprise the "super array"--often called *sets*--are considered to be logical "single units", which means that the controller only considers one of these "single units" to either be "up" or "down" as a whole. It will make use of redundancy features *within* a sub-array, but not *between* sub-arrays, even if the higher-level array means that drives in different sub-arrays will have the same data.

That makes this sound much more complicated than it really is; it's much easier to explain with an example. Let's look at 10 drives and RAID 0+1 vs. RAID 1+0 again:

- **RAID 0+1:** We stripe together drives 1, 2, 3, 4 and 5 into RAID 0 stripe set "A", and drives 6, 7, 8, 9 and 10 into RAID 0 stripe set "B". We then mirror A and B using RAID 1. If one drive fails, say drive #2, then the entire stripe set "A" is lost, because RAID 0 has no redundancy; the RAID 0+1 array continues to chug along because the entire stripe set "B" is still functioning. However, at this point you are reduced to running what is in essence a straight RAID 0 array until drive #2 can be fixed. If in the meantime drive #9 goes down, you lose the entire array.

- **RAID 1+0:** We mirror drives 1 and 2 to form RAID 1 mirror set "A"; 3 and 4 become "B"; 5 and 6 become "C"; 7 and 8 become "D"; and 9 and 10 become "E". We then do a RAID 0 stripe across sets A through E. If drive #2 fails now, only mirror set "A" is affected; it still has drive #1 so it is fine, and the RAID 1+0 array continues functioning. If while drive #2 is being replaced drive #9 fails, the array is fine, because drive #9 is in a different mirror pair from #2. Only two failures in the same mirror set will cause the array to fail, so in theory, five drives can fail--as long as they are all in different sets--and the array would still be fine.

Clearly, RAID 1+0 is more robust than RAID 0+1. Now, if the controller running RAID 0+1 were smart, when drive #2 failed it would continue striping to the other four drives in stripe set "A", and if drive #9 later failed it would "realize" that it could use drive #4 in its stead, since it should have the same data. This functionality would theoretically make RAID 0+1 just as fault-tolerant as RAID 1+0. Unfortunately, most controllers *aren't* that smart. It pays to ask specific questions about how a multiple RAID array implementation handles multiple drive failures, but in general, a controller won't swap drives between component sub-arrays unless the manufacturer of the controller specifically says it will.

The same impact on fault tolerance applies to rebuilding. Consider again the example above. In RAID 0+1, if drive #2 fails, the data on five hard disks will need to be rebuilt, because the whole stripe set "A" will be wiped out. In RAID 1+0, only drive #2 has to be rebuilt. Again here, the advantage is to RAID 1+0.

Tip: For a diagram showing graphically the difference between RAID 0+1 and RAID 1+0, see the page discussing those levels.

Some controllers offer the choice of "what order" you use to set up a multiple RAID array; they will let you do either RAID X+Y or RAID Y+X, as you choose. Others will force you to use only one or the other configuration. Again, ask for details when considering a solution, and be specific with your questions so you can figure out what exactly the controller you are investigating does.

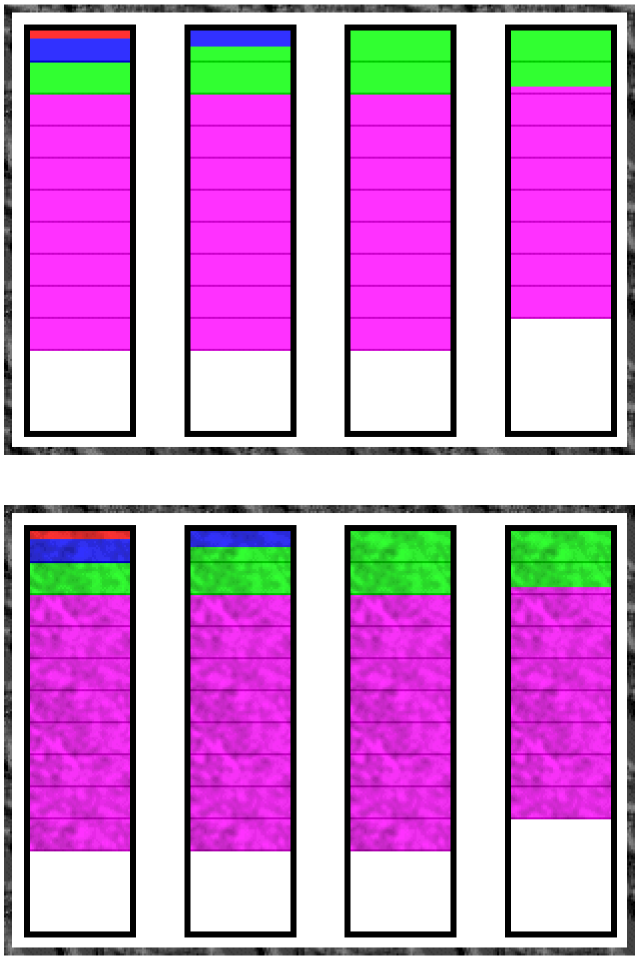
Next: RAID Levels 0+1 (01) and 1+0 (10)

RAID Levels 0+1 (01) and 1+0 (10)

Common Name(s): RAID 0+1, 01, 0/1, "mirrored stripes", "mirror of stripes"; RAID 1+0, 10, 1/0, "striped mirrors", "stripe of mirrors". Labels are often used incorrectly; verify the details of the implementation if the distinction between 0+1 and 1+0 is important to you.

Technique(s) Used: Mirroring and striping without parity.

Description: The most popular of the multiple RAID levels, RAID 01 and 10 combine the best features of striping and mirroring to yield large arrays with high performance in most uses and superior fault tolerance. RAID 01 is a mirrored configuration of two striped sets; RAID 10 is a stripe across a number of mirrored sets. RAID 10 and 01 have been increasing dramatically in popularity as hard disks become cheaper and the four-drive minimum is legitimately seen as much less of an obstacle. RAID 10 provides better fault tolerance and rebuild performance than RAID 01. Both array types provide very good to excellent overall performance by combining the speed of RAID 0 with the redundancy of RAID 1 without requiring parity calculations.



This illustration shows how files of different sizes are distributed between the drives on an eight-disk RAID 0+1 array using a 16 kiB stripe size for the RAID 0
portion. As with the RAID 0 illustration, the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB, with each vertical

pixel
representing 1 kiB of space. The large, patterned rectangles represent the two
RAID 0 "sub arrays", which are mirrored using RAID 1 to create RAID 0+1. The contents of the striped sets are thus identical. The diagram for RAID 1+0 would be the same except for the groupings: instead of two large boxes dividing
the drives horizontally, there would be four large boxes dividing the drives vertically into mirrored pairs. These pairs would then be striped together to form level 1+0. Contrast this diagram to the ones for RAID 0 and RAID 1.

Controller Requirements: Almost all hardware controllers will support one or the other of RAID 10 or RAID 01, but often not both. Even low-end cards will support this multiple level, usually RAID 01. High-end cards may support both 01 and 10.

Hard Disk Requirements: An even number of hard disks with a minimum of four; maximum dependent on controller. All drives should be identical.

Array Capacity: (Size of Smallest Drive) * (Number of Drives) / 2.

Storage Efficiency: If all drives are the same size, 50%.

Fault Tolerance: Very good for RAID 01; excellent for RAID 10.

Availability: Very good for RAID 01; excellent for RAID 10.

Degradation and Rebuilding: Relatively little for RAID 10; can be more substantial for RAID 01.

Random Read Performance: Very good to excellent.

Random Write Performance: Good to very good.

Sequential Read Performance: Very good to excellent.

Sequential Write Performance: Good to very good.

Cost: Relatively high due to large number of drives required and low storage efficiency (50%).

Special Considerations: Low storage efficiency limits potential array capacity.

Recommended Uses: Applications requiring both high performance and reliability and willing to sacrifice capacity to get them. This includes enterprise servers, moderate-sized database systems and the like at the high end, but also individuals using larger IDE/ATA hard disks on the low end. Often used in place of RAID 1 or RAID 5 by those requiring higher performance; may be used instead of RAID 1 for applications requiring more capacity.

Next: RAID Levels 0+3 (03 or 53) and 3+0 (30)

RAID Levels 0+3 (03 or 53) and 3+0 (30)

Common Name(s): The most confusing naming of any of the RAID levels. :^) In an ideal world, this level would be named RAID 0+3 (or 03) or RAID 3+0 (30). Instead, the number *53* is often used in place of 03 for reasons I have never been able to determine, and worse, 53 is often actually implemented as 30, not 03. As always, verify the details of the implementation to be sure of what you have.

Technique(s) Used: Byte striping with dedicated parity combined with block striping.

Description: RAID 03 and 30 (though often called 53 for a reason that utterly escapes me) combine byte striping, parity and block striping to create large arrays that are conceptually difficult to understand. :^) RAID 03 is formed by putting into a RAID 3 array a number of striped RAID 0 arrays; RAID 30 is more common and is formed by striping across a number of RAID 3 sub-arrays. The combination of parity, small-block striping and large-block striping makes analyzing the theoretical performance of this level difficult. In general, it provides performance better than RAID 3 due to the addition of RAID 0 striping, but closer to RAID 3 than RAID 0 in overall speed, especially on writes. RAID 30 provides better fault tolerance and rebuild performance than RAID 03, but both depend on the "width" of the RAID 3 dimension of the drive relative to the RAID 0 dimension: the more parity drives, the lower capacity and storage efficiency, but the greater the fault tolerance. See the examples below for more explanation of this.

Most of the characteristics of RAID 0+3 and 3+0 are similar to those of RAID 0+5 and 5+0. RAID 30 and 03 tend to be better for large files than RAID 50 and 05.

Controller Requirements: Generally requires a high-end hardware controller.

Hard Disk Requirements: Number of drives must be able to be factored into two integers, one of which must be 2 or higher and the other 3 or higher (you can make a RAID 30 array from 10 drives but not 11). Minimum number of drives is six, with the maximum set by the controller.

Array Capacity: For RAID 03: (Size of Smallest Drive) * (Number of Drives In Each RAID 0 Set) * (Number of RAID 0 Sets - 1). For RAID 30: (Size of Smallest Drive) * (Number of Drives In Each RAID 3 Set - 1) * (Number of RAID 3 Sets).

For example, the capacity of a RAID 03 array made of 15 18 GB drives arranged as three five-drive RAID 0 sets would be 18 GB * 5 * (3-1) = 180 GB. The capacity of a RAID 30 array made of 21 18 GB drives arranged as three seven-drive RAID 3 sets would be 18 GB * (7-1) * 3 = 324 GB. The same 21 drives arranged as seven three-drive RAID 3 sets would have a capacity of 18 GB * (3-1) * 7 = "only" 252 GB.

Storage Efficiency: For RAID 03: ((Number of RAID 0 Sets - 1) / Number of RAID 0 Sets). For RAID 30: ((Number of Drives In Each RAID 3 Set - 1) / Number of Drives In Each RAID 3 Set).

Taking the same examples as above, the 15-drive RAID 03 array would have a storage efficiency of (3-1)/3 = 67%. The first RAID 30 array, configured as three seven-drive RAID 3 sets, would have a storage efficiency of (7-1)/7 = 86%, while the other RAID 30 array would have a storage efficiency of, again, (3-1)/3 = 67%.

Fault Tolerance: Good to very good, depending on whether it is RAID 03 or 30, and the number of parity drives relative to the total number. RAID 30 will provide better fault tolerance than RAID 03.

Consider the two different 21-drive RAID 30 arrays mentioned above: the first one (three seven-drive RAID 3 sets) has higher capacity and storage efficiency, but can only tolerate three maximum potential drive failures; the one with lower capacity and storage efficiency (seven three-drive RAID 3 sets) can handle as many as seven , if they are in different RAID 3 sets. Of course few applications really require tolerance for seven independent drive failures! And of course, if those 21 drives were in a RAID 03 array instead, failure of a second drive after one had failed and taken down one of the RAID 0 sub-arrays would crash the entire array.

Availability: Very good to excellent.

Degradation and Rebuilding: Relatively little for RAID 30 (though more than RAID 10); can be more substantial for RAID 03.

Random Read Performance: Very good, assuming RAID 0 stripe size is reasonably large.

Random Write Performance: Fair.

Sequential Read Performance: Very good to excellent.

Sequential Write Performance: Good.

Cost: Relatively high due to requirements for a hardware controller and a large number of drives; storage efficiency is better than RAID 10 however and no worse than any other RAID levels that include redundancy.

Special Considerations: Complex and expensive to implement.

Recommended Uses: Not as widely used as many other RAID levels. Applications include data that requires the speed of RAID 0 with fault tolerance and high capacity, such as critical multimedia data and large database or file servers. Sometimes used instead of RAID 3 to increase capacity as well as performance.

Next: RAID Levels 0+5 (05) and 5+0 (50)

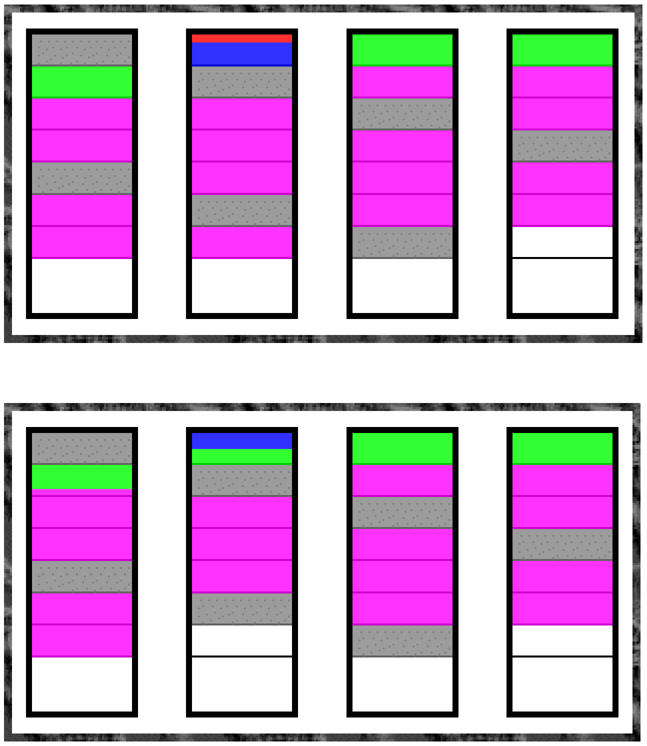
RAID Levels 0+5 (05) and 5+0 (50)

Common Name(s): RAID 0+5 or 05; RAID 5+0 or 50. As with the other multiple RAID levels, verify the exact implementation instead of relying on the label.

Technique(s) Used: Block striping with distributed parity combined with block striping.

Description: RAID 05 and 50 form large arrays by combining the block striping and parity of RAID 5 with the straight block striping of RAID 0. RAID 05 is a RAID 5 array comprised of a number of striped RAID 0 arrays; it is less commonly seen than RAID 50, which is a RAID 0 array striped across RAID 5 elements. RAID 50 and 05 improve upon the performance of RAID 5 through the addition of RAID 0, particularly during writes. It also provides better fault tolerance than the single RAID level does, especially if configured as RAID 50.

Most of the characteristics of RAID 05 and 50 are similar to those of RAID 03 and 30. RAID 50 and 05 tend to be preferable for transactional environments with smaller files than 03 and 30.



This illustration shows how files of different sizes are distributed between the drives on an eight-disk RAID 5+0 array using a 16 kiB stripe size. As with the RAID 0 illustration, the red file is 4 kiB in size; the blue is 20 kiB; the green is 100 kiB; and the magenta is 500 kiB, with each vertical pixel representing 1 kiB of space. Each of the large, patterned rectangles represents a four-drive RAID 5 array. The data is evenly striped between these two RAID 5 arrays using RAID 0. Then within each RAID 5 array, the data is stored using striping with parity. So the first small file, and 12 kiB of the second file, were sent to the top RAID 5 array; the remaining 8 kiB of the second file and the first 8 kiB of the 100 kiB file went to the bottom RAID 5 array; then the next 16 kiB of the 100 kiB went to the top array, and so on. Within each RAID 5 array the data is striped and parity calculated just like a regular RAID 5 array; each array just does this with half the number of blocks it normally would. Contrast this diagram to the ones for RAID 0 and RAID 5.

Controller Requirements: Generally requires a high-end hardware controller.

Hard Disk Requirements: Number of drives must be able to be factored into two integers, one of which must be 2 or higher and the other 3 or higher (you can make a RAID 30 array from 6 drives but not 7). Minimum number of drives is six, with the maximum set by the controller.

Array Capacity: Same as RAID 03 and 30. For RAID 05: (Size of Smallest Drive) * (Number of Drives In Each RAID 0 Set) * (Number of RAID 0 Sets -

1). For RAID 50: (Size of Smallest Drive) * (Number of Drives In Each RAID 5 Set - 1) * (Number of RAID 5 Sets).

For example, the capacity of a RAID 05 array made of 15 18 GB drives arranged as three five-drive RAID 0 sets would be 18 GB * 5 * (3-1) = 180 GB. The capacity of a RAID 50 array made of 21 18 GB drives arranged as three seven-drive RAID 5 sets would be 18 GB * (7-1) * 3 = 324 GB. The same 21 drives arranged as seven three-drive RAID 5 sets would have a capacity of 18 GB * (3-1) * 7 = 252 GB.

Storage Efficiency: Same as for RAID 03 and 30. For RAID 05: ((Number of RAID 0 Sets - 1) / Number of RAID 0 Sets). For RAID 50: ((Number of Drives In Each RAID 5 Set - 1) / Number of Drives In Each RAID 5 Set).

Taking the same examples as above, the 15-drive RAID 05 array would have a storage efficiency of (3-1)/3 = 67%. The first RAID 50 array, configured as three seven-drive RAID 5 sets, would have a storage efficiency of (7-1)/7 = 86%, while the other RAID 50 array would have a storage efficiency of (3-1)/3 = 67%.

Fault Tolerance: Same as for RAID 03 and 30. Good to very good, depending on whether it is RAID 05 or 50, and the number of parity drives relative to the total number. RAID 50 will provide better fault tolerance than RAID 05.

Consider the two different 21-drive RAID 50 arrays mentioned above: the first one (three seven-drive RAID 5 sets) has higher capacity and storage efficiency, but can only tolerate three maximum potential drive failures; the one with lower capacity and storage efficiency (seven three-drive RAID 5 sets) can handle as many as seven , if they are in different RAID 5 sets. Of course few applications really require tolerance for seven independent drive failures! And of course, if those 21 drives were in a RAID 05 array instead, failure of a second drive after one had failed and taken down one of the RAID 0 sub-arrays would crash the entire array.

Availability: Very good to excellent.

Degradation and Rebuilding: Moderate for RAID 50; worse for RAID 05.

Random Read Performance: Very good to excellent.

Random Write Performance: Good.

Sequential Read Performance: Very good.

Sequential Write Performance: Good.

Cost: Relatively high due to requirements for a hardware controller and a large number of drives; storage efficiency is better than RAID 10 however and no worse than any other RAID levels that include redundancy.

Special Considerations: Complex and expensive to implement.

Recommended Uses: Applications that require high fault tolerance, capacity and random positioning performance. Not as widely used as many other RAID levels. Sometimes used instead of RAID 5 to increase capacity. Sometimes used for large databases.

Next: RAID Levels 1+5 (15) and 5+1 (51)

RAID Levels 1+5 (15) and 5+1 (51)

Common Name(s): RAID 1+5 or 15; RAID 5+1 or 51. "Common" is a bit of a stretch with this level, as it is less common than probably any other, so it's important to verify the details of each implementation.

Technique(s) Used: Mirroring (or duplexing) combined with block striping with distributed parity.

Description: RAID 1+5 and 5+1 might be sarcastically called "the RAID levels for the truly paranoid". :^) The only configurations that use both redundancy methods, mirroring and parity, this "belt and suspenders" technique is designed to maximize fault tolerance and availability, at the expense of just about everything else. A RAID 15 array is formed by creating a striped set with parity using multiple mirrored pairs as components; it is similar in concept to RAID 10 except that the striping is done with parity. Similarly, RAID 51 is created by mirroring entire RAID 5 arrays and is similar to RAID 01 except again that the sets are RAID 5 instead of RAID 0 and hence include parity protection. Performance for these arrays is good but not very high for the cost involved, nor relative to that of other multiple RAID levels.

The fault tolerance of these RAID levels is truly amazing; an eight-drive RAID 15 array can tolerate the failure of any *three* drives simultaneously; an eight-drive RAID 51 array can also handle three and even as many as *five*, as long as at least one of the mirrored RAID 5 sets has no more than one failure! The price paid for this resiliency is complexity and cost of implementation, and very low storage efficiency.

The RAID 1 component of this nested level may in fact use duplexing instead of mirroring to add even more fault tolerance.

Controller Requirements: Requires at least a high-end controller; may in fact require multiple systems and/or specialized hardware or software. RAID 51 is sometimes implemented as a "hybrid" of hardware and software RAID, by doing software mirroring at the operating system level over a pair of RAID 5 controllers, thus implementing duplexing for even higher fault tolerance. In theory this could be done with Windows NT software mirroring and a pair of hardware RAID 5 cards, if you could set it all up to work together properly.

Hard Disk Requirements: An even number of hard disks with a minimum of six; maximum dependent on controller. All drives should be identical.

Array Capacity: (Size of Smallest Drive) * ((Number of Drives / 2) - 1). So an array with ten 18 GB drives would have a capacity of 18 GB * ((10/2) - 1) = just 72 GB.

Storage Efficiency: Assuming all drives are the same size, ((Number of Drives / 2) - 1) / (Number of Drives). In the example above, efficiency is 40%. This is the worst storage efficiency of any RAID level; a six-drive RAID 15 or 51 array would have a storage efficiency of just 33%!

Fault Tolerance: Excellent; by far the best of any level.

Availability: Excellent.

Degradation and Rebuilding: Can be substantial.

Random Read Performance: Very good.

Random Write Performance: Good.

Sequential Read Performance: Very good.

Sequential Write Performance: Good.

Cost: Very high. An uncommon solution requiring a lot of storage devices for relatively low capacity, and possibly additional hardware or software.

Special Considerations: Complex and very expensive to implement.

Recommended Uses: Critical applications requiring very high fault tolerance. In my opinion, if you get to the point of needing this much fault tolerance this badly, you should be looking beyond RAID to remote mirroring, clustering or other redundant server setups; RAID 10 provides most of the benefits with better performance and lower cost. Not widely implemented.

Next: "Just A Bunch Of Disks"

"Just A Bunch Of Disks" (JBOD)

If you have some disks in a system that you decide not to configure into a RAID array, what do you do with them? Traditionally, they are left to act as independent drive volumes within the system, and that's how many people in fact use two, three or more drives in a PC. In some applications, however, it is desirable to be able to use all these disks as if they were one single volume. The proper term for this is *spanning*; the pseudo-cutesy term for it, clearly chosen to contrast against "redundant array of inexpensive disks", is *Just A Bunch Of Disks* or *JBOD*. How frightfully clever.

JBOD isn't really RAID at all, but I discuss it here since it is sort of a "third cousin" of RAID... JBOD can be thought of as the opposite of partitioning: while partitioning chops single drives up into smaller logical volumes, JBOD combines drives into larger logical volumes. It provides no fault tolerance, nor does it provide any improvements in performance compared to the independent use of its constituent drives. (In fact, it arguably hurts performance, by making it more difficult to use the underlying drives concurrently, or to optimize different drives for different uses.)

When you look at it, JBOD doesn't really have a lot to recommend it. It still requires a controller card or software driver, which means that almost any system that can do JBOD can also do RAID 0, and RAID 0 has significant performance advantages over JBOD. Neither provide fault tolerance, so that's a wash. There are only two possible advantages of JBOD over RAID 0:

- **Avoiding Drive Waste:** If you have a number of odd-sized drives, JBOD will let you combine them into a single unit without loss of any capacity; a 10 GB drive and 30 GB would combine to make a 40 GB JBOD volume but only a 20 GB RAID 0 array. This may be an issue for those expanding an existing system, though with drives so cheap these days it's a relatively small advantage.
- **Easier Disaster Recovery:** If a disk in a RAID 0 volume dies, the data on every disk in the array is essentially destroyed because all the files are striped; if a drive in a JBOD set dies then it may be easier to recover the files on the other drives (but then again, it might not, depending on how the operating system manages the disks.) Considering that you should be doing regular backups regardless, and that even under JBOD recovery can be difficult, this too is a minor advantage.

Note: Some companies use the term "spanning" when they really mean striping, so watch out for that!

Next: Summary Comparison of RAID Levels

Summary Comparison of RAID Levels

Below you will find a table that summarizes the key quantitative attributes of the various RAID levels for easy comparison. For the full details on any RAID level, see its own page, accessible here. For a description of the different characteristics, see the discussion of factors differentiating RAID levels. Also be sure to read the notes that follow the table:

RAID Level	**Number of Disks**	**Capacity**	**Storage Efficiency**	**Fault Tolerance**	**Availability**	**Random Read Perf**
0	2,3,4,...	S*N	100%	none	★	★★★★
1	2	S*N/2	50%	★★★★	★★★★	★★★
2	many	varies, large	~ 70-80%	★★	★★★★	★★
3	3,4,5,...	S*(N-1)	(N-1)/N	★★★	★★★★	★★★
4	3,4,5,...	S*(N-1)	(N-1)/N	★★★	★★★★	★★★★
5	3,4,5,...	S*(N-1)	(N-1)/N	★★★	★★★★	★★★★½
6	4,5,6,...	S*(N-2)	(N-2)/N	★★★★½	★★★★★	★★★★½
7	varies	varies	varies	★★★	★★★★	★★★★½
01/10	4,6,8,...	S*N/2	50%	★★★★	★★★★½	★★★★½
03/30	6,8,9,10,...	S*N0*(N3-1)	(N3-1)/N3	★★★½	★★★★	★★★★
05/50	6,8,9,10,...	S*N0*(N5-1)	(N5-1)/N5	★★★½	★★★★	★★★★½
15/51	6,8,10,...	S*((N/2)-1)	((N/2)-1)/N	★★★★★	★★★★★	★★★★
0	**Random Write Perf**	**Sequential Read Perf**	**Sequential Write Perf**	**Cost**		
1	★★★★	★★★★½	★★★★	$		
2	★★★	★★	★★★	$$		
3	★	★★★★	★★½	$$$$$		
4	★	★★★★	★★½	$$		
5	★½	★★★	★★	$$		
6	★★	★★★½	★★½	$$		

7	★	★★★½	★★	$$$		
01/10	★★★★	★★★★½	★★★★	$$$$$		
03/30	★★★½	★★★★½	★★★½	$$$		
05/50	★★	★★★★½	★★★	$$$$		
15/51	★★★	★★★★	★★★	$$$$		
	★★★	★★★★	★★★	$$$$$		

Notes on the table:

- For the number of disks, the first few valid sizes are shown; you can figure out the rest from the examples given in most cases. Minimum size is the first number shown; maximum size is normally dictated by the controller. RAID 01/10 and RAID 15/51 must have an even number of drives, minimum 6. RAID 03/30 and 05/50 can only have sizes that are a product of integers, minimum 6.
- For capacity and storage efficiency, "S" is the size of the smallest drive in the array, and "N" is the number of drives in the array. For the RAID 03 and 30, "N0" is the width of the RAID 0 dimension of the array, and "N3" is the width of the RAID 3 dimension. So a 12-disk RAID 30 array made by creating three 4-disk RAID 3 arrays and then striping them would have N3=4 and N0=3. The same applies for "N5" in the RAID 05/50 row.
- Storage efficiency assumes all drives are of identical size. If this is not the case, the universal computation (array capacity divided by the sum of all drive sizes) must be used.
- Performance rankings are approximations and to some extent, reflect my personal opinions. *Please* don't over-emphasize a "half-star" difference between two scores!
- Cost is relative and approximate, of course. In the real world it will depend on many factors; the dollar signs are just intended to provide some perspective.

Next: RAID Configuration and Implementation

RAID Configuration and Implementation

You've probably heard this phrase before: "The devil is in the details". Well, this probably applies to the computing field as much as any other, if not more, and it certainly applies to RAID. All the discussion of RAID concepts, levels, fault tolerance and performance are made with the implied assumption that the RAID array will be properly configured and implemented. If this is not done, you can quickly discover just how wide the gulf is between theory and the "real world". :^)

Though perhaps not as interesting to read about as RAID levels, information about the different ways to set up RAID arrays and the various constraints and issues involved in implementing an array are no less important. In this section I will cover some of the topics related to actually creating a RAID array. This includes a look at hardware and software RAID, RAID controllers, interfacing issues, drive selection criteria, and RAID management. I also explore some advanced RAID features that can be critically important when setting up a high-availability array, such as hot spares and drive swapping.

Next: RAID Controllers and Controller Features

RAID Controllers and Controller Features

While the RAID level of an array gets a lot of attention, and the drives used in the array do too, the RAID *controller* often does not. This is unfortunate, because in many ways your choices for implementing a RAID array are dependent on the type of controller you use. The controller and related hardware also have a very important impact on array capacity and performance.

In this section I take a look at the different options available for controlling a RAID array, examining in particular the important matter of "hardware RAID vs. software RAID". I also explain related issues such as the different interfaces that can be used for RAID implementations, the use of multiple channels for increasing throughput, and also some operating system concerns.

Next: Hardware RAID

Hardware RAID

Most "serious" RAID implementations use what is termed *hardware RAID*. This means using dedicated hardware to control the array, as opposed to doing array control processing via software. Good hardware controllers are in many ways like miniature computers, incorporating dedicated processors that exceed the power of processors that ran entire PCs just a few years ago. For a list contrasting software and hardware RAID, showing the advantages and disadvantages of each, see this page on software RAID.

There are two main types of hardware RAID, differing primarily in how they interface the array to the system:

- **Bus-Based or Controller Card Hardware RAID:** This is the more conventional type of hardware RAID, and the type most commonly used, particularly for lower-end systems. A specialized RAID controller is installed into the PC or server, and the array drives are connected to it. It essentially takes the place of the SCSI host adapter or IDE/ATA controller that would normally be used for interfacing between the system and the hard disks; it interfaces to the drives using SCSI or IDE/ATA, and sends data to the rest of the PC over the system bus (typically PCI). Some motherboards, particularly those intended for server systems, come with some variant of *integrated* RAID controller. These are built into the motherboard, but function in precisely the same manner as an add-in bus-based card. (This is analogous to the way that the integrated IDE/ATA controllers on all modern motherboards function the same way that add-in IDE/ATA controllers once did on older systems.) The only difference is that integrated controllers can reduce overall cost--at the price of flexibility.
- **Intelligent, External RAID Controller:** In this higher-end design, the RAID controller is removed completely from the system to a separate box. Within the box the RAID controller manages the drives in the array, typically using SCSI, and then presents the logical drives of the array over a standard interface (again, typically a variant of SCSI) to the server using the array. The server sees the array or arrays as just one or more very fast hard disks; the RAID is completely hidden from the machine. In essence, one of these units really *is* an entire computer unto itself, with a dedicated processor that manages the RAID array and acts as a conduit between the server and the array.

A PCI-bus-based, IDE/ATA hard disk RAID controller, supporting levels 0, 1, and 01. (Promise's popular FastTrak 66.)

Original image © Promise Technology Inc.
Image used with permission.

Bus-based RAID is cheaper and much simpler to implement than external RAID controllers while still offering often impressive capabilities; they range from entry-level cards for IDE/ATA systems that cost around $100, up to top-of-the-line, full-featured devices costing several thousand dollars. Dedicated, external RAID controller systems are still more expensive but offer many advanced features, are typically more expandable than bus-based RAID implementations (offering support for large array well into the terabytes) and can offer better performance. They often cost well into the five figures, so they are not something a typical PC user would even consider.

Note: External RAID controllers should not be confused with external RAID *enclosures*. Enclosures provide power and physical infrastructure for the drives in a RAID array, but not the smarts of the controller; they are functionally a large, fancy PC system case. An external RAID controller can be thought of as such an enclosure combined with a high-end, integrated controller as well.

In most cases, the decision to use hardware RAID is made almost exclusively on financial grounds: hardware RAID is superior to software RAID in virtually every way, it just costs more. If you want to use any of the more esoteric RAID levels such as RAID 3 or RAID 1+0, you pretty much require hardware RAID, since support for these levels is usually not offered in software. If you need top performance while using a computation-intensive RAID level such as RAID 5, you also should consider a hardware solution pretty much "mandatory", because software RAID 5 can really hurt performance.

Next: Software RAID

Software RAID

Software RAID is just like hardware RAID, except that it uses software instead of hardware. (There, *that* was easy! And here I thought this section would be hard to write. :^))

All kidding aside, that pretty much *is* what software RAID is about. Instead of using a dedicated hardware controller to perform the various functions required to implement a RAID array, these functions are performed by the system processor using special software routines. Since array management is a low-level activity that must be performed "underneath" the other software that runs on the PC, software RAID usually is implemented at the operating system level. Windows NT and Windows 2000, as well as most of the various flavors of UNIX, support some RAID levels in software.

There are a few advantages of using software RAID over hardware RAID, but more disadvantages. First, let's look at the pros of software RAID:

- **Cost:** If you are already running an operating system that supports software RAID, you have no additional costs for controller hardware; you may need to add more system memory to the system, however.
- **Simplicity:** You don't have to install, configure or manage a hardware RAID controller.
- **Duplexing:** Duplexed RAID 1 can sometimes be implemented in software RAID but *not* in hardware RAID, depending on the controller.

That's pretty much it. Now the cons:

- **Performance:** The best-known drawback of software RAID is that it provides lower overall system performance than hardware RAID. The reason is obvious: cycles are "stolen" from the CPU to manage the RAID array. In reality, this slowdown isn't *that* excessive for simple RAID levels like RAID 1, but it can be substantial, particularly with any RAID levels that involve striping with parity (like RAID 5).
- **Boot Volume Limitations:** Since the operating system has to be running to enable the array, this means the operating system cannot boot from the RAID array! This requires a separate, non-RAID partition to be created for the operating system, segmenting capacity, lowering performance further and slowing boot time.
- **Level Support:** Software RAID is usually limited to RAID levels 0, 1 and 5. More "interesting" RAID levels require hardware RAID (with the exception of duplexing, mentioned above.)
- **Advanced Feature Support:** Software RAID normally doesn't include support for advanced features like hot spares and drive swapping, which improve availability.
- **Operating System Compatibility Issues:** If you set up RAID using a particular operating system, only that operating system can generally access that array. If you use another operating system it will not be able to use the array. This creates problems with multiple-OS environments that hardware RAID avoids.
- **Software Compatibility Issues:** Some software utilities may have conflicts with software RAID arrays; for example, some partitioning and formatting utilities. Again, hardware RAID is more "transparent" and may avoid these problems.
- **Reliability Concerns:** Some RAID users avoid software RAID over concern with potential bugs that might compromise the integrity and reliability of the array. While hardware RAID controllers can certainly *also* have bugs, I think it's reasonable to believe that some operating systems are more likely to have these sorts of problems than a good-quality hardware RAID controller would.

All things considered, software RAID doesn't seem to have much to recommend it. At the same time, realize that in many cases it is much better than using nothing at all. If you are running a small business, a software RAID 1 solution is *far* superior to running without RAID at all, especially if you aren't meticulous about your backups. (Then again, if you can afford two drives to do mirroring, a hardware RAID card is often only a small incremental cost...)

Next: Controller BIOS and BIOS Upgrades

Controller BIOS and BIOS Upgrades

Much the way that a PC motherboard has its system BIOS--low-level software that runs the chipset and other key components of the motherboard--hardware RAID controllers do too. The controller BIOS is the code that operates the controller, managing the RAID array.

Over time, the manufacturer of the RAID controller may change the BIOS code; this is usually done to correct minor problems with the controller card, but sometimes occurs to enable new features. It is also occasionally done to enable support for hardware that did not exist at the time the controller shipped. Like PC BIOSes, RAID controller card BIOSes can usually be upgraded through a flash process similar to that used for motherboards. A new BIOS is typically downloaded from the web site of the manufacturer and written to the EEPROM (flash chip) on the RAID controller. A special software program, called a *flash program* for unsurprising reasons, is used to ensure that the BIOS upgrade works properly.

Warning: To avoid potential problems, always follow the manufacturers' instructions carefully when attempting to flash *any* BIOS. Doing the flash procedure incorrectly can render the controller card inoperative. Always use the right BIOS image for your card, and make sure the data on the array has been backed up before you begin.

It's not necessarily a good idea to update the BIOS every time the manufacturer puts out new code. Check the release notes for the new version to find out what has changed; if no important changes have been made that affect how you will use the controller, you may not need to bother.

For software RAID there is no "controller BIOS", since the operating system is running the array. The analog for the controller BIOS upgrade here is the *operating system patch*, which most PC users know all too well. :^) It's rare for problems with RAID in an operating system to occur that necessitate such a software patch, but it's still worth keeping current on whatever operating system you are using, since patches usually correct scores (or even hundreds) of bugs and problems.

Next: RAID Interfaces

RAID Interfaces

There are two primary *interfaces* that are used for RAID arrays. Traditionally, all RAID was done on high-end machines and used the SCSI interface. Lately, hardware RAID cards for the ubiquitous IDE/ATA interface have begun to explode in popularity. Therefore, when designing a system that you intend to use RAID, you have a choice of interface to make. Of course, this is a choice that affects the design of the system in a fundamental way. The matter of SCSI vs. IDE/ATA for RAID is similar to the general ongoing debate over the two interfaces, which I have contrasted in this section.

The decision of SCSI vs. IDE is much like that of hardware RAID vs. software RAID--it comes down to cost vs. features. IDE/ATA RAID is much less expensive than SCSI RAID but much more limited in virtually every way: array capacities are smaller since a smaller number of drives can be used; performance is lower in many different ways; support for more complex RAID levels is absent; and advanced features are much less common. And those are just the more important ones. :^)

In practice, most serious RAID implementations still use SCSI; I don't see IDE/ATA replacing SCSI RAID for heavy-duty use on servers or high-end workstations, because it is just too limiting. What IDE/ATA RAID *is* doing however, is opening up the world of consumer-grade hard disks to RAID, enabling millions who cannot afford the cost of SCSI to enjoy some of the important benefits of RAID economically. If you already have an IDE/ATA system and want to install RAID, you no longer have to face the sometimes daunting task of moving over to SCSI to do it.

Next: Multiple Channels and Throughput Issues

Multiple Channels and Throughput Issues

I consider the external data transfer rate of the interface to be one of the most overrated hard disk performance specifications ever. The reason is that there are no hard disks that can read or write data fast enough to max out modern interfaces. They can only send at the interface's maximum speed for short bursts; see this section for further explanation. It doesn't really matter much if the drive is running on an Ultra ATA/100 interface capable of 100 MB/s theoretical throughput if the drive can only stream from the platters at 40 MB/s. The only data sent anywhere near 100 MB/s will be the small amounts that happen to be in the drive's cache. The same applies to a single drive on a high-speed SCSI bus.

However, when using large numbers of drives with RAID, suddenly things become a bit different. Well, under IDE/ATA they aren't different, because IDE/ATA can only handle transfers from one drive at a time. :^) With SCSI though, it's a very different story. On the SCSI bus, multiple drives can be transferring data simultaneously. This means that if you put a bunch of drives on a single SCSI channel and want to run them all at their maximum potential, you do have to be careful to watch the maximum throughput of the bus.

As usual, an example is much easier than trying to explain it without one. Let's take the Quantum Atlas 10K II. This drive has a maximum sustained transfer rate of 40 MB/s. For ideal performance, we want to make sure that the interface can supply at least that much bandwidth. Now if we put a single one of these drives on an Ultra160 SCSI bus, we obviously have no problems; the theoretical maximum speed of the bus is 160 MB/s (though actual will be below that due to overhead considerations). But what if we want to make a four-drive RAID 0 array for high-speed multimedia editing? In that case we *do* care about the speed of the bus: we're going to use all of it when dealing with large files, because all four drives will be streaming data simultaneously! In fact, we'll be slightly below theoretical maximum potential speed because of overhead, but it's probably close enough, especially since that STR figure is only that high for the outermost tracks of the drives, where the number of sectors per track is at its maximum.

But what about if we decide we want to create a larger array, say, an eight-drive array? Then we have a problem. Even if we use the *average* STR figure of those drives, 32 MB/s, we need 256 MB/s, far in excess of what Ultra160 can provide. To avoid this problem, higher-end SCSI RAID controllers provide support for *multiple channels*. Essentially, the RAID controller has not one SCSI bus with which to communicate with the drives in the array, but two or more. For example, some cards have four channels. Each of these is capable of handling 160 MB/s in theory, yielding a whopping theoretical bandwidth of 640 MB/s. That's obviously more than enough to handle our eight-drive array; we just put two drives on each of the four channels and we are in business; we even have room to substantially expand the array in the future, or add a second array if need be. The use of multiple channels also improves performance by cutting down on contention for the SCSI bus. Of course you don't get four channels on a RAID controller for free; these multi-channel controller cards aren't cheap.

Another issue when dealing with very high transfer rates is the bandwidth of the bus itself. The standard PCI bus as implemented in regular PCs--which seemed to have so much bandwidth five years ago :^)--is 32 bits wide and runs at 33 MHz, providing a total maximum *theoretical* bandwidth of about 127 MB/s, not nearly enough to handle multiple-channel SCSI RAID. For this reason, high-end cards with multiple channels often use the enhanced 64-bit, 66 MHz PCI bus. This version of PCI has a theoretical bandwidth of over 500 MB/s, but it of course requires a server motherboard that has a matching high-speed PCI slot. Again, not cheap.

Next: RAID Hard Disk Drive Requirements

RAID Hard Disk Drive Requirements

The "I" in "RAID" stands for *Inexpensive* (even though it sometimes rendered as *Independent*). This seems somewhat puzzling, since it is only recently that RAID has begun to become popular on inexpensive IDE/ATA hard disks--high-end SCSI disks are still the primary domain of RAID, and more often than not, RAID arrays are constructed of some of the fastest drives on the planet. The reason for the term "inexpensive" is that RAID is designed to use *regular* hard disks, as opposed to proprietary, special hard disks. This is in fact a major reason why it is so appealing. Before RAID, to get high performance or fault tolerance required the purchase of specialized, very expensive drives. RAID lets you get those benefits while using off-the-shelf drives.

Perhaps ironically, hard disks today are becoming downright inexpensive period, not just when compared to proprietary solutions (which RAID has caused to all but disappear from the market). This has fueled the RAID phenomenon and caused more and more RAID systems to be created using disks of all pricing levels.

While RAID arrays use standard hard disks, that doesn't mean that you can just pick "any old drives" you find kicking around and make a successful RAID array out of them. There are a number of requirements that must be considered when choosing drives (beyond the obvious, such as choosing drives matching the interface of your controller implementation!) In this section, I will discuss factors such as drive size, number of drives and how to choose specific units, along with taking a look at how to meet the "needs" of your drives in terms of power, enclosures and cabling.

Next: Number of Drives

Number of Drives

The number of hard drives in the system has an important impact on both capacity and performance. Many RAID enthusiasts believe that the more drives you put in the array, the better off you are--and this usually is mostly true. At the same time, there are also disadvantages to using more drives as opposed to fewer.

The number of drives in the array affects the following aspects of the array directly:

- **Capacity and Storage Efficiency:** The size of the array you end up with depends on the RAID level implemented and the number of drives in the array (as well as the size of each drive, of course). Storage efficiency, meaning the percentage of the total amount of space on all the drives that contains user data, is also a function of the number of drives in the array for all RAID levels that use striping with parity (the number of parity drives is fixed so the efficiency goes up as you add more drives.) See this discussion of capacity and storage efficiency for more.
- **Performance:** For RAID arrays that use striping, the stripe width (not stripe size) is equal to the number of disks in the array. Adding more drives improves both random and sequential performance; in the case of sequential performance, theoretical throughput is roughly proportional to the number of drives in the array (with random performance it is more complicated).
- **Cost:** More drives cost more money, even if you don't increase the capacity of the array. Ten 20 GB drives is 200 GB, but it will cost more than five 40 GB drives do. Then, above the cost of the drives, you have to consider support costs: more drives take more space, need more power, and often require more cooling. Going to too many drives may require the purchase of an extra enclosure, and this incremental cost alone can dwarf the other hardware costs for a RAID array in a smaller workstation or server. Also, see here for issues related to power and cabling.
- **Reliability:** More hardware means more to potentially fail; the reliability of the overall system will go down when more drives are involved. See here for more.

The #1 question that people ask about the number of drives to use in an array goes something like this: "Should I use three 36 GB drives, or six 18 GB drives?" The answer, of course, is "it depends". For people who really want to maximize performance--which is most people--the answer is six 18 GB drives. However, if you do this, you must be prepared for the extra costs. As mentioned above, sometimes the deciding factor is how many drives the system can hold; it might be ideal to make a striped array out of twelve 9 GB drives instead of either of the options above, but few systems can handle twelve drives without adding an expensive external enclosure, which totally changes the cost picture of the entire system. The capabilities of the controller can also enter the picture, as they will often have a limit on the number of drives they will support.

Next: Drive Size

Drive Size

When I talk about *drive size* in a RAID array, I really mean two different things: first, the capacity of the drives in gigabytes, and second, the physical size of the drives. The first is important to the overall capacity of the array; the second has important implementation implications (gotta love that alliteration. :^)) And the two are related as well, because larger drives are often physically larger than small ones.

The capacity of the drives used in a RAID array affects the capacity of the overall array, of course. Larger drives yield larger overall capacity for the array, all else being equal. For this reason, RAID arrays tend to be constructed of large drives whenever possible. This is especially true because the cost of hard drives per gigabyte decreases when you buy the larger sizes of any given family. Another important issue is that drives should all be the same capacity when used in a RAID array, or you will forfeit any additional capacity on the larger drives in the array. For more on these and other capacity issues, see here.

If you have a limited budget, you will be forced to some extent to trade off individual drive capacity against the number of drives you purchase; this is the reason for the "should I buy more small drives or fewer large" drives conundrum examined here. More drives means better performance, but fewer large drives can be substantially less expensive. If you need large capacity but are limited by physical considerations to a small number of drives, obviously, get large drives. Another reason to consider large drives besides cost and space is *expansion*. If your system can handle a total of six drives and you fill all six spots with small drives, you'll need to replace drives--possibly all of them--if you want to increase capacity later on (which sometimes is the best idea anyway, but can be expensive and time-consuming; see here.)

Physical size is another issue, more specifically the form factor of the drives. The standard hard disks used today in RAID configurations are almost always 3.5" form factor drives, which describes a standard width and depth for these units. There are two different heights found in 3.5" form factor drives, however. The standard height, called *slimline* or *low-profile*, is used for all standard IDE/ATA and most smaller SCSI hard disks; these drives are 1" high. For high-end SCSI drives however, the largest member of most families use anywhere from 6 to 12 platters, too large to fit in a 1" package; these drives are 1.6" in height and are called *half-height* drives (the "half" being relative to the full size of the ancient 5.25" floppy drive bay in the original PC--don't ask. :^)) The hard disk shown on the main page of the hard disk reference section is a 1.6" "half-height" drive.

Most regular PCs are not designed to take these larger form factor drives, because their drive bays are sized for low profile drives. Some may still take 1.6" drives, especially if there are two contiguous 3.5" bays without obstructions between them. Still, if you really want to use half-height drives you want to use a server case that is designed specifically for these larger devices. This will also help avoid cooling and power problems when using several larger drives.

Next: Drive Selection Criteria

Drive Selection Criteria

In other sections of the site's RAID coverage, I have hinted at some of the considerations that go into selecting drives for a RAID array. Some of these are fairly obvious, but others are not. Here is a fairly comprehensive list of issues to keep in mind when considering drives for a RAID array:

- **Interface:** You must of course get drives that correspond to the interface your controller uses (really, this is a "group decision" of course.)
- **Capacity:** Get the largest drives that you can afford and that will physically fit your system or enclosure.
- **Physical Size:** Don't get half-height drives unless you have a case or enclosure that can handle them.
- **Performance:** Get the fastest drives you can afford. Hard disk performance is a huge subject unto itself, which I won't try to summarize in a paragraph here, other than to say that if performance is important to you, you must educate yourself about performance issues and choose drives that meet your needs. The only synopsis I'll provide is to say that if your main concern is random access performance, look for a drive model with low overall access time; if transfer rate is of utmost importance, instead look for drives with the highest sustained transfer rates.
- **Quality:** Get high-end, high-quality drives. See this discussion of quality and reliability issues.
- **Source:** If you are buying a "pre-made" array, get it from a high-quality manufacturer or system integrator. If you are setting up the array yourself, *always* buy drives from a reputable source, preferably an authorized dealer. This will ensure that you get a proper warranty, that you have support if you need it, and that you don't end up buying second-rate merchandise.

- **Uniformity:** To keep performance balanced and to maximize storage efficiency, drives should all be the same size and model. *However*, also keep in mind the point below...
- **Diversification:** This is a very important consideration when selecting drives for a RAID array. Drive failures are sometimes caused by random flaws, making an individual unit fail while others made at the same time are just fine. But sometimes, drive failures are related to occasional manufacturing difficulties; this can cause a specific batch of drives to have a much higher rate of failure than normal for that drive family. If you set up six drives in a RAID 5 array and they are all from a faulty lot, you run the very *real* risk of having two simultaneous failures and losing the entire array. For this reason, you should get drives that are all the same make, model and size, but you should if at all possible get drives from different manufacturer lots. A good vendor should be able to help you do this.

Next: Enclosures

Enclosures

Most PC system cases are designed, well, under the assumption that they will be used for regular PCs. Typically, space is provided within the case for one, two, or maybe three hard drives. Sometimes there will be enough space for four or more, and you can also "make space" by using drive bay adapters. A regular PC case can therefore be satisfactory for small RAID arrays of 2, 3 or 4 drives, and certainly this is the least expensive option. This is often how low-end, IDE/ATA RAID is done.

For "serious RAID" using many drives and the SCSI interface however, this sort of arrangement is not really acceptable. More drives are often needed, and in particular, drive swapping has become an important feature that most high-end RAID users insist upon. To enable hot swapping and large numbers of drives, you must look beyond regular PC cases.

One way to enable RAID and features such as hot swapping is to use a case specifically designed for servers; you can see a picture of one below, and a different one here. Note the drive bays on both cases; the large number of spaces for drives is specifically intended for RAID applications. You can see the removable drive handles on the drives. These cases are usually very high quality, and come with very beefy power supplies--and substantial price tags.

A very large, very nice server case, which would make easier the life of someone using it to implement a RAID array. (Enlight's 8850.)

Image © Enlight Corporation
Image used with permission.

If you already have an existing system or for another reason don't want to go with a specialty server case, another way to go is a separate RAID *enclosure*. This is a fancy word for what is in essence an auxiliary case. It functions exactly the same way that a regular case would except that it is external to the main system box and is connected to it using one or more interface cables. Enclosures are also very expensive. (Incidentally, if the enclosure also includes a RAID controller, then it is no longer just an enclosure, it's an external, stand-alone hardware RAID array; see here for more.)

Next: Cabling and Power Requirements

Cabling and Power Requirements

Much the way the many drives used in larger RAID arrays can tax the available space in a regular PC case, they do the same to the capabilities of the system power supply. If you examine the output ratings of a typical power supply, you'll find that the amount of +12 V power provided is more than adequate for as many as four hard disks, but some lower-end supplies may not be able to handle more than two. Virtually no regular PC power supply will be up to the challenge of providing adequate power to a RAID array of 8, 10, 12 or more hard drives. This is particularly true when you consider the peak startup draw of hard drive motors.

For this reason, it's important to carefully check the capabilities of the power supply when implementing a larger RAID array. The need for power is another reason why larger RAID arrays are usually implemented in either specialized server cases or external enclosures. These cases are matched with larger power supplies that can handle the load of many hard drives. Often these cases will in fact be equipped with redundant power supplies, which provides fault protection against problems with the power supply. If one supply fails the other will continue to seamlessly provide power to the entire array. I recommend these for those who are setting up a RAID array for fault tolerance and can afford them.

Tip: Power supplies provide power using the integrated power cables and connectors that come from the power supply box. You will often notice that these connectors are in separate "groups"; for example, there may be 8 power connectors arranged into two groups of four connectors. If at all possible, "spread these around" so that if a fault develops with one set of cables it will not take the entire array down. In the example above, if you are running a RAID 1+0 array, "split" each of the mirrored pairs so one drive is attached to one group and the other drive is attached to the second group. Even if the power is cut to entire group then, the array will stay up since one drive in each RAID 1 sub-array will still be powered.

Another issue with larger arrays is cabling: it can become quite a mess if you have a large number of drives. Snaking four large SCSI cables to a dozen hard drives and running the power cables to them isn't a lot of fun. Separate cables also make drive swapping difficult to impossible, so a drive failure means taking down the system. Larger cases will help to keep the cabling from becoming unmanageable, but a better solution now being used by many server cases is a SCSI variation called single connector attachment or SCA.

In an SCA system the separate data and power cables that normally run to each drive are eliminated and replaced with a single 80-pin connector. Special *backplanes* are installed in the server case and the drives snap into the mating connectors on the backplanes, in a process not dissimilar to how you connect a printer cable to a parallel port. The connection between the backplanes and the RAID controller is greatly simplified compared to running data and power cables to each drive, and SCA is designed specifically to allow drive swapping. Most RAID enclosures and server cases are now designed to use SCA due to its significant advantages for RAID arrays.

A final not relates to controllers that use multiple channels. If your controller has this useful feature then you should always use all of the channels available to improve performance. The optimal way of connecting the drives depends on the number of channels you have and the type of RAID array you have implemented. In general, you want to evenly distribute the drives between the channels to allow as little contention as possible within each channel, and to improve throughput. So if you have a six-drive array on a three-channel controller, put two drives on each channel. If you have eight drives, split them up as into groups of 2, 3, and 3.

Next: RAID Management

RAID Management

RAID systems, especially high-end ones, can be complex--more features means more complexity. A simple IDE/ATA RAID array typically needs little in the way of management, but it becomes important for larger arrays with multiple RAID levels, many virtual drives, and so on. All RAID systems, whether hardware or software, come with some sort of administration program for setting up, configuring, and managing the array(s) they control.

In this section I will address some of the basic issues related to managing a RAID array. This includes a description of RAID management software, including a look at the types of errors and warnings that such software may generate, a discussion of partitioning and partitioning software and how RAID affects them, and an explanation of remote RAID management. I conclude with a brief discussion of service and support issues, which can be very important in a RAID environment where high availability is a priority.

Next: Management Software

Management Software

All RAID controllers come with some sort of software that lets you manage the controller and its connected arrays and drives. This software can range from very simple to very complex, depending on the type of controller and the number of features it supports. Most hardware controllers come with two different software components:

- **Controller BIOS Setup Program:** This is a hardware-based program that can be run at boot time in exactly the same way that a motherboard's BIOS setup program operates. By pressing the appropriate key(s) at the appropriate time during the boot sequence, a special program contained in a ROM chip on the controller appears on the screen (usually in a text-only mode) and allows various functions to be performed. This program manages a set of data contained on the controller that is used by the controller's onboard BIOS. This type of program is essential for hardware RAID since you need to be able to set up and configure arrays before any operating system is loaded.
- **Operating System Management Utility:** This is a straight software utility that runs after the operating system is loaded and running on the array. It allows many of the same features as the setup program, except it is designed more towards checking the status of the array and changing various parameters that control how it operates. It typically also will allow operations that continue while an array is operating normally, such as a rebuild on a fault-tolerant array. It works in conjunction with the setup program.

I couldn't even begin to describe in detail all the features and options found in the software utilities that come with high-end RAID controllers--they often come with manuals that approach 200 pages in length. Of course, the software also varies significantly from one manufacturer to another, and one product to another. Regardless of how it is set up, the functionality of RAID management software allows you to perform the following basic categories of functions:

- **Controller Configuration:** Configuring the controller and its features at a hardware level. For example, setting how the internal cache, if any will work; controlling alarms; selecting manual vs. automatic rebuild for failed drives, disabling the BIOS if necessary, and so on.
- **Array Configuration:** Defining and configuring RAID arrays, setting up which drives are in which arrays, and more.
- **Physical Drive Management:** Checking the status of drives connected to the controller, formatting drives, designating drives as hot spares, and so on.
- **Logical Drive Management:** Creating logical drive volumes from an array, formatting a logical volume, etc.
- **SCSI Channel Management:** For SCSI cards, controls various settings and parameters related to setting up and managing the SCSI channels on the RAID controller. Many of these are related to managing SCSI bus termination.

Tip: As I mentioned above, RAID controllers normally come with extensive documentation. You can usually find the manuals for most RAID products in downloadable form on the web site of their manufacturer. Reading through the software manual for a controller you are considering buying can give you a *much* clearer picture of the controller's capabilities and limitations than you will get from glossy marketing blurbs.

How about software RAID? Well, there's no surprises here: a management program is required to manage all RAID features. There is obviously no BIOS setup program because there is no controller BIOS, so everything is done at the operating system level. RAID management functionality for Windows NT and 2000 is integrated into the Disk Administrator tool that is used for managing disk volumes.

Next: Remote Management

Remote Management

Normally, the management of each RAID controller is performed by software running on the machine where the RAID card is installed. This works fine for smaller organizations that might only run RAID on a central server. For larger companies, which might well have many machines with RAID controllers, this can become cumbersome. It gets even worse if there are servers in different geographical locations.

To address the need for central administration of multiple RAID arrays on different machines, some high-end RAID solutions include support for *remote management* This enables an administrator to monitor the operation and status of a RAID array of a server or workstation without being physically present. This is normally done over a local area network, with support from the RAID controller card. In fact, some RAID controllers now actually support certain management features over the Internet!

Remote management is a feature that most RAID users won't need to worry about, but if you will be in a different location from the array it can be very useful. You can usually find out if this feature is supported by a controller card by checking its feature listings, or even better, downloading its user manual from its manufacturer's web site.

Next: Partitioning and Partitioning Software

Partitioning and Partitioning Software

As explained in detail in this part of the discussion of file systems, partitioning is the process of dividing up a hard disk into pieces so they can be treated as logical disk volumes. Under RAID arrays are treated as "virtual hard disks" and are partitioned just as any regular hard disk would be.

In theory, hardware RAID arrays should be completely transparent to any software, including operating systems, that are installed on them. An operating system driver may be required for the controller card, but that's about it. Since it should be transparent, you should be able to use any partitioning software you want, ranging from vanilla FDISK to more sophisticated programs like Partition Magic. Despite this, some controller manufacturers specifically say not to use third-party partitioning software with their products. And despite *that*, many people use them anyway without any problems. It is unclear if there is really a problem with third-party tools on these (typically low-end) controllers, or if the manufacturers decide that the easiest way to reduce support costs and hassles is just to say "don't use anything but FDISK". At any rate, this should not occur with more expensive controllers.

One real concern with RAID and partitioning software has to do with size. As hard disk capacities increase, partitioning software is slowly revised or updated to handle ever-increasing drive sizes. Unfortunately, some of these programs are unprepared for the much larger "virtual disks" that a hardware RAID solution may present to them as if they were single disks. Using the latest operating system updates should let you avoid these problems. When running large arrays, the NTFS or UNIX file systems are preferable to the different flavors of FAT.

Warning: Do not use overlay software in conjunction with hardware RAID controllers. It's just asking for trouble, and it's really not necessary. The controller should be able to handle any size drives you can throw at it; if not, get a BIOS upgrade for it. If you are running software RAID you may be OK with an overlay, but I still recommend hardware support, not drive overlay software.

Next: Alarms and Warnings

Alarms and Warnings

While the software that comes with RAID controllers will let you check the status of the array at any time, there are situations where the administrator of the array needs to know that something has happened, *now*. Finding out about important bad news "the next time you run the management utility" just isn't good enough, and anyone who manages RAID arrays is typically too busy to keep checking for problems all day long--especially since they occur rarely anyway.

For this reason, controllers usually are programmed to generate *alarms* and *warning* messages when certain problems occur with the controller or the array. On better controllers these take the form of an audible alarm: loud beeping coming from the controller card that will certainly make you sit up and take notice, believe me. :^) Audible notification greatly increases the chances that trouble will be addressed immediately. There are cases where this feature can be the difference between a hardware problem being an inconvenience, and being a disaster.

The conditions that will trigger a warning vary from one controller to another, but the most common ones include these:

- **Array Failure:** An array connected to the controller has failed due to a hardware fault. This would occur due to failure of enough drives to compromise the array, so one failure will do it for RAID 0, two for RAID 1, 3, 4 or 5, and so on. For a multiple RAID level, the failure of a component "sub-array" will normally trigger an alert even if the "super-array" continues working. So in RAID 0+1, one RAID 0 array may fail while its mirror carries on, but the failure of the RAID 0 sub-array will generate an alert. This error condition is sometimes described as the array being "offline" (which of course it would be.)
- **Degraded Mode Operation:** An array connected to the controller is running in a degraded state due to a hardware fault. This warning situation will occur in a redundant RAID level where a number of drives have failed, but not enough to take the array offline. The array will continue to run but performance will be degraded until the fault is corrected and the failed drive is rebuilt. This alert is arguably the most

important one of all, because it's hard not to notice an outright array failure, but it *can* be hard to know that an array is still up, but running in a degraded state.

- **Rebuild Completion:** If an automatic rebuild of a degraded array is in progress, the controller may signal when the rebuild is complete. This signals that the array is no longer in degraded mode, which can be important to know if, for example, a drive failed and a hot spare was rebuilt in its stead in your absence (you'll know that the failure occured and that you now need to replace the failed drive.)
- **Controller Hardware Fault:** The controller has detected some sort of internal fault or problem. For example, some controllers monitor their own temperature and may issue a warning if acceptable limits are exceeded.

In addition to audible alerts, notification of important conditions can usually be sent over a local area network to an administrator. Controllers that support remote management will of course allow remote notification as well. In addition, modern controllers also usually support the SMART feature and will report SMART warnings generated by hard disks that include SMART.

Warning: Some RAID controllers will let you disable the audible warning feature if you find it too "annoying". Doing this is like pulling all the batteries out of your smoke detectors so they won't "irritate you" while you're trying to sleep...

Next: Service, Support and Maintenance

Service, Support and Maintenance

No discussion of managing a hardware system would be complete without mentioning maintenance. At least, it shouldn't be! :^) I think enough about maintenance that I have a special section where I talk about it in some detail. Rather than repeat all of that information here, I will once again exploit the power of the web and simply refer you to there. Here I will only discuss maintenance as it applies specifically to RAID arrays.

That said, there isn't a lot to say. :^) RAID arrays don't generally require a lot in the way of regular preventive maintenance. You do need to maintain your server hardware, and the array should be part of that, but that's really all you need to do under normal circumstances: No special maintenance is required for RAID controllers or drives. At the same time, many high-end controllers do offer advanced maintenance features which can be useful, such as the following:

- **Consistency Checking:** This important feature will proactively check the data on a RAID array to ensure that it is consistent, meaning that the array data is correct and has not become corrupted. It is especially useful for RAID levels that use striping with parity, as it will check for any situations where the parity information in a stripe has become "out of sync" with the data it is supposed to match (which shouldn't happen in practice, but you know how Mr. Murphy works...) It will of course also correct any problems it discovers.

- **Spare Drive Verification:** If you are using hot spares, they will tend to sit there for weeks or months on end unused. This feature, if present, will check them to ensure they are in good working order.
- **Internal Diagnostics:** Some better RAID controllers may include routines to periodically check their own internal functions and ensure that they are working properly.

Now, let's take a look at service and support issues with RAID (I discuss these in more general terms in this troubleshooting section). In fact, there isn't anything different about service and support of RAID *hardware* than any other hardware. The difference is that RAID arrays are usually employed in servers used by many people, or in other critical situations which require a minimum of down-time. This means a failure that takes down an array can quickly cost a lot of money, lending an *urgency* to RAID array service that may not be present for other PCs.

There is no way to avoid down-time entirely unless you spend a truly staggering amount of money (and even then, you should "expect the unexpected".) If a failure occurs, you want to get it corrected as soon as possible, and that means you are reliant to some extent on whatever company is supporting your hardware. If uptime is critical to your application, then in addition to incorporating fault tolerance into your RAID setup, you should purchase an on-site service contract covering your system(s). Be sure to look at all the conditions of a service contract to be sure you understand what it covers--and what it doesn't. If you need immediate response in the event of a hardware fault be sure that the contract specifies that--you'll certainly pay more for it, but you have to weigh that against the cost of an entire company "waiting for the system to come back up".

Note: Another important issue to keep in mind when considering service and support of your RAID array, is that some arrays "insist" upon having failed drives replaced with identical models. You want the company that supplies you with hardware to be able to provide you with new drives of the appropriate type for the life of the array.

Next: Advanced RAID Features

Advanced RAID Features

There are several important features that have been developed over the last few years to improve the capabilities of RAID-equipped systems. They are usually found on high-end systems that use more expensive RAID controllers. In addition to a capable controller, these features may also require other hardware in the system to be designed to support them. For example, the system case or RAID enclosure must be designed to support drive swapping for that feature to work.

In this section I take a look at caching (including write caching), hot drive spares and drive swapping, three key features that improve performance and availability in RAID arrays. Finally, I discuss the important issue of RAID array expansion, and how to plan for the future of your RAID array.

Next: Caching

Caching

Caching is a technique that is used to buffer discrepancies between the performance of different hardware components. It is used throughout the PC: cache is found within system processors, on motherboards, within hard disks themselves, and many other places. In every case, the goal of the cache is the same: to provide a temporary storage area that allows a faster device to run without having to wait for a slower one. For more on the theory of caching and how it works in PCs, see this explanation.

Most advanced RAID controllers include on-board cache, which in many ways acts exactly the same way that the cache within a hard disk does: it improves performance to some extent by storing information that was recently used, or that the controller predicts will be used in the future, so it can be supplied to the system at high speed if requested instead of necessitating reads from the slow hard disk platters. Since a RAID controller turns an array of hard disks into one "virtual hard disk", putting cache on the controller is a natural enhancement. Typically this is implemented as a slot on the controller that takes a standard PC memory module; some controllers can take an amount of cache exceeding the total system memory on most regular PCs! While caching does improve performance, as with cache size in hard disks, don't overestimate the performance impact of increasing the size of the cache.

One area where caching can impact performance significantly is *write caching*, sometimes also called *write-back caching*. When enabled, on a write, the controller tells the system that the write is complete as soon as the write enters the controller's cache; the controller then "writes back" the data to the drives at a later time. As described in detail in this page on write caching, this improves performance but imposes the risk of data loss or inconsistency if, for example, the power to the system is cut off before the data in the cache can be "written back" to the disk platters.

Tip: To avoid potential problems with power failures and write-back caching, some controllers actually incorporate a built-in backup battery! This battery will allow any unwritten data in the cache to be retained for a period of time until the power is restored to the system. A very neat feature--though if the server is connected to a UPS, as it should be anyway, its value is debatable.

The reason that write-back caching is so important with RAID is that while writes are slightly slower than reads for a regular hard disk, for many RAID levels they are much slower. The cache insulates the system from the slowdowns inherent in doing writes to arrays such as those using RAID 5, which can improve performance substantially. The bigger the gap between read and write performance for a given RAID level, the more that array will benefit from using write caching. It is recommended for high-performance applications using striping with parity (though it will improve performance somewhat for all RAID levels.)

Next: Drive Swapping

Drive Swapping

In the "good old days" of RAID, fault tolerance was provided through redundancy, but there was a problem when it came to availability: what do you do if a drive fails in a system that runs 24 hours a day, 7 days a week? Or even in a system that runs 12 hours a day but has a drive go bad first thing in the morning? The redundancy would let the array continue to function, but in a degraded state. The hard disks were installed deep inside the server case, and this required the case to be opened to access the failed drive and replace it. Furthermore, the other drives in the array that continued to run despite the failure, would have to be powered off, interrupting all users of the system anyway. Surely there had to be a better way, and of course, there is.

An important feature that allows availability to remain high when hardware fails and must be replaced is *drive swapping*. Now strictly speaking, the term "drive swapping" simply refers to changing one drive for another, and of course that can be done on any system (unless nobody can find a screwdriver! :^)) What is usually meant by this term though is *hot swapping*, which means changing a hard disk in a system without having to turn off the power and open up the system case. In a system that supports hot swap, you can easily remove a failed drive, replace it with a new one and have the system rebuild the replaced drive immediately. The users of the system don't even know that the change has occurred.

Unfortunately, "hot swap" is another one of those terms that is used in a non-standard way by many, frequently leading to confusion. In fact, there are a hierarchy of different swap "temperatures" that properly describe the state of the system at the time a drive is swapped:

- **Hot Swap:** A true hot swap is defined as one where the drive can be replaced while the rest of the system remains completely uninterrupted. This means the system carries on functioning, the bus keeps transferring data, and the hardware change is completely transparent.

- **Warn Swap:** In a so-called "warm swap", the power remains on to the hardware and the operating system continues to function, but all activity must be stopped on the bus to which the device is connected. This is worse than a hot swap, obviously, but clearly better than a cold one.
- **Cold Swap:** The system must be powered off before making the swap.

It is common for a system to be described as capable of hot swapping when it really is only doing warm swaps. True hot swapping requires support from all of the components in the system: the RAID controller, the bus (usually SCSI), the enclosure (which must have open bays for the drives so they can be accessed from the front of the case), and the interface. It requires special connectors on the drives that are designed to ensure that the ground connections between the drive and the bus are maintained at any time that the device has power. This means that when removing a device, the power connection has to be broken before the ground connection, and when re-inserting a device, the ground connection has to be made before the power connection is re-established. This is typically done by designing the connectors so that the ground connector pins are a bit longer than the other pins. This design is in fact used by SCSI SCA, the most common interface used by hot-swappable RAID arrays. See this discussion of SCA for more, as well as this discussion of drive enclosures.

As mentioned above, the SCA method on SCSI is most commonly used for hot-swappable arrays. In the IDE/ATA world, the best you can usually do is warm swapping using drive trays, which "convert" regular IDE/ATA drives to a form similar in concept to how SCA works, though not quite the same. This is still pretty good, but not really hot swapping. The system usually needs to be halted before you remove the drives.

A system that cannot do hot swapping, or even warm swapping, will benefit from the use of hot spares. If your system can only cold swap, you will at some point have to take it down to change failed hardware. But if you have hot spares, you can restore the array to full functionality immediately, and thus delay shutting the system down to a more convenient time, like 3:00 am (heh, I meant more convenient for the *users*, not you, the lucky administrator. :^)) In fact, hot sparing is a useful feature even if you have hot swap capability; read more about it here.

Next: Hot Spares

Hot Spares

If a drive fails in a RAID array that includes redundancy--meaning all of them except RAID 0--it is desirable to get the drive replaced immediately so the array can be returned to normal operation. There are two reasons for this: fault tolerance and performance. If the drive is running in a degraded mode due to a drive failure, until the drive is replaced, most RAID levels will be running with no fault protection at all: a RAID 1 array is reduced to a single drive, and a RAID 3 or RAID 5 array becomes equivalent to a RAID 0 array in terms of fault tolerance. At the same time, the performance of the array will be reduced, sometimes substantially.

An extremely useful RAID feature that helps alleviate this problem is hot swapping, which when properly implemented will let you replace the failed drive immediately without taking down the system. Another approach is through the use of *hot spares*. Additional drives are attached to the controller and left in a "standby" mode. If a failure occurs, the controller can use the spare drive as a replacement for the bad drive. A very simple concept, and a feature that is supported by most RAID implementations, even many of the inexpensive hardware RAID cards and software RAID solutions. Typically, the only cost is "yet another" hard disk that you have to buy but can't use for storing data. :^)

You may ask though: if I have hot swap capability, why do I need hot spares anyway? I can just replace a drive when it fails, right? That's true, but the main advantage that hot sparing has over hot swapping is that with a controller that supports hot sparing, the rebuild will be *automatic*. The controller detects that a drive has gone belly up, it disables it, and immediately rebuilds the data onto the hot spare. This is a tremendous advantage for anyone managing many arrays, or for systems that run unattended--do you really want to have to go into the office at 4 am on a rainy Sunday to hot-swap a drive for the benefit of your overseas users?

As features, hot sparing and hot swapping are independent: you can have one, or the other, or both. They will work together, and often are used in that way. However, sparing is particularly important if you *don't* have hot swap (or warm swap) capability. The reason is that it will let you get the array back into normal operating mode quickly, delaying the time that you will have to shut down the system until when you want to do it. You of course lose the hot sparing capability in the meantime; when the failed drive is replaced, the new drive becomes the new hot spare.

Tip: Hot spares may sit dormant on a system for months at a time. It's a good idea to periodically test the spare drive to make sure it is still working properly. Some controllers offer a maintenance utility specifically for this purpose. Some may automatically test the spares on occasion.

If for whatever reason your RAID setup won't support hot sparing, you can still do the next best thing, which is what I call "cold sparing". :^) This is simple: when you buy the drives for your RAID array, buy one extra drive; keep it in a safe place near the system. If a drive ever fails in the array, you'll have to swap it out, but you *won't* have to wait for hours or days while you try to locate, order and have delivered a replacement drive. Another good reason to do this is that you will be sure that the drive you are replacing is the exact same as the original ones in the array--some arrays don't like having a drive replaced with anything but another of the exact same type.

Next: Array Expansion

Array Expansion

One of the most amazing phenomena in the world of storage is the way that hard disks shrink. You probably know what I mean: in 1998 you installed a *massive* 4 GB hard disk into your PC thinking you were set for life; 18 months later you were lamenting the fact that you had "no space on this teeny hard drive". Now you have a 40 GB drive and "can't imagine how you got by with only 4 GB". Been there, done that. :^) The same thing happens to RAID arrays--in fact, it often happens faster than it does with individual drives, since RAID arrays are usually shared by many users and it can be hard to anticipate the capacity needs of a business that is growing. This sometimes necessitates *array expansion*, adding capacity to an existing RAID array. This can be a process fraught with difficulties.

There are two main issues involved in expanding an array:

- **Physical Expansion:** You need to be able to physically put drive(s) into the system.
- **Logical Expansion:** You have to somehow get the controller to add the new drive(s) to the array so that the array is enlarged in overall size.

Physical expansion is usually the easier of the two, but not always. As discussed in some detail here, the number of bays in a case is usually limited. If you use all the available spots when you first set up the array, you won't have anywhere to add new drives. This happens more often than you might think, because the performance of the array will generally be higher if you use more smaller drives instead of fewer larger ones, so many people use as many drives as will fit in the case. If you max out your case's physical capacity, your only choices are to add an (expensive) external enclosure, or replace the existing drives with higher-capacity models (which you may want to do anyway; see below.)

Logical expansion comes down to the RAID level you choose to employ, and the features of your RAID controller. Some will allow a drive to be added to a striped set (with or without parity) while others will not. If your controller supports expansion you are in good shape--the controller will let you add the drive and then will reconfigure the array to use the added storage. If your controller *doesn't* support this advanced feature, you will have to destructively "disassemble" the smaller array and assemble the new, larger array. This will generally result in all the data on the array being lost, so you must be sure you have it properly backed up--and *tested*--before you do this.

Even if your controller supports logical expansion and you have the room for the drives, you should think twice about doing it for the following reasons:

- **Size Matching and Capacity:** With few exceptions, all RAID arrays are constrained by the size of the smallest drive(s) in the array--any drives larger than this will not utilize their additional capacity. This means that if you have an array with five 9 GB drives and you add a spankin' new 36 GB drive to it, surprise! 75% of the drive will sit there wasted.
- **Performance:** Older drives that are much smaller than new ones are also slower than new ones. You will lose most of the benefit of a new drive if you put it in an array with a bunch of smaller ones.
- **Reliability:** If the existing drives are old enough to be too small, they are probably at least *nearing* the end of their service life. This means they will become increasingly likely to fail.

For all of the reasons above, many RAID users never expand their arrays: they just use all the drive slots on the case, and when the drives get old, they replace them. They give up the storage of the old array, but with drive capacities doubling every year or two, this can make less difference than you might think. If you pay $X for four drives of a given capacity today, in three or four years that same $X will buy a single drive with capacity as high as the entire array! There are arrays still in use from the mid-1990s that have 4 GB drives in them; a new array with disks comparable to what these drives cost five years ago would dwarf the capacity of the old array.

If you don't want to expand an existing array, and don't want to replace it either, you can just create a new array alongside it. If there is room in the server, add the new drives and create a new, separate array from them. Then use the older array in a backup or secondary role, for older or less-important data. Another alternative is to just set up a new array on a different machine. Since servers are networked to allow many users to access them anyway, this will let the extra capacity be used by everyone on the network without necessitating changes to the existing system. In addition, it removes a single server from being a sole point of failure, to some extent: you won't have all your array eggs in the same basket. The drawbacks are obvious: the cost of another entire machine (unless you already have one), other hardware costs, and the effort to set it all up.

Next: Hard Disk BIOS and Capacity Factors

Hard Disk BIOS and Capacity Factors

The operation of your hard disk drives is controlled by the interface from the system to the hard disk itself. This interface is the conduit for addressing instructions and commands, sent to the hard disk to select what data is requested, and then a conduit for the data itself, flowing to and from the system. The system BIOS plays a role in the operation of the hard disk, as it provides the standard software routines that allow applications and operating systems such as DOS to access the hard disk. It is also the cause of many configuration and capacity limitation problems that many users have when setting up their hard disks, especially newer ones on older systems.

This section takes a look at issues related to how the BIOS and operating system interact with the hard disk, and BIOS-related issues and problems. This includes a full look at the many capacity limitations inherent in using IDE/ATA interface drives, and other BIOS restrictions on hard disk capacity. Many of the items in this section are really of relevance only to IDE/ATA drives; SCSI drives use their own BIOS and a different addressing mechanism from IDE/ATA, and so suffer from fewer of these problems. However, some BIOS issues affect SCSI as well, because of problems associated with operating system limitations.

Next: BIOS and the Hard Disk

BIOS and the Hard Disk

The BIOS and operating system play an important role in how your hard disk is used. While the BIOS itself has taken more of a "back seat" role to direct access by the operating system over the last few years, it is still there "in the mix" in several ways. This section takes a brief look at the impact of the BIOS on hard disk setup and access. See the section on the BIOS for more complete details on how it works.

Next: Role of the BIOS in Hard Disk Access

Role of the BIOS in Hard Disk Access

The system BIOS is the lowest-level interface between the hardware of your system and the software that runs on it. It has several significant roles that it plays in the control of access to hard disks:

- **BIOS Interrupt Routines:** In order to ensure the interoperability of various hardware and software products, the BIOS of the system is tailored to the needs of its hardware, and provides a standard way of letting software addressing the hardware. These are called BIOS services and are used by many operating system and application programs. They provide a uniform interface to the hard disk, so applications don't need to know how to talk to each type of hard disk individually. (Many newer operating systems today regularly bypass these BIOS services but still may use them for compatibility purposes.)
- **Hard Disk Detection and Configuration:** Standard IDE/ATA hard disks are configured in the BIOS using various BIOS settings. Modern BIOSes can in fact interrogate modern IDE/ATA disks to determine these parameters, and automatically configure them.
- **Hard Disk Interface Mode Support:** The BIOS, working with the system chipset on the motherboard and the system I/O bus, controls which types of interface modes can be used with the hard disk. This refers specifically to features such as high-performance PIO modes, DMA modes, and block mode.

Next: The Int13h Interface

The Int13h Interface

When the operating system or an application wants to access the hard disk, it traditionally employs BIOS services to do this. The primary interface to the BIOS has been the software interrupt known as *Int13h*, where "Int" stands of course for interrupt and "13h" is the number 19 in hexadecimal notation.

The Int13h interface supports many different commands that can be given to the BIOS, which then passes them on to the hard disk. These include most anything that you would normally want to do with a disk--reading, writing, formatting, and so on. Int13h has been the standard for many years because it has been used by DOS for ages. It is only in recent years that the limitations of this old interface have caused it to be abandoned in favor of a new way of addressing hard disks, as described below.

Using Int13h requires the invoking program to know the specific parameters of the hard disk, and provide exact head, cylinder and sector addressing to the routines to allow disk access. The BIOS uses the geometry for the hard disk as it is set up in the BIOS setup program. The Int13h interface allocates 24 bits for the specification of the drive's geometry, broken up as follows:

- 10 bits for the cylinder number, or a total of 1,024 cylinders.
- 8 bits for the head number, or a total of 256 heads.
- 6 bits for the sector number, or a total of 63 sectors (by convention, sectors are numbered starting with one instead of zero, so there are only 63).

This means that the Int13h interface can support disks containing up to approximately 16.5 million sectors, which at 512 bytes per sector yields a maximum of 8.46 GB (or 7.88 GiB). Of course, twenty years ago when this methodology was developed, an 8 GB hard disk was Buck Rogers fantasyland material; a 10 MB hard disk was a luxury. Today, for many PC users, an 8 GB hard disk is "a bit on the small side". :^) As a result, the Int13h interface has finally come to the end of its usefulness in modern systems, and has been replaced with a newer interface called Int13h extensions. Int13h still may be used by DOS and some other older operating systems, and for other compatibility purposes.

Next: Int13h Extensions

Int13h Extensions

As discussed in the section on the standard Int13h BIOS interface, that older standard has an important limitation that has become a serious issue for PC upgraders over the last few years: it uses 24 bits of addressing information, and as such can only handle drives that contain up to approximately 16.5 million sectors, which at 512 bytes per sector yields a maximum capacity of 8.46 GB (or 7.88 GiB). As modern drives approached 8 GB in size, hardware and operating system makers all realized that they had a problem here: something had to be done to allow access to the larger hard disks of the future.

When a bridge is too narrow to handle increased traffic, the usual solution is to widen it, and that's exactly what was needed here: to widen the access path from 24 bits to something larger. Unfortunately, it was not possible to expand the existing Int13h BIOS interface. The reason is that if this were done, a lot of older hardware and software would stop working. Making changes that cause millions of older hardware and software products to stop working is *not* how you win friends in the PC world. :^)

Instead, a new interface was developed to replace Int13h: these routines are called *Int13h extensions*. This new interface uses 64 bits instead of 24 bits for addressing, allowing a maximum hard drive size of 9.4 * 10^21 bytes. That's 9.4 trillion gigabytes! I'm sure that when the original Int13h interface was developed, nobody ever expected us to hit 8 GB drives as fast as we did. Still, even with the rapid pace of technological advancement, I'd say we're pretty safe with 9.4 trillion gigabytes as a limit. If not, I'll be pleased as punch to move to a still newer interface in exchange for a hard drive that big. ;^)

There's a catch to these Int13h extensions of course: they are different from the old way of doing things, and therefore support for them must be incorporated into several key areas of the system. This includes the system BIOS and the operating system. For more information on this, see this section.

For more information on the Int13h interface limitation and the problems it causes, see the section on the Int13h interface size barrier.

Next: Direct Disk Access (Bypassing the BIOS)

Direct Disk Access (Bypassing the BIOS)

As I mentioned in the section on the role of the BIOS, one of its traditional responsibilities is acting as a "middleman" between the operating system and the hard disk. Well, there's a problem with middlemen: they may be convenient in some situations, but they are *inefficient*. Having the BIOS in the middle of every transaction can hurt performance, and make more advanced transfer methods difficult to implement.

As a result, many more advanced operating systems take responsibility for data transfer between themselves and the hard disk away from the BIOS. For example, modern versions of Windows employ their own 32-bit protected mode access routines for the hard disk, which are faster and more efficient than using the default BIOS code. This has now in fact become pretty much the standard way of doing things.

Unfortunately, bypassing the BIOS does not mean that we are able to avoid its problems or limitations. :^) Traditional BIOS routines are still needed for compatibility with DOS and older programs. The BIOS is also responsible for compatibility with older hardware, and also for its other roles. And unfortunately, all that means that we still have to deal with the BIOS's capacity barriers.

Next: Two and Four Disk BIOS IDE Support

Two and Four Disk BIOS IDE Support

From the early 1980s until about 1993 or so, many PCs supported only two hard disks. For the first decade or so that hard disks were used in PCs, partitioning of a drive into multiple volumes was not really done; as such, the two hard disks in a system were usually just "C:" and "D:". BIOS code writers would often refer to these two drives as "C" and "D". (Though in reality, drive letters are dynamically assigned, and are not a matter of hardware at all.)

All modern BIOSes, since about 1994, have supported four IDE/ATA hard disks: two channels (primary and secondary) and a master and slave device on each. Some actually support more, and of course, additional IDE/ATA channels can be added to existing systems.

Any system that supports only two hard disks may theoretically be upgradable to allow four-disk support. In practice, any system that old has at best a 486 processor and probably very little system memory. Such a PC is so obsolete by today's standards that it really isn't worth the effort. Find a low-demand use for the machine and get a more modern system; even a used machine a few years old will be several notches above such an old machine.

Next: IDE/ATA Disk BIOS Settings

IDE/ATA Disk BIOS Settings

Since the system BIOS on virtually every PC provides native support for IDE/ATA hard disks, there are a number of parameters that can be set to tell the BIOS what hard disks are in the system, and how to control them. Each hard disk in the system will have its own settings, so there is one set for the primary master, one for the primary slave, and so on. This normally applies only to IDE/ATA hard disks; SCSI hard disks are configured through their host adapter and built-in SCSI BIOS.

Most modern BIOSes support hard disk autodetection, which allows the BIOS to interrogate each hard disk to determine its logical geometry, supported transfer modes and other information. This can be done either at setup time or dynamically each time the machine is booted, depending on the BIOS. This is described in detail here.

Dynamic autodetection is the standard way of setting up modern drives, especially since drives over 8 GB in size cannot be described using traditional IDE/ATA BIOS geometry parameters. In some situations, especially with older PCs, you may still set some drive parameters manually. The following are the settings normally found in the BIOS setup program for configuring IDE/ATA hard disks. Since these are described in full detail in the BIOS chapter, I include only a very brief description of each here. Note that on modern systems some of the oldest compatibility settings may not even be present any more:

- **Disk Type:** Originally used to allow you to pick your hard disk from a predefined list, this is now used to control automatic or manual parameter setup for the drive. (The old tables describing ancient drives are now often not even present on many modern BIOSes, since they no longer serve any purpose.)
- **Size:** This is the size of the drive in decimal megabytes. It is calculated from the other parameters.
- **Cylinders:** The number of logical cylinders on the disk. The value used depends on whether BIOS translation is enabled, in some BIOSes. For a drive set to "Auto", no number may be shown.
- **Heads:** The number of logical heads on the disk. The value used depends on whether BIOS translation is enabled, in some BIOSes. For a drive set to "Auto", no number may be shown.
- **Sectors:** The number of logical 512-byte sectors in each logical track on the disk. This is usually 63 for modern drives. Again, for a drive set to "Auto", you may not see a number here.
- **Write Precompensation:** A compatibility setting that specifies at which cylinder number write adjustments should be made, for very much older drives.
- **Landing Zone:** The cylinder where the heads are parked by the BIOS when the drive is shut off; not used on modern drives since they automatically park their heads.
- **Translation Mode:** The BIOS translation mode being used, for support of hard disks over 504 MB. Translation issues are discussed in detail here.
- **Block Mode:** Controls the BIOS's ability to perform disk transfers in blocks.
- **PIO or DMA Mode:** The programmed I/O mode or DMA mode used to perform transfers to and from the hard disk.
- **32-Bit Transfer Mode:** Controls the use of higher-performance 32-bit data transfers.

Next: Hard Disk Size Barriers

Hard Disk Size Barriers

One of the most common problems people have with hard disks, especially when trying to add a new hard disk to an older system, is the frustration of finding that not all of the hard disk is actually accessible. This is almost always due to BIOS and operating system issues that are a result of short-sighted planning done by the people that invented hard disk structures, access routines and operating systems many years ago. In some cases they are due to actual hardware or software bugs that are not detected until hard disks grow in size beyond a certain point.

Fortunately, there are now solutions to most of these problems. This section takes a complete look at these issues so you can finally understand what these barriers are all about. Hopefully. :^) And then also hopefully, solve the problem through hardware and software changes.

Each of the barriers is described in its page title with a name that briefly summarizes the reason for the barrier, and also the capacity figures generally associated with that barrier. The capacity limits are given in both decimal (GB) and binary (GiB) formats since some people look for specific decimal or binary numbers when identifying a size barrier. For a full discussion of the differences between decimal and binary measurements, see this page.

Note: Most of the issues discussed here are due to BIOS issues, and hence of relevance primarily to IDE/ATA hard disks and not SCSI disks. Some though are also relevant for SCSI drives, especially ones that are due to operating system limits.

Note: The pages in this section are focused primarily on explaining the nature of the various hard disk barriers and how they come about. To avoid duplication, I do not discuss how to deal with size barriers here, because the techniques are common to many different barrier types. Look at this section for details on overcoming size barriers.

Next: Older Size Barriers

Older Size Barriers

The first widely-publicized hard disk barrier was the infamous 504 MiB / 528 MB barrier that showed up in the mid 1990s. Though not widely known, there were a number of older capacity barriers that affected hard drives before the 504/528 limit made so many PC headlines. Most of these got little attention, most likely because there were fewer PC users then, but also because upgrading was less common. (Upgrades are often the cause of barrier troubles.)

At any rate, these have no relevance at all to modern computing, but might be of interest to those who have much older machines. I mention them here for completeness, if for no other reason, but I will describe them only briefly since again, they have no impact on modern PCs:

- **PC/XT Parameter (10.4 MiB / 10.9 MB) Barrier:** The very first PC to use a hard disk was IBM's PC/XT. This machine was specifically designed to use a particular type of disk with 312 cylinders, 4 heads and 17 sectors per track. As such, it was hard-wired to the approximately 10 MB capacity of those very early drives.
- **FAT12 Partition Size (16 MiB / 16.7 MB) Barrier:** The first FAT format used for hard disks was the 12-bit FAT12 partition type (still used for floppy disks). This allowed a maximum of 4,086 clusters of 4,096 bytes, for a total of 16,736,256 bytes per disk.

- **DOS 3 (32 MiB / 33.6 MB) Barrier:** To get around the 16 MiB barrier, DOS 3.x was altered when the IBM PC/AT was introduced with larger drives. The first support for the FAT16 file system was added. However, a new barrier was introduced by the rather limited way in which FAT16 was originally implemented: cluster size was set to 2,048 bytes, and only 16,384 FAT entries were allowed, fixing maximum capacity at around 32 MiB. The ability to have multiple partitions was introduced at around the same time, but each partition could only be 32 MiB or less.
- **DOS 4 (128 MiB / 134 MB) Barrier:** DOS 4.x improved over DOS 3.x by allowing 65,526 clusters instead of 16,384, quadrupling maximum partition size to about 128 MiB. Cluster size was still fixed at 2,048 bytes.

As you can see, most of these early limits were not due to BIOS issues, but rather some very short-sighted thinking on the part of the MS-DOS design team, which was apparently only trying to stay a year or two ahead of the hard disk technology curve! In addition to the above, there was a 512 MiB barrier as well, caused by the change made with DOS 5. That operating system changed DOS 4 by allowing cluster size in a single partition to increase to 8,192 bytes, allowing a theoretical maximum partition size of about 512 MiB or 537 MB. However, most systems that had drives large enough for this to be an issue were not able to use that full size due to the slightly smaller 504 MiB / 528 MB barrier, caused by BIOS issues.

Next: The 1,024 Cylinder (504 MiB / 528 MB) Barrier

The 1,024 Cylinder (504 MiB / 528 MB) Barrier

The most (in)famous hard disk barrier of all time was probably the 504 MiB limitation for standard IDE/ATA hard disks, which started showing up in systems starting in around 1994. Today hard disks are so much larger and this barrier is so many years in the past, that many PC users aren't aware of how much trouble this first big size barrier caused. At the time though, this barrier was a very new experience for PC users and technicians alike.

Due to this barrier, a hard disk with a size over 504 MiB will normally appear only as having 504 MiB under some circumstances. This problem is a result of combining the geometry-specification limitations of the IDE/ATA standard and the BIOS Int 13h standard. This barrier is alternatively referred to as the 504 MB or the 528 MB barrier, depending on whether you are looking at binary or decimal megabytes.

OK, now I'll try explaining it in English. :^) Every hard disk is presented to the BIOS through its geometry, which tells the BIOS how many cylinders, heads and sectors the disk uses. This is the way that the hard disk is addressed, when a sector needs to be read or written. The geometry that a modern hard disk uses for the BIOS is normally its logical geometry, and not the physical geometry actually inside the hard disk assembly.

Various software structures reserve a certain amount of space for specifying each of the three parameters that make up the hard disk geometry. The amount of space reserved is dictated by standards that control how IDE/ATA hard disks are supposed to work, and also how the BIOS sees hard disks through its Int13h software interface.

The problem is that due to (very) poor planning and coordination, the standards are not the same; they each reserve different numbers of bits for the geometry. In order to use an IDE/ATA hard disk with the standard BIOS disk routines then, the limitations of both standards must be observed, which means that only the smaller of each geometry number can be used. Here is how the two standards allocate bits for the geometry:

Standard	**Bits For Cylinder Number**	**Bits for Head Number**	**Bits for Sector Number**	**Total Bits for Geometry**
IDE/ATA	16	**4**	8	28
BIOS Int 13h	**10**	8	**6**	24
Combination (Smaller of Each)	**10**	**4**	**6**	**20**

Since each geometry figure is a binary number with a number of bits indicated above, this means that the maximum number supported for any parameter is 2^N, where N is the number in the table above. So for example, under IDE/ATA, 2^16 or 65,536 cylinders are supported. We can then multiply all the figures together to get a total number of sectors supported, and then multiply that by 512 bytes (per sector) to get the maximum supported capacity:

Standard	**Maximum Cylinders**	**Maximum Heads**	**Maximum Sectors**	**Maximum Capacity**
IDE/ATA	65,536	**16**	256	128 GiB
BIOS Int 13h	**1,024**	256	**63**	7.88 GiB
Combination (Smaller of Each)	**1,024**	**16**	**63**	**504 MiB**

Note: The BIOS Int 13h limit for sectors is 63, and not 64, because by convention sectors are numbered starting at 1 and not 0.

As you can see, the 504 MiB figure is just 1,024 * 16 * 63 * 512, which equals 528,482,304. The problem is the combination of the limitations of the two standards. Due to the 16-head limitation of IDE/ATA, no IDE hard disk is ever specified with more than 16 logical heads; they always have a large number of cylinders instead. The problem is that when you put the disk in a machine with a standard, non-translating BIOS, it can't see more than 1,024 of the cylinders. There are several different ways that the system may react to a drive too large for it to handle.

The normal solution to the 504 MiB problem is to use a system that supports BIOS translation. This gets around the problem by using what is little more than a software trick, but it does work. Software drive overlays will also avoid the problem, but at a cost. See this section for a full discussion on dealing with this disk size barrier.

Next: The 4,096 Cylinder (1.97 GiB / 2.11 GB) Barrier

The 4,096 Cylinder (1.97 GiB / 2.11 GB) Barrier

As discussed in great detail in the section discussing the 504 MiB barrier, the basic problem with BIOS-related capacity barriers is that the normal system BIOS interface on older PCs is not designed to handle hard disks that employ over 1,024 cylinders. Every hard disk made today uses more than 1,024, which causes a drastic reduction in visible capacity due to this limitation. See that section if you need to get an understanding of this basic matter.

Systems that use an enhanced BIOS are able to employ translation to get around the 1,024 cylinder limitation. However, some BIOSes, despite supporting translation, will again choke if the number of cylinders exceeds 4,095, causing the same problems with the 504 MiB barrier to occur all over again. 2^12 is 4,096, which means that if you go beyond 4,095 cylinders on a drive, the number requires a 13th bit to properly represent it (for example, 4,097 in decimal is 1000000000001 in binary). This should be no problem, but due to poor BIOS code writing on some systems, only 12 bits of the cylinder number are recognized. The actual limitation is 4,096 * 16 * 63 * 512 bytes, which is about 1.97 binary gigabytes or 2.11 decimal gigabytes. Some BIOSes with this problem will show a disk with more than 4,096 cylinders as being 1.97 GB, while others will show it as substantially less. For example, 4,097 cylinders may show up as only 1 cylinder if the 13th bit is just ignored and the lower-order 12 bits are used by themselves!

Note: Do not confuse this capacity barrier with the other capacity barrier which is exactly 2 binary gigabytes. That one is a file system issue and is unrelated to the BIOS matter we are discussing here.

This particular size barrier began to show up on systems in 1996. The options for solving it are similar to those for dealing with the 504 MiB limitation.

Next: The FAT16 Partition Size (2.00 GiB / 2.15 GB) Barrier

The FAT16 Partition Size (2.00 GiB / 2.15 GB) Barrier

The 2 GiB capacity barrier is, unlike most of the other barriers discussed here, a file system problem that has nothing to do with the BIOS. It is different than the 1.97 GiB barrier, which *is* BIOS-related, and is in many ways most similar to the older BIOS barriers that preceded the 504 MiB barrier.

The 2 GiB capacity barrier is a limitation on the size of disk volumes in the FAT16 file system. Due to the way that disks are set up using clusters, it is not possible to have more than 2 GiB in a single partition when using the DOS or Windows 3.x operating systems, or the first version of Windows 95 (sometimes called "Windows 95A"). Under Windows NT, the limit is 4 GiB instead of 2 GiB when using FAT partitions (NTFS partitions do not have this limitation). This is all discussed in great detail in the section on file system structures.

Note: Using 4 GiB FAT partitions under Windows NT requires the use of 64 kiB clusters. This is supported but non-standard, and will result in problems if you try to set up a system using other operating systems in addition to Windows NT.

If you put a hard disk over 2 GiB into a machine that is using regular FAT (16-bit FAT) under DOS, Windows 3.x or the first version of Windows 95, you can use all of it--assuming that you aren't limited by one of the other BIOS-related barriers mentioned in adjacent sections. However, to access the full contents of the disk, you must partition it into multiple pieces. Since this limitation is a function of the operating system, it affects IDE/ATA and SCSI hard disks equally.

This limitation does not apply to disks formatted using the newer FAT32 file system. FAT32 is an enhancement that was in fact created specifically to get around this problem, and was introduced in Windows 95 OEM SR2. It is also supported in Windows 98, Windows ME and Windows 2000. There is also the NTFS file system, supported by Windows NT and Windows 2000, which uses a completely different set of structures and can have truly enormous-sized partitions (tens of gigabytes).

Next: The 6,322 Cylinder (3.04 GiB / 3.26 GB) Barrier

The 6,322 Cylinder (3.04 GiB / 3.26 GB) Barrier

This is one of the most obscure size barriers around, apparently affecting only a small percentage of systems. It appears that on these units, the BIOS cannot handle hard disk geometry that has more than 6,322 cylinders. Attempting to set a higher cylinder value than 6,322 may cause the PC to hang. This typicall fixes capacity on such systems to about 3.04 GiB or 3.26 GiB (6322 cylinders * 16 heads * 63 sectors * 512 bytes).

Note: I have absolutely no idea what the significance of the "6,322" number is at all. It is not a "round number" in either decimal or binary!

Getting around this barrier is typically accomplished in a manner similar to how other BIOS barriers are tackled. I have not encountered this particular barrier myself on any of my systems.

Next: The Phoenix BIOS 4.03 / 4.04 Bug (3.05 GiB / 3.28 GB) Barrier

The Phoenix BIOS 4.03 / 4.04 Bug (3.05 GiB / 3.28 GB) Barrier

This is another rather obscure size barrier, and one that differs from most of the others I have described in this area of the site. It is different because it is not due to any inherent BIOS design issue, nor due to an operating system limit or characteristic. It occurs due specifically to a programming error or *bug* in a few types of systems made in the mid-to-late 1990s. Some systems that use Phoenix BIOSes, versions 4.03 or 4.04, have a problem with the BIOS routine that calculates the size of hard disk drives. Note that BIOS code is initially written by the BIOS maker, and is subsequently tailored by specific system or motherboard manufacturers. This means that some specific implementations of these BIOS versions may not have this bug while others will.

This problem is also strange because the barrier actually isn't just a single value; it seems to depend on the values of the geometry parameters, and behavior can be different based on the values entered. Assuming standard IDE head and sector values of 16 and 63 respectively, the cylinder field can have a maximum value of 6,349 without any problems, resulting in a maximum capacity of 3.05 GiB or 3.28 GB. If a cylinder value of 6,350 to 8,322 is used the BIOS setup program may lock up. Cylinder values of 8,323 to 14,671 apparently work but the displayed drive size is incorrect.

Subsequent versions of this BIOS code have of course corrected this bug, which occurred several years ago. If you still have a system exhibiting this problem, you may be able to get a BIOS upgrade to correct the problem.

Next: The 8,192 Cylinder (3.94 GiB / 4.22 GB) Barrier

The 8,192 Cylinder (3.94 GiB / 4.22 GB) Barrier

After the discovery of the 504 MiB BIOS barrier, the normal way of getting around that problem was to make use of BIOS geometry translation. (This continued until hard drives exceeded about 8 GB in size and the whole IDE/ATA geometry scheme had to be abandoned altogether.) In a nutshell, this translation works by dividing the hard disk's number of cylinders by a binary number such as 2, 4, 8 or 16, and multiplying the number of heads by the same number. This lets the number of cylinders that the BIOS sees fall below the Int13h limit of 1,024. This translation however causes a problem in some systems when using a hard disk over about 4 GB in size.

Note: To understand how translation causes this particular new barrier to arise, you need to understand how BIOS translation works. See here for an explanation if you are not familiar with this.

When the number of cylinders on the drive is between 8,192 and 16,383, the number typically used for translation is 16. Here's an example of how this might work with a theoretical 6.4 GB hard disk if it used the normal way that IDE/ATA drives are specified, with 16 heads and 63 sectors per track:

	Cylinders	Heads	Sectors	Capacity
IDE/ATA Limits	65,536	16	256	137 GB
Hard Disk Logical Geometry	**12,496**	**16**	**63**	**6.45 GB**
BIOS Translation Factor	**divide by 16**	**multiply by 16**	--	--
BIOS Translated Geometry	**781**	**256**	**63**	**6..45 GB**
BIOS Int 13h Limits	1,024	256	63	7.88 GB

This *should* actually work just fine; it overcomes the BIOS issues and results in geometry that falls within acceptable limits. However, there's an unfortunate gotcha that was discovered when drives first exceeded 8,192 cylinders in around 1997: MS-DOS and early versions of Windows choked when presented with a drive that had (apparently) 256 heads! Thus, this is actually a barrier that is due to both the operating system and the system BIOS: the operating system should have been able to handle 256 heads, but the BIOS was creating the problem due to its translation.

It was decided that the easiest way to deal with this problem was to change the way the BIOS did translation. As a result, BIOSes stopped creating translated geometries that had 256 heads. One common way that this was done was to use 15 as the translation factor instead of 16, resulting in this sort of conversion:

	Cylinders	Heads	Sectors	Capacity
IDE/ATA Limits	65,536	16	256	137 GB
Hard Disk Logical Geometry	**12,496**	**16**	**63**	**6.45 GB**
BIOS Translation Factor	**divide by 15**	**multiply by 15**	--	--
BIOS Translated Geometry	**833**	**240**	**63**	**6..45 GB**
BIOS Int 13h Limits	1,024	256	63	7.88 GB

Of course, if you have a BIOS that doesn't know about the 256 head problem, you will need to address this with either a hardware or software solution. To help avoid some of these problems, many hard disk manufacturers also changed their specified geometries to use only 15 heads instead of 16. So instead of the example drive above being specified with 12,496 cylinders, 16 heads and 63 sectors, it might have been 13,329 cylinders, 15 heads and 63 sectors. With these parameters, even if the BIOS uses a translation factor of 16, the resulting number of heads will be only 240.

Next: The 240 Head Int 13 Interface (7.38 GiB / 7.93 GB) Barrier

The 240 Head Int 13 Interface (7.38 GiB / 7.93 GB) Barrier

The Int13h interface limit normally restricts some systems to 7.88 GiB or 8.46 GiB as a result of the limits of the BIOS Int13h interface: 1,024 cylinders, 256 heads and 63 sectors of 512 bytes. (I discuss this very important size barrier in some detail here, and you may want to read that section before you read this one.) However, in some systems the Int13h interface restriction results in a smaller limit: only 7.38 GiB (7.93 GB).

The reason why this occurs is related to a *different* size barrier problem (sigh, can't these engineers get their acts together? :^)) As I described in the discussion of the 8,192 cylinder limit, DOS and some Windows versions cannot handle translated geometry that specifies 256 heads. To get around this, some BIOSes change their translation method so that only 240 heads are presented to the operating system. This fixes the "256 head problem" but shaves some capacity off the Int13h limit. The 1,024 cylinder and 63 sector restrictions remain, but with only 240 heads the maximum drive capacity becomes 1024 * 240 * 63 = 15,482,880 sectors of 512 bytes, or 7,927,234,560 bytes.

In practical terms, there isn't any difference in how this barrier is handled than the standard Int13h problem is tackled. You still need to use Int13h extensions; see the discussion of the Int13h interface barrier for more.

Next: The Int 13 Interface (7.88 GiB / 8.46 GB) Barrier

The Int 13 Interface (7.88 GiB / 8.46 GB) Barrier

This barrier, often just called the "8 GB barrier", is one of the most important in the hard disk world. Now that hard disk capacities have moved into the tens of gigabytes and beyond, it gets most of the attention that the old 504 MiB / 528 MB barrier used to get in the mid-to-late 1990s. Many people run into this particular barrier as they attempt to upgrade systems originally purchased in the late 1990s with hard disks of 1 GB to 8 GB or so in size.

Like most of the others, this barrier is also based on a BIOS limitation. It is a tougher nut to crack than most of the smaller-valued barriers, however. The reason for this is that with this particular barrier, we have actually come up against one of the traditional limits of how hard disks are used in the PC: the Int13h interface. That standard allocates 10 bits for the cylinder number (and thus a maximum of 1,024 cylinders), 8 bits for the head number (maximum of 256) and 6 bits for the sector number (maximum of 63, since the number 0 is not used). Multiplying these together, and assuming the standard of 512 bytes per sector, you get a maximum of 8,455,716,864 bytes. This is the largest hard disk size that can be addressed using the standard Int13h interface.

Unlike the old 504 MiB barrier, there is no translation that can get around this because it isn't the result of a *combination* of limitations like the 504 MiB barrier is. It is in fact the limit of how hard disks can be represented using the BIOS Int 13h routines used by DOS and applications to access the hard disk. To get around this barrier, we must change the way hard disks are accessed entirely. This means leaving Int13h behind and using Int13h extensions.

Note: Int13h extensions require support from both the BIOS and the operating system. Some older operating systems do not support Int13h extensions, and there are no plans to provide it for them. In particular, all versions of straight non-Windows DOS (6.22 and earlier), and Windows NT version 3.5 will not support Int13h extensions and cannot use hard disks over 8.4 GB in size.

Note: Some systems have a smaller Int13h capacity limit due to the use of modified translation to avoid presenting geometry with 256 heads to the operating system. See here for details.

Next: The Windows 95 Limit (29.8 GiB / 32.0 GB) Barrier

The Windows 95 Limit (29.8 GiB / 32.0 GB) Barrier

Microsoft officially announced in 1999 that Windows 95 does not support hard disks over 32 GB in size. For that reason, I am including this in my discussion of hard disk capacity barriers. However, I must embarrassingly admit that after many months of trying to determine the reason for this exclusion, I have been unable to find out what it is! So I can't give an explanation for this limit, because I don't know myself. All I know is that Microsoft would not officially announce that Windows 95 could not handle hard disks over 32 GB if it could--at least, I don't *think* they would, would they? ;^) At any rate, you can read their knowledge base article on the subject *here* and make up your own mind. Don't expect to find any details there however, or I'd discuss them myself. (If by any chance you *do* know the reason for this barrier, please let me know, I'd certainly appreciate it.)

All Microsoft says for its description of the limit is "32 GB"; since I don't know the details behind this I don't know the exact number that the capacity limit represents. In the same knowledge base article where Microsoft says Windows 95 won't support drives over 32 GB, they say that Windows 98 (and presumbly, Windows ME) *will* support drives over 32 GB, but that a patch may be needed due to a bug in Scandisk on drives over 32 GB in size; *see here*. It may be that this Scandisk problem is the same as the one that caused Microsoft to write off large drives under Windows 95. Microsoft may have just decided it would not bother to patch Windows 95. Alternatively, they could be separate issues. Again, if you have any information on this subject, I'm all ears. :^)

Next: The 65,536 Cylinder (31.5 GiB / 33.8 GB) Barrier

The 65,536 Cylinder (31.5 GiB / 33.8 GB) Barrier

This is a relatively new hard disk barrier that showed up in early 1999. It is yet another in a long series of limits caused by the inability of a BIOS version or type to handle a particular number of cylinders, much like several smaller barriers have been. It is often called the "32 GB size barrier", which is approximately correct anyway. :^)

In this particular case, some versions of Award BIOS cannot handle drives that have more than 65,535 cylinders. Since hard disk parameters usually use 16 heads and 63 sectors, this works out to a capacity of about 33.8 GB or 31.5 GiB before trouble occurs. As of about June 1999, this problem had been corrected, so it is most likely to show up on systems purchased before that month.

I must say that I find this to be a rather strange hard disk barrier, because hard disks above about 8 GB in size no longer really *use* discrete geometry for access; they are instead addressed using LBA and a flat sector number from 0 to one less than the number of sectors on the drive. This 65,536 cylinder problem must be a remnant of some older code, or something related to compatibility with older hard drives. Regardless of its origin, many system owners will have to deal with it.

Next: The ATA Interface Limit (128 GiB / 137 GB) Barrier

The ATA Interface Limit (128 GiB / 137 GB) Barrier

To get around past hard disk barriers, most modern hard disks are now no longer addressed using discrete geometry (cylinder, head and sector numbers) but rather logical block addressing and a sector number. However, even if we go away from the problems associated with assigning some bits in an address to cylinder number and others for head number and sector number, we eventually reach the limit of the addressability of all the bits taken together. In the case of the ATA interface, 28 bits are used for the sector number interface between the operating system, BIOS and the hard disk. This means a hard disk can have a maximum of 2^28 or 268,435,456 sectors of 512 bytes. This puts the ATA interface maximum at 128 GiB or approximately 137.4 GB.

Of course, as of this writing in mid-2000, there aren't any ATA hard disks that are this large. As a result, most people don't think much about this particular barrier. But things change quickly in the hard disk world, and we'll be there before you know it. Based on the current rate of hard disk capacity improvement, I'd guess that we'll be pushing the limits of the ATA interface no later than 2002. So consider this one the "big hard disk size barrier of the future". :^)

Much as the Int13h interface barrier was a tough nut to crack, this one will be as well, and for similar reasons. Fairly significant changes will need to be made to the interface between the hard disk and the rest of the system. Seeing this barrier on the horizon, the T13 technical committee (which works on standards for the IDE/ATA interface) is working on a couple of different proposals for expanding ATA addressing from 28 bits to either 48 or 64, either of which would allow rather monstrous hard disk sizes (even the smaller 48-bit proposal would result in drive sizes a million times higher than the current limit).

This page will be updated as we get closer to hitting this barrier, and more information becomes available about how it will be addressed.

Next: BIOS Handling of "Oversized" Hard Disks

BIOS Handling of "Oversized" Hard Disks

When you put a hard disk into a machine that has a BIOS unable to handle its size, the system can react in a number of different ways. How it responds depends on the system, how old the BIOS is, and how well tested and debugged the BIOS routines are. These issues normally are a result of the hard disk having a number of cylinders larger than the maximum the BIOS supports.

These are the four most common ways that a machine with an older BIOS will handle a hard disk larger than it supports:

- **Truncation:** Many BIOSes, when presented with a logical geometry containing more cylinders than they can handle, will simply truncate the total to the maximum they support. This is usually seen in older BIOSes that don't support more than 1,024 cylinders, and also in some cases in a BIOS that maxes out at 4,096 cylinders. It is also commonly found in systems that do not support Int13h extensions--these units will typically see a drive larger than 8.4 GB as being just 8.4 GB in size. Truncation of course wastes some space on the drive, but is still far preferable to the other possibilities described below.
- **Wrap-Around:** Some very old BIOSes, assuming that the number of cylinders will always be 1,024 or below, only look at the bottom 10 bits of the cylinder number coming from the hard disk (2^10 = 1,024). As a result, when used with numbers over 1,023, they do the equivalent of counting up to 1,024 and then "wrapping around" to zero again and starting over. (This is equivalent to N modulo 1024, where N is the number of real cylinders, for those who know what that means). As an example, if you tried to use a drive with 3,500 cylinders, the BIOS would see 428 cylinders, because it would count up 1,024 three times (to yield 3,072), wrap around three times, and then end up with 428 cylinders (3,500 minus 3,072).
The same exact thing can happen to a BIOS that support only 4,096 cylinders: it may only look at the bottom 12 bits. This means that in some cases you can put a 2.5 GB hard disk into your system and end up with only about 400 MB of usable space showing up. This is unfortunately a common failure mode with BIOSes that don't support more than 4,096 cylinders.

Note: Some BIOSes that support translation will do this wrapping around if you disable translation. When you turn it on again, the problem may go away.

- **"Ignorance":** Some BIOSes will report the true number of logical cylinders that the drive has, making you think your system supports the full size of the hard disk. Really, the BIOS just has no clue what it is seeing. When you go to partition and format the hard disk, you will be stuck with the same limit (which can be both confusing and frustrating). This is usually seen with older BIOSes and the 1,024 cylinder limitation.
- **Failure:** Some BIOSes will totally lock up if you try to use them with a disk larger than they can support. These are actually pretty uncommon, fortunately. They are also more common with some of the larger hard disk barriers and also with some of the more obscure ones.

Next: BIOS Translation Modes

Hard Disk BIOS Translation Modes

One of the most important techniques used to break the 504 MiB disk size barrier that results from the combination of BIOS and IDE/ATA hard disk restrictions is the use of BIOS translation. This can be a rather confusing subject and so this section discusses in detail the various types of BIOS translation that are used in PCs. A BIOS that supports the extended CHS and/or LBA modes is often said to be an *enhanced BIOS*.

Note: For completeness, and to assist those working with older hardware, I maintain the descriptions of translation modes in some detail here. Realize however that on most modern systems they are no longer of much relevance. Drives over 8.4 GB in size, which is what modern PCs use, are generally accessed by setting the BIOS to dynamically autodetect their size at boot time, and then using logical block addressing. Most of the issues that PC users once had to deal with in terms of tweaking BIOS translation modes are no longer necessary for modern drives over 8.4 GB in size, because these drives are no longer address using cylinder, head and sector numbers.

Note: Little of this section has any relevance when using only SCSI hard disks. They are not generally subject to the BIOS limitations that are overcome through translation.

Next: Normal / Standard CHS Mode

Normal / Standard CHS Mode

The normal or default mode used by "regular" hard disks that are below 504 MiB in size is called variously *Normal* or *CHS* mode, where "CHS" stands for "cylinder, head, sector", the three parameters used in hard disk geometry specifications. In this mode, there is no translation done at the BIOS level, and the logical geometry presented by the disk is used by the BIOS directly. Remember that this is still logical geometry. The actual physical geometry is known only to the disk controller.

Each hard disk using CHS mode is limited to 1,024 cylinders, 16 heads and 63 sectors, or 504 binary megabytes. This is also the only mode available on older BIOSes, from before about 1994. When hard disks above 504 MiB in size are used with one of these older BIOSes, the infamous 504 MiB barrier is hit.

Next: Extended CHS (ECHS) / Large Mode

Extended CHS (ECHS) / Large Mode

Extended CHS, also called *ECHS* or *large mode* in some BIOSes, uses BIOS translation to get around the 504 MiB size barrier inherent in standard CHS mode. It's kind of amusing to realize this, but the BIOS translation that is usually used to get around the 504 MiB barrier is not a great innovation of any sort. In fact, it's basically a hack. :^) It's a trick that is employed to get around a problem.

The idea behind translation is as follows. Recall that the 504 MiB barrier is a combination of the limitations of the IDE/ATA standard and the BIOS Int 13h routines, due to the different limits they place on the numbers of cylinders, heads and sectors allowed for a drive. This table shows how the 504 MiB barrier comes about:

Standard	**Maximum Cylinders**	**Maximum Heads**	**Maximum Sectors**	**Maximum Capacity**
IDE/ATA	65,536	**16**	256	128 GiB
BIOS Int 13h	**1,024**	256	**63**	7.88 GiB
Combination (Smaller of Each)	**1,024**	**16**	**63**	**504 MiB**

As you can see, the IDE/ATA standard allows for many more cylinders than the BIOS does, and the BIOS allows for many more heads than IDE/ATA does. (In practice, no IDE/ATA hard disk ever specifies more than 63 logical sectors--despite the theoretical limit of 256--for the exact reason that the BIOS's limit is 63. If they did go over 63, this would confuse matters even more). Again, remember that these are logical disk parameters, not physical ones.

BIOS translation works by having the BIOS act as a "middleman" of sorts between the IDE/ATA hard disk and the standard BIOS Int 13h, and by taking advantage of the fact that one standard allows more heads than the other but fewer cylinders. The BIOS takes the logical geometry that the hard disk specifies according to the IDE/ATA standard, and *translates* it into an equivalent geometry that will "fit" into the maximums allowed by the BIOS Int 13h standard. This is done by dividing the number of logical cylinders by an integer, and then multiplying the number of logical heads by the same number. The technique is sometimes called *bit shift translation* (since the multiplication and division is done by shifting the cylinder and head bits).

This is hard to understand, so here is an example (you may find referring to the table immediately below helpful when reading this). Let's take the case of a 3.1 GB Western Digital Caviar hard drive, AC33100. This drive actually has a capacity of 2.95 binary GB, and logical geometry of 6,136 cylinders, 16 heads and 63 sectors. This is well within the bounds of the IDE/ATA limitations, but exceeds the BIOS limit of 1,024 cylinders. The BIOS picks a translation factor such that dividing the logical number of cylinders by this number will produce a number of cylinders below 1,024. Usually one of 2, 4, 8, or 16 are selected; in this case the optimal number is 8. The BIOS then divides the number of cylinders by 8 and multiplies the number of heads by 8. This results in a translated geometry of 767 cylinders, 128 heads and 63 sectors. The capacity is of course unchanged, and the new geometry fits quite nicely into the BIOS limits:

	Cylinders	**Heads**	**Sectors**	**Capacity**
IDE/ATA Limits	65,536	16	256	128 GiB
Hard Disk Logical Geometry	**6,136**	**16**	**63**	**2.95 GiB**
BIOS Translation Factor	**divide by 8**	**multiply by 8**	--	--
BIOS Translated Geometry	**767**	**128**	**63**	**2.95 GiB**
BIOS Int 13h Limits	1,024	256	63	7.88 GiB

The BIOS presents the translated geometry to the operating system and application, and as far as basically *every* piece of software in the PC is concerned, the hard disk really has 767 cylinders, 128 heads and 63 sectors. Whenever the operating system or an application wants to use BIOS Int13h calls, they use this geometry. The BIOS, when it executes its disk access routines, translates back to the real logical geometry used by the hard disk before sending its request to the disk. The result is that everyone is happy, and there is a minor amount of extra work for the BIOS to do, but not very much.

Extended CHS or large mode are important to understand, but in practice are not that frequently used. Instead, LBA mode is more popular; it is similar in concept but does the translation differently. It is described in the next section.

Next: Logical Block Addressing (LBA)

Logical Block Addressing (LBA)

Regular addressing of IDE/ATA drives is done by specifying a cylinder, head and sector address where the data that is required resides. Extended CHS addressing adds a translation step that changes the way the geometry appears in order to break the 504 MiB barrier, but the addressing is still done in terms of cylinder, head and sector numbers (they are just translated one or more times before they get to the actual disk itself).

In contrast, *logical block addressing* or *LBA* involves a totally new way of addressing sectors. Instead of referring to a cylinder, head and sector number, each sector is instead assigned a unique "sector number". In essence, the sectors are numbered 0, 1, 2, etc. up to (N-1), where N is the number of sectors on the disk. An analogy would be as follows. Your address (assuming you live in the U.S. and have a regular address) is composed of a street number, street name, city name and state name. This is similar to how conventional CHS addressing works. Instead however, let's say that every house in the U.S. were given a unique identifying number. This would be more how LBA works.

In order for LBA to work, it must be supported by the BIOS and operating system, but since it is also a new way of talking to the hard disk, the disk must support it as well. All newer hard disks do in fact support LBA, and when autodetected by a BIOS supporting LBA, will be set up to use that mode.

A drive using LBA is not subject to the 504 MiB disk size barrier, however there has been a great deal of confusion regarding LBA and what it does. In particular, a lot of people think that it is the LBA addressing that "gets around the 504 MiB barrier". Strictly speaking, this is inaccurate. It isn't the LBA that is getting around the barrier, because LBA is just a different way of addressing the same geometry; if you were still limited to 1,024 cylinders, 16 heads and 63 sectors, you would still have logical sectors numbered 0, 1, 2, etc. up to 1,032,191, and you would still be stuck with 504 MiB.

The reason that setting a drive's mode to LBA will get around the 504 MiB barrier is that in virtually every case, LBA mode automatically enables geometry translation as well. This translation is still required because the software calling the BIOS Int 13h routines knows nothing about LBA. It is the translation that is what really gets around the barrier, but of course all of this happens transparently to the user.

When LBA is turned on, the BIOS will enable geometry translation. This translation may be done in the same way that it is done in Extended CHS or large mode, or it may be done using a different algorithm called *LBA-assist translation*. The translated geometry is still what is presented to the operating system for use in Int 13h calls. The difference between LBA and ECHS is that when using ECHS the BIOS translates the parameters used by these calls from the translated geometry to the drive's logical geometry. With LBA, it translates from the translated geometry directly into a logical block (sector) number.

LBA has in recent years become the dominant form of hard disk addressing. Since the 8.4 GB limit of the Int13h interface was reached, it became impossible to express the geometry of large hard disks using cylinder, head and sector numbers, translated or not, while remaining below the Int13h limits of 1,024 cylinders, 256 heads and 63 sectors. Therefore, modern drives are no longer specified in terms of classical geometry, but rather in terms of their total number of user data sectors and addressed using LBA. See here for more on this.

Next: Comparison of Translation Modes

Comparison of Translation Modes

The table below shows how the three translation modes compare. This is done by showing a summary of the different modes used in the overall path from the operating system, through the BIOS, to the hard disk controller, to the physical drive platters:

Interface	**Standard CHS**	**Extended CHS (ECHS) / Large**	**Logical Block Addressing**
Physical Drive Platters to Integrated Disk Controller	Physical Geometry	Physical Geometry	Physical Geometry
Integrated Disk Controller to BIOS	Logical Geometry	Logical Geometry	Logical Block Address
BIOS to Operating System and Applications (through Int 13h)	Logical Geometry	Translated Geometry	Translated Geometry

Notice how in all three schemes the physical geometry is hidden within the hard disk itself. In the case of LBA, the logical geometry is presented to the BIOS only so it can be translated for use by the operating system; communication between the BIOS and the hard disk controller occur using the logical block address.

For modern drives that use Int13h extensions, only logical block addressing is used.

Next: Caveats on Changing Translation Modes and Transferring Hard Disks Between PCs

Caveats on Changing Translation Modes and Transferring Hard Disks Between PCs

It is important to recognize that even though both enhanced CHS and LBA modes involve the use of translated geometry, this does not mean that it is guaranteed that they will work the same way. In particular, LBA translation may result in a different set of geometries than ECHS translation. Similarly, different BIOSes can in theory use different means of translating. Moving a hard disk that was formatted on one PC to another will *usually* work OK, but sometimes may not. The chances of a problem increase as the difference in age between the two systems increases.

Warning: If you change the translation mode of your hard disk, you risk permanent loss of all the data on the drive. Setting the translation mode should normally be done only once, when the disk is set up. If you later discover that you set up originally using the wrong mode, make sure your data is backed up before changing the mode. In many cases there will be no problem, but better safe than sorry. This also applies to removing a dynamic drive overlay (software translation) from a drive. Only attempt to remove a DDO using the appropriate utility provided by the drive overlay company for this purpose, and only after doing a full backup of the drive's contents.

Next: Overcoming BIOS Disk Size Barriers

Overcoming BIOS Disk Size Barriers

As hard disk capacity has reached and then exceeded the various hard disk size barriers, various methods have come about to allow these larger disks to be used. These solutions include various hardware and software techniques, primarily aimed at getting around limitations imposed by the system BIOS. This section looks at these in detail.

It's important to realize that some of these solutions are simpler and more elegant than others; some are based on *fixing* the BIOS problem, while others are oriented more towards *working around it*. The simpler the solution, in general, the better. More complex solutions, especially ones involving software drivers, tend to have a higher incidence of incompatibilities and other issues.

Note: One solution to overcoming BIOS size barriers that I don't discuss in detail here is to upgrade an older motherboard to a new one. Needless to say, this is not a cheap or simple solution, and not one that I would generally recommend solely to get around a size barrier problem. Still, it's an option I wanted to mention, if but briefly. :^)

Next: BIOS Upgrades

BIOS Upgrades

The most commonly used and important method of getting around the key 504 MiB barrier is through the use of an *enhanced BIOS* that supports BIOS translation. This translation allows the BIOS to break the 504 MiB barrier by translating between disk parameters that the disk understands, and a different set that the BIOS understands. The main problem with BIOS translation is that very old BIOSes don't support it. In some cases, a BIOS upgrade can enable an older system to employ BIOS translation, and a system that supports flash BIOS upgrades can be updated to this support without even opening up the system case. The manufacturer of the system or motherboard is the place to start when considering a BIOS upgrade.

Similarly, since many of the later size barriers are due to limitations in BIOS code, they can similarly be overcome by upgrading the BIOS with new code that can handle larger drives. A new BIOS today will enable Int13h extensions to get around the 8 GB size barrier, for example. It will also address other

BIOS code problems such as the one that causes the so-called 32 GB size barrier.

A BIOS upgrade from the system or motherboard manufacturer is generally the best solution to a hard disk size barrier problem, for two reasons. First, it is generally free, if the upgrade has been provided by the original hardware maker. Second, it is the simplest and most direct solution to the problem: you are replacing broken code with fixed code, and once that is done your hard drives will work properly without any other work being required. These two attributes make this solution far superior to the others I discuss in this section, and I strongly advise that you look for a BIOS upgrade before you consider the others.

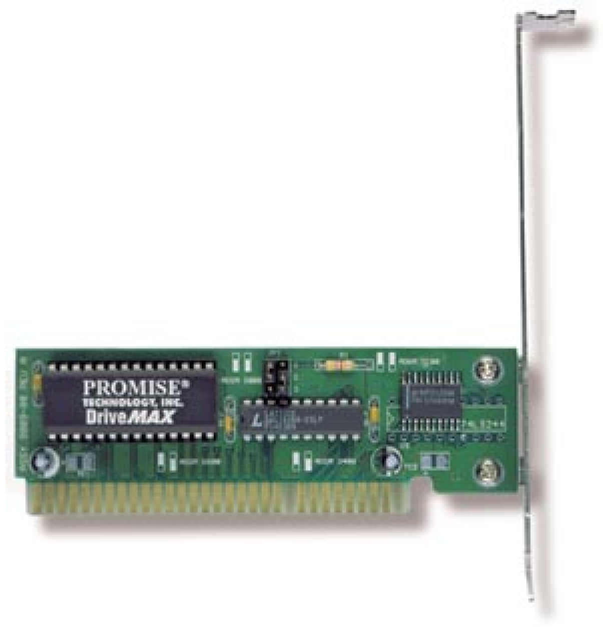
Unfortunately, some BIOSes cannot be easily upgraded. This is usually because the manufacturer has obsoleted the motherboard and has decided to no longer support it. (This is understandable for very old machines, but occasionally manufacturers give up on hardware as little as two years old, which I consider unacceptable--avoid such hardware.) In some cases where this has happened, you may be able to purchase a *third-party* BIOS upgrade. This is BIOS code that has been written by a company other than the one that initially made your system or motherboard. This is a viable option, and retains the advantage of being an elegant solution, but not the one about the upgrade being free! In some cases these upgrades are $75 or more, and frankly, if all you want is updated hard disk size support it's often not worth the money compared to other solutions. If you want other features that can come with a BIOS upgrade then it may be worthwhile.

Next: Enhanced BIOS Expansion Cards

Enhanced BIOS Expansion Cards

One possible solution to a system with a size barrier that cannot have its BIOS natively upgraded, is the use of an *enhanced BIOS expansion card*. In essence, this is a "BIOS upgrade on a card". It goes into a system expansion slot and contains nothing more than new BIOS code for controlling your motherboard's IDE/ATA controllers. The new BIOS code takes over for the hard disk controller code of your system's BIOS, and in doing so, gets around hard disk size barriers. You continue to use the IDE/ATA connectors on your existing motherboard or controller card.

These expansion cards are *very* inexpensive--often under $20. They usually use an ISA expansion slot, and most older systems have an extra ISA slot available, so this is a fairly easy upgrade. (Many newer systems have only one ISA slot or even no ISA slots at all, but they usually don't run into size barriers the same way.)



An add-in ISA BIOS expansion card. (The Promise DriveMAX.)

Image © Promise Technology, Inc. Image used with permission.

Despite their low cost, these cards have never seemed to be quite as popular amongst PC enthusiasts as add-in controller cards that include actual IDE/ATA interface ports. This may be because the add-in cards continue to let you use the existing motherboard controllers for other devices, or it may be the ISA slot requirement. BIOS expansion cards were more popular in the 1990s when the earliest hard disk size barriers began to appear, but add-in cards are now more commonly prescribed for BIOS size barrier problems, even though they generally cost more.

Next: Upgraded Controller Cards

Upgraded Controller Cards

Another solution to size barriers for a system that cannot have its BIOS upgraded is the purchase and installation of an upgraded hard disk controller card. Such a card completely replaces the on-board IDE/ATA controller hardware of your system, and in doing so, eliminates any BIOS size restrictions associated with the controller in your system.

These cards differ from BIOS expansion cards in that they are not just BIOS code, but rather a complete controller including the IDE/ATA ports. To use this hardware, you install the card and then move your hard disk cable(s) to attach to the new card instead of the ports on your motherboard. These controller cards are also generally more expensive than straight BIOS expansion cards, but we're still not talking about a lot of money here: generally less than $50.

The Promise Ultra66 PCI IDE/ATA controller card, one of the most popular on the market.

Image © Promise Technology, Inc.
Image used with permission.

Despite being more expensive and more complicated than BIOS expansion cards, these upgraded controller cards are more popular. There are a few issues involved in choosing them however:

- **System Bus Matching:** You need to make sure you get a card that works with the system bus type that you are using. Today, PCI controller cards are pretty much the only ones available. If your system uses the VESA local bus (VLB) you may be able to find a controller that will work with that older bus, but to be honest, any system that old is probably getting to the point where spending money upgrading it *may* be a bad use of funds. If your system only supports ISA then you should really be looking at a new system! The ISA bus is so slow and such systems are so ancient that there's generally no point in even trying to upgrade them.

- **Controller Conflicts:** If you use an add-in card to replace an older hard disk controller card (as is the case with most VLB systems) you just need to take the old card out and replace it with the new one, making sure it is configured properly. If you are putting the card into a PCI-based system that has the hard disk controller built into the motherboard, you should disable the controller in the BIOS setup program or the two controllers may conflict and your system will not work. (Some systems will stubbornly refuse to disable the controller; if your system is like this then this solution is not for you.) Many newer systems that support PnP will let an add-in controller work with its existing controller, but there are other issues. You will spend an IRQ on the new card, and you may not be able to control whether the drives on the new card are "seen" by the system before or after the ones on the existing controller.
- **Compatibility Issues:** These are atypical but there have been occasional problems caused by replacing the existing controller with an add-in card. Mostly these add-in controllers work rather well, though.

Next: Software Translation Drivers (Dynamic Drive Overlays)

Software Translation Drivers (Dynamic Drive Overlays)

I always suggest that those who are trying to overcome a hard disk size barrier, first attempt to correct the problem through a free BIOS upgrade, if possible. If that's not an option for whatever reason, however, you will have a key decision to make in choosing between the other alternatives I am presenting: do you want to spend money on a hardware solution, or go with a "free" software solution?

For those who can afford to do so, I strongly recommend a hardware solution, such as an expansion BIOS card, an add-in controller card, or a third-party (not free) BIOS upgrade. The reason is simple: these solutions get around the hard disk barrier at a very low level within the system, and in doing so ensure that you will have few problems using your new hard disk on your old system. There are not likely to be any software issues with such a solution.

If you cannot spend funds on a hardware solution (or simply don't want to for whatever reason) then your alternative to allow access to the full capacity of your hard disk is the use of a *software translation driver*, also called a *dynamic drive overlay* or *DDO*. These usually go by names like Disk Manager, EZ-Drive and the like. The idea behind one of these programs is pretty simple: they override in software some of the BIOS code in your motherboard or hard disk controller, allowing access to the full size of a new hard disk on an older system. The software must be loaded immediately when the machine is booted, to ensure that the driver is in place before any other piece of software tries to access the disk. Otherwise, the disk will not work properly.

To ensure that they are always loaded immediately at boot time, the installer for this sort of program modifies the boot disk's master boot record and installs the driver at the beginning of the disk.

When you buy a new hard disk at retail, the drive manufacturer will often include a copy of one of these driver programs, "free", with the drive. (You can often download them for free from the drive maker's web site too.) These are normally a specially modified version of something like *Ontrack's Disk Manager* that is customized for that manufacturer's drives; these utilities normally have proprietary names as part of the licensing agreement between the drive maker and the company that writes the overlay. Drive manufacturers provide these as a convenience for those whose machines don't have real hardware BIOS support for larger disks, and using them is a viable option.

However, do *not* believe these manufacturers when they sometimes say that using these software drivers is as good as proper BIOS support. It isn't. There are numerous problems associated with using these drivers for large disk support, which is why I do not recommend their use. Here are just a few:

- **Compatibility Problems:** When you use one of these drivers they essentially set up their own logical disk volumes using a non-standard format. This means you are not using your disks the standard way. This isn't usually a problem in and of itself, since most operating systems know about these drivers, but the potential for incompatibility exists.
- **Reduced Drive Interoperability:** The drivers that come with the various manufacturers' drives are normally customized for that manufacturer's equipment only. This means that if you put a Quantum disk in your PC and later want to add a Seagate, for example, you may have a bit of a problem. You will have to at this point probably purchase the full version of something like Disk Manager, and for the extra cost you will be better off buying an add-in hard disk controller.
- **Problems Removing the Driver:** Some of these overlays can be very difficult to remove from the disk, and require you to use uninstall facilities that come with the driver, if you want to get rid of them. When you do remove the driver, say because you have upgraded to a PC that supports large drives, you may have to repartition and reformat the disks (though this may not be required).
- **Floppy Disk Booting Complications:** Because the driver is located on the hard disk, you *must* boot from the hard disk to load it. If you boot from a floppy, your hard disk may seem to "disappear" because the overlay wasn't loaded. The driver will allow you to boot from a floppy, but you must do it by booting the hard disk, waiting for the overlay to load and a message to be displayed that says "To boot from a floppy disk, press the space bar", and then put the floppy into the drive and press the space bar.
- **Operating System Installation Issues:** The drive overlay located on the hard disk can cause problems when using alternative operating systems, attempting to set up a multiple-OS system, and so on. You must verify that every operating system that you install is capable of handling the driver you are using.

Again, most of the time these drivers will work OK, especially if you are not doing anything too unusual with your PC. With so many people upgrading older systems and running into size barriers, they are becoming more prevalent than ever. I just think that they are not the best way to deal with BIOS translation, given that much more reliable, and relatively inexpensive, hardware solutions exist. It's all a matter of your priorities, really.

Next: Disk Size Reduction Jumpers

Disk Size Reduction Jumpers

A rather sub-optimal solution that is offered as an alternative by some hard disk manufacturers is the *disk size reduction jumper*. Certain hard disk size barriers can cause a hard drive to not be seen by a system at all. If you set one of these jumpers on the hard disk, this tells the disk to change the drive parameters it presents to the system, reducing the size of the drive. Since the system sees the drive as being small enough to avoid the size barrier, the barrier is avoided. Of course, this costs you capacity: you lose the storage of the drive above the size barrier, unless you supplement the jumper with a software overlay.

The first time size reduction jumpers were commonly used was to get around the 4,096 cylinder barrier. One of the common ways that a BIOS with this barrier may treat a disk that is over about 2.1 GB is wrapping around. So a 2.5 GB disk would be seen as only around 400 MB. The size reduction jumpers would cause the hard disk to "pretend" that it had less than 4,096 cylinders, so it was only 2.1 GB in size. This is of course a waste of however much space the disk really has over 2.1 GB, but it is better than only using 400 MB of the disk.

Similarly, some newer drives may come with size reduction or capacity limiting jumpers to get around the 32 GB size barrier. These force the drive to present a size small enough to avoid triggering that capacity barrier.

This "solution" is in some ways the hard disk equivalent of that old Marx Brothers routine: "Doctor, it hurts when I do this"... "So don't do that!" ;^) It's not really a solution, but rather a way to avoid the consequences of the problem. It's better than having your PC hang when you try to boot the system, but still, this solution is a very poor one. The only time these reduction jumpers really make sense is when you use them *in conjunction* with drive overlay software--in some cases, without the jumpers you may not even be able to get the system booteed so you can install the overlay! Proper hardware support is still a better solution of course.

Next: Manually Entering Drive Geometry Parameters

Manually Entering Drive Geometry Parameters

Some drive problems associated with older size barriers and drives that are small by today's standards, can be avoided simply by avoiding the use of

BIOS hard disk autodetection and entering drive parameters manually. Yes, I know, I am always saying to use autodetection. I do think it is the best way for most people to set up their drives--assuming their BIOS is new enough and supports the drives being used.

In some cases, for example, where the BIOS wraps around when the disk is too large, you can get around the problem by manually entering the maximum parameters that your disk can support. Let's suppose that you have a 540 MB hard disk that your old 486 system is choking on. The drive has 1048 (logical) cylinders, 16 heads and 63 sectors per track. If you manually set the disk up in your BIOS as having 1023 cylinders, 16 heads and 63 sectors per track, it will probably work fine, but as a 528 MB disk. You still get the use of 97% of the disk.

Note that this technique will be of no help for modern systems with large drives; for example, it will do nothing to get around larger size barriers such as the 8 GB barrier.

Next: Int 13h Extensions

Int 13h Extensions

The only solution to the 8GB hard disk size barrier is to do away with using the standard Int13h disk access routines. This new method of access requires changes to the BIOS to support what are called *Int 13h Extensions*. These extensions are discussed in general terms here.

Using Int13h extensions in fact requires changes to everything that is associated with accessing the disk: the disk itself, the BIOS, and the operating system. Newer operating systems all support this important change to how hard disks are addressed, including all versions of Windows from Windows 95 on. Modern drives also support this method of being addressed. This means that getting past the 8 GB hard disk barrier is generally a matter of ensuring that the system BIOS or hard disk controller supports Int13h extensions.

Tip: Some hard disk manufacturers make available drive utilities that can test your system BIOS and determine if it will natively support Int13h extensions. Check the download section of the drive manufacturer you are considering.

Another implication of going away from Int13h is that the tired old "geometry method" of specifying hard disk size finally must be put to rest. In fact, due to multiple levels of translation both within the BIOS and the hard disk itself, the logical geometry parameters of IDE/ATA hard disks stopped having any real relationship to the actual drive characteristics long ago. However with drives now above 8.4 GB in size, they cannot even be expressed using traditional geometry terms, without going over the cylinder, head and sector limits mentioned above. Instead, the total number of sectors on the drive is now the key parameter, and the drive is accessed using logical block addressing. For example, a 45 GB IBM Desktop 75GXP drive would, if expressed in conventional geometry, have to be said to have 89,355 cylinders, 16 heads and 63 sectors. Instead, it is just said to have 90,069,840 data sectors. By convention, and for compatibility, all drives over 8.4 GB have logical geometry parameters of 16,383 cylinders, 16 heads and 63 sectors, which is why these drives show up as being about 8.4 GB in size if Int13h extensions have not been implemented.

Next: Hard Disk Interfaces and Configuration

Hard Disk Interfaces and Configuration

The interface that the hard disk uses to connect to the rest of the PC is in some ways as important as the characteristics of the hard disk itself. The interface is the communication channel over which all the data flows that is read from or written to the hard disk. The interface can be a major limiting factor in system performance. The choice of interface also has an essential impact on system configuration, compatibility, upgradability and other factors.

Over time, several different standards have evolved to control how hard disks are connected to the other major system components used in the PC. These have tended to build upon one another, and often use confusing and overlapping terminology. The result has been a great deal of confusion surrounding the entire subject. Each time a new variant or enhancement of an interface is introduced, the interface becomes just a bit more confusing, particularly for those trying to use older hardware, or to mix newer and older devices.

To help you understand what can be a baffling subject, this section of the site takes a comprehensive look at the different interfaces used to connect hard disks to the PC. I begin by discussing two obsolete interfaces no longer used, and also provide brief coverage of some "alternative" interfaces that are not commonly employed by typical PC users, but are important for special applications. Most of the focus is on the two interfaces most often used on the hard disk. I discuss in detail IDE/ATA and its enhancements, with a focus on clarifying the confusion that surrounds the use of this most popular PC interface. I then cover SCSI, the more advanced and flexible interface that dominates the business workstation and server world, and is becoming the choice of a growing number of performance-oriented desktop PC users.

Note: This part of the site is in the discussion of hard disks, and so they will be my primary focus. However, many other devices use the same interfaces that hard disks do; where appropriate, distinctions between how hard disks and other devices use the interfaces will be specified. Otherwise, you can assume that using the interface for optical drives and similar storage devices will be similar to how hard disks use the interface.

Next: Hard Disk General Interface Factors

Hard Disk General Interface Factors

There are a number of attributes that determine the quality and usefulness of a hard disk interface. One can theoretically evaluate a hard disk interface by looking at how well it "scores" in these various attributes, and comparing it to alternative interfaces. In the real world, of course, we are limited in our choices to what the industry uses as a standard; it's not like one can easily design a new interface. Still, it's useful to understand these general interface factors, and in particular, how they trade off against each other--as with most areas of life, you "can't have everything". :^)

In this section I begin by looking at compatibility issues related to the system bus, and then cover the very important matter of interface performance, and a related issue, command overhead. Then I examine issues of device support and expandability. Last, but definitely not least, is the essential matter of cost.

Next: System Bus Interface

System Bus Interface

Every hard disk interface communicates with the PC over one of the system's I/O buses. On modern systems, the main system bus is PCI, with some support for the old ISA standard still provided. Older systems used the VESA local bus (VLB) in combination with ISA. Logically, the hard disk interface is one device on the system I/O bus, which is connected to the memory, processor and other parts of the system.

The choice of bus type has a great impact on the features and performance of the interface. Higher-performance interfaces, including the faster transfer modes of both IDE/ATA and SCSI, require an interface over a high-speed local bus. On modern systems, this means PCI. Older systems that use VLB for the hard disk interface still offer acceptable performance. Extremely old systems that still use ISA for the hard disk controller will be severely limited in performance.

The speed of the system bus is also important; the faster the bus, the faster the interface (simplistically speaking). The standard speed for the PCI bus on Pentium-class systems is 33 MHz, which is fast enough for current hard disks but will need to be replaced within a few years as hard disk performance continues to increase. Some disk subsystems are even now moving to higher-performance versions of PCI, as the limits of conventional PCI are reached. Some older systems used slower versions of the PCI bus, running at 25 or 30 MHz, which wasn't an issue for hard disks of that era.

Most older machines use a dedicated hard disk interface card that goes into a system bus slot and then connects to the drives internally. For the last several years, however, all newer PCs have had the ports for two IDE/ATA channels built into the motherboard itself. In practical terms, there is no real difference except for the cost savings associated with not needing to put a separate hard disk interface card into the PC when it is built into the motherboard. In fact, many people add PCI after-market controller cards to their systems, to enable access to a different interface (such as SCSI) or features not supported at the time their systems were originally designed. For example, inexpensive PCI cards are available to provide support for Ultra DMA modes for newer IDE/ATA drives, which can augment or replace the disk controller built into the motherboard.

Next: Interface Performance

Interface Performance

Obviously, the interface's job is to allow for the expedient transfer of data between the hard disk and the system. Since the hard disk is an important performance component in the PC, the performance of the hard disk interface is probably the most frequently examined aspect of an interface's overall quality. In fact, it is generally given far too *much* emphasis in my opinion, particularly by companies looking to sell hardware based on the speed of the interface.

I'm certainly not suggesting that the performance of the interface is not important. However, it's important to keep it in perspective. The main problem when it comes to looking at interface performance is that on modern systems, the speed of the interface is not the limiting factor to overall hard disk performance. If a given hard disk can't read data from its platters fast enough to saturate an interface of a given speed, going to a faster interface yields improvement only on reads of data already in the drive's internal buffer--which makes virtually no difference in overall, real-world performance. Despite this, companies try to claim that (for example) drives using Ultra DMA/100 are "50% faster" than those using Ultra DMA/66. In fact, at the time of this writing (late 2000) no IDE/ATA drive can saturate even a 66 MB/s interface, so going to 100 MB/s is essentially pointless. For a more complete discussion of this issue, see this page on the interface transfer rate specification.

Actually, I have devoted an entire section of the site to discussing hard disk performance, so you should look there for a lot more detail on this important subject. Of particular interest will be the page I referenced above on the interface transfer rate specification, as well as the general discussion of interface performance factors.

Next: Command Overhead

Command Overhead

All interface communication is done through command and data transfer; the host (the PC, through the interface controller) sends commands over the interface to the disk, and the disk sends data over the interface back to the host. Some interfaces are simpler to use than others and therefore require less command "talk" over the interface. This can lead to improved responsiveness in simple setups, such as when there is only one disk on the interface. The amount of time that the interface, its controller, and the hard disk, require for processing commands is referred to as *command overhead*.

This is one factor that influences overall interface performance. For more information on command overhead and its performance implications, see this page in the discussion of hard disk performance.

Next: Device Type Support and Software Compatibility

Device Type Support and Software Compatibility

Not all interfaces support the same devices. Depending on what devices and models you are trying to use, one interface may make much more sense than another. Fortunately, the market has evolved so that there are really only two main interfaces used today for hard disks: IDE/ATA and its variants, and SCSI and its variants. The fact that there are only two standards in common use means that each of them supports a large number and variety of devices. In addition, this makes it easier for software support to be made universal, and this is no longer much of an issue when using these mainstream interfaces.

IDE/ATA tends in general to support a larger number of hard disk models, and also optical drives and other devices, especially economy models. This is not due to any particular technical advantages that IDE/ATA possesses over SCSI. It's simply a function of IDE/ATA being more popular than SCSI, and therefore offering manufacturers more of a target market than SCSI does. SCSI tends to have better support for high-end devices and also more device types.

All of this means that the casual home user is more likely to be interested in IDE/ATA, while SCSI is usually the choice for businesses and performance hounds. See this comparison of the two interfaces for more. In addition, some of the specialty interfaces fill a particular "niche" by offering connection options and device types not found in the "big two" interfaces. For example, using USB or a PC Card adapter, you can easily connect an external hard disk to your notebook PC.

Next: Multiple Devices and Expandability

Multiple Devices and Expandability

All interfaces are limited in terms of the number of devices they can support, but some are more limited than others. This is rarely perceived to be an important issue by most new PC users, but after a couple of years of accumulating devices it can rather suddenly grab your attention. :^) For example, the standard IDE/ATA controllers included in most systems will support a total of four devices. Most PCs come with a single hard disk and one optical drive, leaving room for two more devices. No problem right? Not at first. :^) However, you may decide to add to this a Zip drive and then perhaps a CD-RW unit. Then, two years down the road, you might realize you want to add a new hard disk... This happens to a lot of people who never expected they would ever need more than four IDE/ATA devices. :^)

Expanding a system to allow for more IDE/ATA devices can be done; in many cases it is fairly simple to do, but in others it is far more involved. In contrast, some interfaces are relatively easy to expand. For example, once one has a SCSI host adapter and a SCSI bus running, adding a new device to the existing SCSI bus is fairly trivial, and support is provided for 7 or 15 different devices in a single chain.

Easy expansion is also part of the appeal of specialty interfaces like USB: you simply plug the device in and load a driver, and off you go! This makes it very flexible for a variety of uses. For systems like notebooks, USB, PCMCIA or the parallel interface may be selected for additional hard disks or similar devices primarily because of the limited expansion options of the IDE/ATA implementations on such systems.

Next: Cost

Cost

A final consideration when evaluating different interfaces is the *cost* of the interface. In fact, I should really says *costs*, because there are several different aspects involved in assessing the monetary costs associated with an interface. While this is a non-technical consideration, cost considerations are actually probably the most often used by those making an interface choice, not technical ones.

The most important cost consideration is the hardware and software cost for implementing the interface. What this often boils down to is whether or not support for the interface is already included in a given system. For example, the presence of IDE/ATA controllers on all modern motherboards makes this interface less expensive for most people than going with SCSI, which would require the addition of a SCSI host adapter. Similarly, USB has been around for years, but is only becoming popular now that almost all new systems have built-in support for it, both hardware and software. Some interfaces also require more expensive support hardware than others--such as specialized cabling, termination, external enclosures and so on.

Another cost consideration is the cost of the actual devices that are used on the interface, as not all are equal. Generally speaking, this matter is related to the cost of the interface hardware as well--interfaces that are more expensive to implement generally have more expensive hardware sold with them, while inexpensive interfaces have cheaper storage devices. This results naturally from market forces: few people will want to pay $200 to implement a particular interface to enable use of a $50 device.

When considering the costs associated with an interface, be sure to look at *all* of the costs, including some that may be hidden. For example, the SCSI interface may require an up-front investment, but once set up offers expandability and other advantages that may offset its cost. Of course, what is a worthwhile expenditure depends primarily on the needs of the individual.

Next: Obsolete Hard Disk Interfaces

Obsolete Hard Disk Interfaces

There are two hard disk interfaces that were used in the early days of the PC, in the 1980s. These are still found on any systems of this "bygone era" that are still in operation; this may surprise you, but there are in fact many thousands of 15-year-old hard disks still chugging along! :^) However, they are no longer used in modern systems, nor are new drives made that use these interfaces.

What makes an interface "obsolete"? Essentially, when something new comes along that is sufficiently superior that it pushes the old standard out of the market completely. Just being "newer" or "better" isn't really enough: it has to be so superior that the market sees no reason to keep using the older standard. Both of the interfaces described here were made obsolete by IDE/ATA and SCSI, which offered significant advantages over them without imposing any real cost. (It's actually fairly uncommon in the PC world for a standard to be made obsolete by another, because in most cases the older standard still has adherents due to some particular advantage it offers.)

Since these interfaces are of little or no interest to 99% of current PC users, their descriptions will be brief.

Next: ST-506 / ST-412 Interface

ST-506 / ST-412 Interface

The original hard disk interface used in the PC world was developed in 1980 by Seagate Technologies, to work with that company's 5 MB ST-506 hard disk. It was later revised to support the 10 MB ST-412, which was the first hard disk drive model used in the IBM PC/XT. You can obviously see where the name of the interface comes from, given this bit of history! In common parlance, this older interface is sometimes called "MFM" or "RLL", so you may hear someone refer to a drive using this interface as an "RLL drive" for example. These actually refer to the encoding method used for storing data on the disk. Both encoding methods were used on these early drives. (In fact, RLL is used on some IDE and SCSI drives also, making it a poor name for referring to drives of one particular interface.)

The ST-506/ST-412 interface differs from the IDE/ATA and SCSI standards in one very important respect: the hard disks were "dumb", meaning there was no built in logic board as modern drives have. All of the smarts resided in the controller card that plugged into the PC. This caused a host of problems relating to compatibility, data integrity and speed, because the raw data from the read/write heads was traveling over a cable between the controller and the drive. This interface also required a lot more work on the part of the user, because while a newer drive ships with an integrated controller card built into the drive that is optimized for that drive, these older ones didn't have this, and therefore the person setting up the drive had to program the interleave ratios and other factors into the drive to achieve maximum performance.

By today's standards, this interface and the drives that use it are microscopic in capacity (although enormous in physical size), slow, cumbersome, error-prone and completely obsolete. You will never see ST-506/ST-412 used in a new system, and in fact, it's hard to find them in *any* systems still being used, unless you look around or know a very frugal person. :^) You can recognize this interface in older systems by the use of two ribbon cables (instead of the single cable used by IDE/ATA and SCSI). One of the cables is 20 pins wide and carries data, and the other is 34 pins and carries control signals.

Next: Enhanced Small Device Interface (ESDI)

Enhanced Small Device Interface (ESDI)

The first attempt at improving the original ST-506/ST-412 hard disk interface was the *Enhanced Small Device Interface* or *ESDI*. ESDI was developed in the mid-1980s by a consortium of hard disk manufacturers led by Maxtor. It was eventually codified as an ANSI standard; the peak of its popularity was in the late 1980s.

ESDI improved on ST-506/ST-412 in several ways. It moved some drive controller functions to the hard disk from the controller card, eliminating some of the reliability problems associated with its predecessor. It had a maximum theoretical bandwidth of 24 Mbits/second (fairly fast for those days), though in practice the limit was about half of that. There were other added features and small performance enhancements as well. Its primary design still had almost all of the intelligence on the controller and not on the hard disk.

While ESDI was a real improvement over the older ST-506/ST-412 interface, it was "too little, too late" in a lot of ways. In the late 1980s ESDI suffered under competition from IDE/ATA in the mainstream market and from SCSI in the high-end market, both of which offered significant advantages over ESDI, such as simpler configuration, lower cost and improved performance. As a result, by the early 1990s ESDI had been all but wiped off the interface map.

Next: Specialty and Future Hard Disk Interfaces

Specialty and Future Hard Disk Interfaces

While IDE/ATA and SCSI are used for the vast majority of hard disks in the PC world, there are some other interfaces that are used on occasion, or even used heavily in certain segments of the market. These are typically employed to fill a particular need where IDE/ATA and SCSI don't fit well. In particular, many of these interfaces allow for easy, inexpensive connection of external devices to PCs, especially ones that have limited expansion options, such as notebooks.

In this section I will first discuss some of the less common interfaces that are used for special applications or purposes in the hard disk world. This includes coverage of the parallel port interface, PCMCIA (PC Card) options, USB, IEEE-1394 (Firewire) and Fiber Channel. I will then also discuss two upcoming interfaces that may significantly change the landscape of the hard disk interface world: USB 2.0 and Serial ATA.

Note: Most of the interfaces described in this section are used for connecting a wide variety of devices to the PC, and many are not even generally associated with storage at all. Consistent with the goals of this part of the site, the discussion of these interfaces here will be limited to an overview, and focused primarily on each interface's role in the interfacing of hard disks and other storage devices. Complete discussion of various PC interfaces in a general way will be provided in a future expansion of The PC Guide.

Next: Parallel Port Interface

Parallel Port Interface

Originally designed only for printers, the parallel port turned out to be an amazingly flexible and universal interface for attaching a host of different devices to a variety of PCs. While rather slow, the parallel port has the advantages of being (mostly) standard, and present on virtually every PC. For this reason, it is often used for attaching external devices such as hard disks, CD-ROM drives and tape drives, especially for PCs that lack other expansion alternatives such as notebooks. For many years, I personally used a parallel-port Iomega Zip drive when I was maintaining several dozen PCs that were not networked; it was very helpful for transferring files between machines and doing backups. The parallel port can also be used with special software to connect two PCs together to let them share files.

Today, the role of the parallel port for storage devices is diminishing rapidly, as its "niche" is increasingly being filled by other interfaces that offer advantages over it. Almost all notebook PCs, for example, now come with PC Card slots and USB ports, which offer plug-and-play operation and better performance than the parallel port. The rise of networking has also largely obviated the need to use the parallel port for file sharing or connecting PCs together. I do expect use of the parallel port interface to "hang around" for many, many years, for the same reason floppy disk drives continue to be used: they are universal and there are a lot of devices around that make use of the parallel interface. As long as printers continue to use the parallel port as a primary means of attachment, other devices will be available for the parallel port as well.

Next: PCMCIA (PC Card) and CardBus

PCMCIA (PC Card) and CardBus

The first laptop PCs had very few expansion options. They typically had serial and parallel ports, and some expensive units had add-on docking stations (port replicators) but these are quite limited in their capabilities. In 1989, a group of PC system and computer manufacturers created the *Personal Computer Memory Card International Association* or *PCMCIA*. The goal of this group was to define a standard for expansion devices--called *PC Cards*--to be used in special slots in mobile computing equipment. The group of course succeeded--probably beyond even their own expectations! Virtually all mobile PCs made in the last five years have the ability to accept these credit-card-sized PC Cards (often called "PCMCIA cards" after the name of the trade organization, even though they don't like it). In addition, this technology is also used in other types of electronic gear, outside of the PC world.

As the name "PCMCIA" suggests, one of the original main applications of PC Card technology was memory expansion. Ironically, today many laptops don't even need expansion slots for this purpose: they have separate user-upgradeable memory slots! The new PC Card slots however were quickly embraced, and are now used for many other purposes. Due to the explosion in mobile computing, manufacturers of devices of all sorts moved quickly to exploit this new interface--in addition to memory cards, you can get modems, network interface cards, sound cards, and other devices that fit into PC Card slots. Of course, this includes a rather wide variety of storage devices, or I wouldn't be bothering to tell you all of this. :^)

There are actually two different interfaces that are used with PC Cards today. The first is the older, "original" PC Card or PCMCIA interface, which is a 16-bit interface that runs at 8 MHz. If this sounds a bit like the regular ISA system bus found on desktop PCs, there's a good reason for that: the traditional PC Card interface is based on ISA. Of course the form factor and device packaging is different, but in terms of operation, they are similar. This means that conventional 16-bit PCMCIA cards have similar performance constraints to ISA: they are perfectly fine for modems and similar slow devices, but inadequate for performance-intensive applications such as networking--or storage devices.

To address the limitations of conventional PC Cards, in 1995 the PCMCIA announced a new interface called *CardBus*, which is a 32-bit interface running at 33 MHz. Say... doesn't that sound a bit like good old PCI? Bingo, you win again. :^) In some ways, CardBus is to PC Card what PCI is to ISA. Cards using this newer interface maintain the same physical packaging as older cards--they just run faster. Most newer notebook PCs have PC Card slots that will automatically support both 32-bit CardBus devices and also the older 16-bit PC Cards. CardBus support is essential for high performance devices such as modern hard disks or high-speed Ethernet.

There are in fact several ways that hard disks can be interfaced to a system using the PC Card or CardBus interfaces:

- **PC Card Hard Disks:** Several companies make tiny hard disks that actually fit within the very small physical dimensions of PC Cards. For more on these PC Card form factor hard disks, see this page. Note that few notebooks use these as primary hard disks, because they are relatively small and rather expensive; the 2.5" form factor is standard for main hard disks on modern notebook systems.
- **Proprietary PC Card Storage Devices:** Many manufacturers make storage devices that connect to notebook systems directly through a proprietary PC Card interface. For example, a tape backup unit may come with a special adapter card in the form of a PC Card to connect it to a notebook.
- **PC Card "Interface Converter" Cards:** PC Cards are made that "convert" the PC Card or CardBus interface to either IDE/ATA or SCSI. These let notebook PCs connect regular IDE/ATA or SCSI storage devices (though providing power for them may be an issue.)

PC Cards continue to be a force in the growing mobile computing market, and I don't expect them to go away any time soon. While PC Card slots are standard equipment on notebooks, you can add support for them to desktop PCs, using hardware kits that include an interface card and slots that mount in a regular drive bay. While of little attraction to most PC users, these can be helpful to some people who want to share devices between a desktop PC and a notebook.

For more information on PC Cards and the PCMCIA, see http://www.PCMCIA.org.

Next: Universal Serial Bus (USB)

Universal Serial Bus (USB)

The conventional serial ports used on PCs have been around since the earliest PCs; they have changed a little, but not much. While functional, they have serious limitations in terms of expandability, software support and performance. In the mid-1990s, a consortium of PC and telecommunications industry giants--including Compaq, IBM, Intel, Microsoft, NEC and Nortel Networks (then Northern Telecom) developed a new interface standard for attaching relatively low-speed devices to the PC. This interface is called the *Universal Serial Bus* or *USB*. While USB support started being supported in many PCs many years ago, it has only recently started to gain widespread acceptance. All new PCs are now equipped with USB support, and USB ports can also be added to many older systems as well.

The USB interface is specifically designed to allow easy connection of a wide variety of devices; it is intended to be user-friendly and truly "plug and play". On a system equipped with USB, one can "hot swap" devices, meaning they can be plugged into the system or removed without needing to power the system down or doing anything to it before the change is made. Up to 127 devices are supported, and multiple devices can easily be added to a single PC by chaining them together using hubs. The USB connection is of course serial, and the current version (1.1) runs with a maximum throughput of 12 Mbits/second (1.5 Mbytes/second), which is shared by all devices. There is also a slower-speed, 187.5 kbytes/second mode available for very slow devices, such as keyboards.

USB's initial acceptance was relatively slow in coming, but once it got going, the variety of USB devices that appeared on the market surprised many people, probably including its developers. The flexibility and expandability of the interface, and the ease with which devices could be attached, made it very attractive to many users. Many of the devices that were made available in the USB interface were never intended for USB when it was first created. Most storage devices would certainly fall into this category.

Today, one can get a wide variety of storage units that use the USB interface. This includes floppy disk drives, hard disks and optical drives. They work on the USB interface, and certainly there is no issue with floppy disk drives for example. The problem is that the maximum throughput of the interface is 1.5 MB/second--and that's a theoretical maximum, meaning the reality is somewhat lower. This is fine for mice, modems and scanners, but it's very slow for hard disks, considering that modern drives can have throughput level 20 to 30 times above that number. Even something like a 40X CD-ROM drive has a maximum throughput far above 1.5 MB/second. Of course, USB was never envisioned as an interface for high-speed devices like hard disks, so its designers can't be blamed for these limitations.

USB remains popular for hard disks and other storage devices for those who don't have other options; however, in almost every case a higher-speed interface is a better idea if that's at all possible. Otherwise, you will seriously restrict the performance of the device. This is fine for some tasks, such as occasional backup, but far from ideal. Responding to the faster devices that many companies began developing for USB, USB 2.0 was developed, with similar attributes but much faster performance. IEEE-1394 is another high-speed serial interface that competes for the same areas of the market.

Note: For more information on USB, see the *USB Implementers Forum site*.

Next: IEEE-1394 (FireWire, i.Link)

IEEE-1394 (FireWire, i.Link)

IEEE-1394 is the rather bland name for a relatively new, high-speed serial interface that has actually been around in one form or another for many years. In some ways, the *name* is the most confusing aspect of this technology. It was originally developed by Apple, who called it *FireWire*; this name became popular, but Apple owns the rights to it, and many companies refused to pay to license the name. So then everyone started to refer to it by the standard number assigned it by *the IEEE*, which formally published the interface as a standard in 1995. To further muddle the issue, Sony then created its own marketing term for the interface, *i.Link*.

IEEE-1394 is defined part of the SCSI-3 family of related standards, and was at one point sometimes called "serial SCSI". It is, in fact, a type of SCSI, based on the broad converage of SCSI-3, which goes beyond regular SCSI to cover several similar, "SCSI-like" technologies. In terms of signaling and some aspects of operation, IEEE-1394 really can be thought of as "serial SCSI".

In terms of configuration and how it is used in the PC however, IEEE-1394 is better thought of as "USB, only faster". (Or alternately, USB is IEEE-1394, only slower. ;^)) It is a serial interface that supports dozens of daisy-chained devices, hot-swapping, and plug-and-play. However, instead of USB's 12 Mbits/second maximum transfer rate, IEEE-1394 supports up to 400 Mbits/second. Sounds good, right? In fact, it does; when originally introduced, IEEE-1394 had considerable promise, and there were some analysts who thought it would eventually become a major player in the mainstream hard disk interface market. For example, it is not as fast as high-speed implementations of SCSI, but is considerably simpler to implement, and doesn't suffer from the speed limitations of USB.

In reality, though, IEEE-1394 still has not taken off as a storage interface within the PC. There could be any number of reasons for this, but as of 2000, IEEE-1394 is not a major player in the storage industry. Some systems are now equipped with this interface, and a variety of storage devices are made for it, so it is a viable option if your system supports it, or if you wish to add support for it. IEEE-1394 does continue to grow in popularity in a variety of specialty markets, especially digital video, where it has established quite a following. It may well become the next big interfacing standard for consumer electronics devices like camcorders and VCRs. As for the PC world, the future is uncertain, and the creation of the new, faster USB 2.0 standard continues to keep the waters cloudy.

Note: For more information on IEEE-1394, see the web site of the *IEEE-1394 trade association*.

Next: Fibre Channel (FC-AL)

Fibre Channel (FC-AL)

You probably know that parallel SCSI is a high-end hard disk interface that is an alternative to the ubiquitous IDE/ATA. However, you may not realize that regular SCSI itself has a "big brother" of sorts. This interface is called *Fibre Channel*. The name comes from the fact that it was originally designed to operate over fiber-optic physical channels; copper wiring is now also supported. (The correct spelling is also "Fibre", not "Fiber"; this reflects the standard's European origins.) Like IEEE-1394, Fibre Channel is actually defined as part of the SCSI-3 family of standards, so it really is sort of a "sibling" to conventional SCSI. Sort of. :^)

Like regular SCSI, Fibre Channel is a collection of protocols and options, and it would take some time for me to explain them all in detail, which I'm not going to do at present. The current implementation that is in use is a subset of Fibre Channel called *Fibre Channel Arbitrated Loop* or *FC-AL*. FC-AL allows many servers and storage devices to be connected into what is essentially a "storage network". This configuration offers flexibility, performance and reliability advantages to high-end systems.

Despite being a serial interface, FC-AL allows for throughput of up to 4 Gbits/s, with future versions likely to increase this. One of the primary benefits of Fibre Channel is that when using fiber optic connections, devices can be separated by up to 10 kilometers--yes, 10,000 meters. Even with copper connections, 30 meters is the limit, which is pretty good compared to other interfaces.

So if Fibre Channel is so great, why isn't it used by PCs? The main reasons are cost, and the lack of necessity. FC-AL is currently used almost exclusively on servers--and groups of servers working together. It's a high-end interface that you aren't likely to run into in your personal system, at least for now. I did want to mention it, however, so you know it is out there. :^) I will provide more explanation of Fibre Channel in the future if and when it becomes more widely implemented. In the meantime, for more information on this technology, see the *Fibre Channel Industry Association web site*.

Next: Universal Serial Bus 2 (USB 2.0)

Universal Serial Bus 2 (USB 2.0)

After a slow start, the Universal Serial Bus (USB) is taking the PC world by storm. USB ports are showing up on all new PCs, and at the same time, a surprisingly wide array of USB peripherals is appearing on the market. However, the initial implementation of USB (version 1.1) has a critical restriction that limits its ability to function as an effective interface for high-speed devices, such as storage units: it is too slow. With a maximum bandwidth of 12 Mbits per second, conventional USB is already many times slower than what most optical and hard disks need, and that bandwidth must be shared amongst all peripherals using the interface.

Seeing the limitations of USB, Intel has spurred the development and implementation of *USB 2.0*, an updated version of the interface that increases throughput from 12 Mbits per second all the way up to 480 Mbits per second! USB 2.0 devices and systems will be both forward and backward compatible with USB 1.1 devices, which means that you will be able to mix both types of devices on systems that have either USB 1.1 or USB 2.0 host controllers. (Of course, systems without a USB 2.0 controller will only function at the slower 1.1 speed.)

You might wonder: why bother with this at all? Why not leave USB for slower devices, as it was originally intended, and use IEEE-1394 for high-speed devices, as *it* was intended? It's a good question. The answer is: that would make too much sense. :^) More seriously, one reason may be that USB has just been more widely-accepted in the PC world, and with USB 2.0 both backward and forward compatible with USB 1.1, hardware makers felt it made more sense to upgrade USB's capabilities. USB 2.0 may also be cheaper to implement. I suspect that the driving reason, however, is "more of the same" in terms of big companies clashing for control over the market: Apple Computer owns the FireWire name and has royalty rights on IEEE-1394, and Intel probably doesn't want to "play the game with Apple's ball", if you know what I mean. Just my personal suspicions, but this has happened many times before in the world of computers...

So will USB 2.0 become the next standard, or will IEEE-1394 finally gain acceptance in the PC world? Your guess is as good as mine. :^) As for our area of prime concern here, storage devices, USB 2.0 is a *big* win. While the fastest hard disks will eventually saturate 480 Mbits/second, it will be many years before USB 2.0's bandwidth comes even close to being a big issue, while USB 1.1 is today. With USB 2.0, serially-connected, high-speed hard disks and optical drives will be a practical reality. Look for USB 2.0 to hit the shelves early in 2001, barring any major complications in development (which are always a possibility...)

Note: For more information on USB 2.0, see the *USB Implementers Forum site*.

Next: Serial ATA

Serial ATA

The current IDE/ATA standard is a *parallel* interface; this means that multiple bits of data are transmitted at one time. In the case of ATA, 16 bits are moved across the interface simultaneously during each transfer. The advantage of a parallel interface is that it allows for high throughput; the problem with it is that as the frequency of the interface is increased, signaling problems and interference between signals become common. To combat this, techniques such as CRC and special 80-conductor cables are used in higher-speed transfer modes such as Ultra DMA. These are really "kludges" that are used to work around problems with the interface as it moves to higher speeds. A different approach, however, is to abandon the parallel concept in favor of a *serial* interface, where only one bit is transferred at a time. This is what the *Serial ATA* proposal is all about: creating a serial version of ATA for attaching IDE/ATA hard disks.

Obviously, in going from 16 bits to 1 bit, the speed of the interface must be increased by a factor of 16 just to "break even". The idea is that the simplicity of the serial interface will enable *much* higher speeds than would be possible from a parallel implementation, because the signaling problems are largely eliminated. In fact, this is the same reasoning that led to the creation of other high-speed serial interfaces, such as IEEE-1394. Serial ATA is still in development at the time that I write this page, but indications are that it will support maximum throughput of somewhere between 150 and 300 MB/s.

As enticing as the higher speed of the interface is its promise of improvements to some of the well-known (and well-hated) weaknesses of IDE/ATA. Since Serial ATA is a point-to-point serial protocol, each device communicates directly with the host system over a flexible, thin cable that can be made a reasonable length. This means no more master/slave jumpering hassles, elimination of the difficult-to-deal-with ribbon cables, and more flexibility in the placement of devices within the PC. It's also possible that hot-swapping will be supported when Serial ATA is implemented, which would be a welcome feature as far as I am concerned!

We'll have to see what happens regarding Serial ATA. I am a little skeptical about both its likelihood of being successfully implemented, and in fact with whether it is even needed. Proponents of serial hard disk interfaces have been saying "the end is nigh" regarding parallel interfaces for many years; I remember hearing about the "end of the road" when parallel IDE/ATA was at 16.6 MB/s, and parallel SCSI was at 40 MB/s, but these interfaces continue to get faster and faster. However, if they can really make Serial ATA work, especially at such high speeds, it may be a welcome improvement on what has really become a rather outdated design.

I guess we'll just have to see what happens. At any rate, I wouldn't look for Serial ATA to even be introduced to the market until late 2001 or even 2002. Even if it is successful, conventional parallel IDE/ATA will be popular for many, many years due to the hundreds of millions of systems that use it, so don't worry about obsolescence just yet. :^)

Next: Integrated Drive Electronics / AT Attachment (IDE/ATA) Interface

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.

Next: Overview of the IDE/ATA Interface

Overview and History of the IDE/ATA Interface

As I mentioned in the introductory page to my coverage of this interface, its common name, "IDE", also tells you quite a bit about its history. IDE drives were the first ones to popularize integrating the logic controller onto the hard disk itself. This change corrected many of the problems that had been associated with hard disks up to that point, such as poor signal integrity, complexity and the need for every controller to be "generically" capable of dealing with any hard disk. You can read more about the logic board and its significance on this page. Here's a (brief) bit of history of how integrated disk controllers, and the IDE/ATA interface itself, came about.

The very first hard disks to have integrated controllers weren't technically using the IDE/ATA interface as we currently know it. They were in fact so-called "hardcards", which were designed and sold by the "Plus Development" division of Quantum. These devices were simply 3.5" hard disks that were mounted directly to a controller card that plugged into an ISA expansion slot. They were popular amongst those who wanted to add additional storage to existing systems, because of their simplicity: with the disk and controller integrated, one could just slap the package into an existing PC like any add-in peripheral, without affecting the existing hard disk and controller.

While an ingenious idea--one that basically put Quantum "on the map" in the hard disk world--these hardcards had several problems. Hard disks are heavy, and are mechanical devices. They just aren't well-suited to being mounted in expansion slots like a video card or modem. Physical support for expansion cards is poor, because they are held in place using just one screw; this resulted in vibration issues. The width of the drive would typically block off at least one additional bus slot, meaning one of these devices took two expansion bus slots. Cooling was also sometimes a problem. Of course, all of these are physical implementation issues, and don't detract from the significance of the advance in integrating the controller and disk assembly.

It didn't take long until manufacturers realized that there was really no reason to keep the hard disk physically on the controller at all. They decided to put the controller on the bottom of the hard disk and move the entire hard disk and controller assembly to a regular drive bay. The connection to the system bus was maintained through the use of a cable that ran either directly to a system bus slot, or to a small interfacing card that plugged into a system bus slot. In many ways, then, these drives were connected directly to the ISA system bus. The official name for the interface, "AT Attachment", reflects this, as the IBM PC/AT was the first PC to use the now-standard 16-bit ISA bus.

The origins of the actual IDE/ATA interface go back to around the same time of the hardcard. Compaq Computer, an up-and-coming competitor to IBM at the time (and still a major market player today) wanted to integrate the hard disk controller into the hard disk itself to improve flexibility and save a bus slot. They worked to create the precursors of the modern IDE interface electronics with Western Digital (which was then an interface logic company and not a hard disk maker) and a division of Control Data Corporation that is now part of Seagate--isn't this fun? :^) Compaq began selling PCs with integrated hard disks using Western Digital controllers starting with their IBM-compatible Deskpro 386 in 1986. Other manufacturers quickly caught on to the idea and the IDE concept grew in popularity rather quickly.

As system and hard disk performance improved, the slow speed of the ISA bus became an issue, so interface cards--often called *controller cards*, despite the confusion that this causes--were created for the higher speed VESA local bus, and then the PCI bus. Today, all modern PCs have their IDE/ATA interface attached directly to the PC bus.

The next evolution of how IDE/ATA drives interface to the system occurred when it became obvious that every PC was going to have a hard disk, and it was therefore silly to waste an expansion slot even on a hard disk interface card. Chipset manufacturers began integrating IDE/ATA hard disk controllers into their chipsets, so that instead of connecting the drives to a controller card, they were connected directly to the motherboard. With this change, integration of the interface was complete, with all the logic either on the motherboard, or the hard disk itself. This is the technique that is used today (though one can still add a physical controller card if necessary.)

In terms of its basic operation, the IDE/ATA interface is fairly straight-forward, and also reflects its origins as an extension of the ISA system bus. The connection between the system and the hard disks is 16 bits wide, so two bytes of data are passed at a time between the system and any hard disk. This is true regardless of the width of the system bus, and persists even today with high-performance enhancements like Ultra DMA. Two drives are supported on each IDE/ATA channel, with special signaling used to ensure that commands sent for one drive don't interfere with the other. Over time, many performance and reliability features have evolved as the interface has matured, which I will go into in detail as we discuss the various aspects of the IDE/ATA interface.

Note: In the early days of IDE, there were two other variants on the IDE/ATA interface, that were not compatible with regular IDE/ATA. One was an 8-bit version intended for use on the ancient 8-bit ISA bus of the first IBM PC/XT machines and clones. Another was a 16-bit version designed for IBM's MCA (MicroChannel) systems. Both of these have been obsolete for a decade or more and are no longer seen in modern systems; I do not cover them on this site (other than this note. :^))

Next: Official IDE/ATA Standards and Feature Sets

Official IDE/ATA Standards and Feature Sets

There's an old joke that says the great thing about standards is that there are so many to choose from. Anyone who has tried to understand hard disk interface standards knows exactly what this means. :^) To help you comprehend what can be a very confusing subject, I have spent considerable time researching all of the issues related to IDE/ATA standards, and have presented them here in what I hope is fairly plain English.

Standards probably get just a bit too much flak. Despite the confusion that standards can cause, that's *nothing* compared to the confusion caused by a *lack* of standards. So it was in the early days of the IDE/ATA interface. During the late 1980s, IDE/ATA grew in popularity, but there were no standards in place to ensure that the interface decisions made by different hardware companies were compatible with each other. Several manufacturers succumbed to the temptation to make slight "improvements" to the interface. As a result, many early IDE/ATA drives exhibited compatibility problems, especially when one attempted to hook up a master and slave device on the same cable.

Recognizing the potential for utter chaos here, a number of designers and manufacturers of hard disks and related technology got together to form the *Common Access Method* (*CAM*) committee on the AT Attachment interface.

The first document describing the proposed IDE/ATA standard was introduced in early 1989. It was submitted in 1990 to the American National Standards Institute (ANSI), and eventually became the first formal ATA standard. The CAM committee was eventually replaced with other similar groups charged with the various tasks associated with managing the IDE/ATA interface.

Today, ATA standards are developed, maintained and approved by a number of related organizations, each playing a particular role. Here's how they all fit together:

- **American National Standards Institute:** *ANSI* is commonly thought of as an organization that develops and maintains standards, but in fact they do neither. They are an oversight and accrediting organization that facilitates and manages the standards development process. As such, they are the "high level management" of the standards world. They qualify other organizations as *Standards Developing Organizations* or *SDOs*. ANSI also publishes standards once they have been developed and approved.
- **Information Technology Industry Council:** *ITIC* is a group of several dozen companies in the information technology (computer) industry. ITIC is the SDO approved by ANSI to develop and process standards related to many computer-related topics.
- **National Committee for Information Technology:** *NCITS* is a committee established by ITIC to develop and maintain standards related to the information technology world. NCITS was formerly known under the name "Accredited Standards Committee X3, Information Technology", or more commonly, just "X3". It maintains several sub-committees that develop and maintain standards for various technical subjects.
- **T13 Technical Committee:** *T13* is the actual technical standards committee responsible for the IDE/ATA interface specifically.

So basically, T13 is the group that actually does the work of developing new IDE/ATA standards. ;^) The T13 group is comprised primarily of technical people from various hard disk and other technology companies, but the group (and the development process itself) is open to all interested parties. Comments and opinions on standards under development are welcomed from anyone, not just T13 members. The standards development process is intended to create consensus, to ensure that everyone who will be developing hardware and software agrees on how to implement new technology.

Once the T13 group is done with a particular version of the standard, they submit it to NCITS and ANSI for approval. This approval process can take some time; which is why the official standards are usually published several months, or even years, after the technology they describe is actually implemented. While approval of the standard is underway, companies develop products using technology described in the standard, confident that agreement has already been reached. Meanwhile, the T13 group starts work on the next version of the standard.

Now that you understand how the standards process works, you are in much better shape to read the rest of this section, which describes all of the different standards that describe the IDE/ATA interface. They are listed in approximately chronological order. Remember when reading that subsequent standards build upon earlier ones, and that in general, hardware implementing newer standards is backward-compatible with older hardware.

Note: Standards that have been approved and published by ANSI are available for purchase in either print form or electronic format from *ANSI's web site*. Draft standards that are under development (as well as older drafts of approved standards) can be found at the *T13 Technical Committee web site*.

Tip: If there's an IDE/ATA standard you are looking for information on but can't find in this section, it might in fact be an "unofficial" standard or marketing term.

Next: ATA (ATA-1)

ATA (ATA-1)

The first formal standard defining the AT Attachment interface was submitted to ANSI for approval in 1990. It took a looooooong time for this first ATA standard to be approved. :^) Presumably, it took so long because it was the first standard to define the interface, and therefore much debate and discussion probably took place during the approval process. It was finally published in 1994 as ANSI standard X3.221-1994, titled *AT Attachment Interface for Disk Drives*. This standard is sometimes called *ATA-1* to distinguish it from its successors.

The original IDE/ATA standard defines the following features and transfer modes:

- **Two Hard Disks:** The specification calls for a single channel in a PC, shared by two devices that are configured as master and slave.
- **PIO Modes:** ATA includes support for PIO modes 0, 1 and 2.
- **DMA Modes:** ATA includes support for single word DMA modes 0, 1 and 2, and multiword DMA mode 0.

"Plain" ATA does not include support for enhancements such as ATAPI support for non-hard-disk IDE/ATA devices, block mode transfers, logical block addressing, Ultra DMA modes or other advanced features. Drives developed to meet this standard are no longer made, as the standard is old and obsolete. In fact, at the recommendation of the T13 Technical Committee, ATA-1 was withdrawn as an official ANSI standard in 1999. This is presumably due to its age, and the large number of replacement ATA standards already published by that time.

Next: ATA-2

ATA-2

The original ATA standard defined features that were appropriate for early IDE/ATA hard disks. However, it was not well-suited to support the growing size and performance needs of a newer breed of hard disks. These disks required faster transfer rates and support for enhanced features.

In an ideal world, the standards committee would have gotten the various hard disk manufacturers together to define a new standard to support the added features everyone wanted. Unfortunately, several companies were impatient, and once again started the industry down the road to incompatible proprietary extensions to the original ATA standard. Seagate defined what it called "Fast ATA", an extension to regular ATA, and "Fast ATA-2" soon followed. These extensions were also picked up and used by Quantum. Western Digital, meanwhile, created "Enhanced IDE" or "EIDE", a somewhat different ATA feature set expansion. All of this happened in around 1994.

To try to once again correct the growing confusion being caused by all these unofficial standards, the ATA interface committee created a new, official ATA-2 specification that essentially combines the features and attributes defined by the marketing programs created by Seagate, Quantum and Western Digital. This standard was published in 1996 as ANSI standard X3.279-1996, *AT Attachment Interface with Extensions*.

ATA-2 was a significant enhancement of the original ATA standard. It defines the following improvements over the base ATA standard (with which it is backward compatible):

- **Faster PIO Modes:** ATA-2 adds the faster PIO modes 3 and 4 to those supported by ATA.
- **Faster DMA Modes:** ATA-2 adds multiword DMA modes 1 and 2 to the ATA modes.
- **Block Transfers:** ATA-2 adds commands to allow block transfers for improved performance.
- **Logical Block Addressing (LBA):** ATA-2 defines support (by the hard disk) for logical block addressing. Using LBA requires BIOS support on the other end of the interface as well.
- **Improved "Identify Drive" Command:** This command allows hard disks to respond to inquiries from software, with more accurate information about their geometry and other characteristics.

Unfortunately, even after consensus was reached on ATA-2, the old marketing terms continued to be used. Fortunately, all of the drives of this era have now passed into obsolescence, and the hard disk companies are in much better agreement now on what terms should be used to describe the hard disk interface. Although the marketing people keep trying. ;^)

Next: ATA-3

ATA-3

The ATA-3 standard is a minor revision of ATA-2, which was published in 1997 as ANSI standard X3.298-1997, *AT Attachment 3 Interface*. It defines the following improvements compared to ATA-2 (with which it is backward compatible):

- **Improved Reliability:** ATA-3 improves the reliability of the higher-speed transfer modes, which can be an issue due to the low-performance standard cable used up to that point in IDE/ATA. (An improved cable was defined as part of ATA/ATAPI-4.)
- **Self-Monitoring Analysis and Reporting Technology (SMART):** ATA-3 introduced this reliability feature.
- **Security Feature:** ATA-3 defined security mode, which allows devices to be protected with a password.

ATA-3 was approved rather quickly after ATA-2, while the market was still spinning from all the non-standard "ATA-2-like" interface names being tossed. This, combined with the fact that ATA-3 introduced no higher-performance transfer modes, caused it to be all but ignored in the marketplace. Hard disk manufacturers added features defined in the standard (such as SMART) to their drives, but didn't tend to use the "ATA-3" term itself in their literature.

Note: You may see a so-called "PIO Mode 5" described in some places, with the claim that it was introduced in ATA-3. This mode was suggested by some controller manufacturers but never approved and never implemented. It is not defined in any of the ATA standards and only exists in some BIOS setup programs... See the discussion of PIO modes for more information.

Note: ATA-3 does not define any of the Ultra DMA modes; these were first defined with ATA/ATAPI-4. ATA-3 is also not the same as "ATA-33", a slang term for the 33 MB/s first version of Ultra ATA, *itself* a slang term for the 33 MB/s Ultra DMA transfer mode 2. :^)

Next: SFF-8020 / ATA Packet Interface (ATAPI)

SFF-8020 / ATA Packet Interface (ATAPI)

Originally, the IDE/ATA interface was designed to work only with hard disks. CD-ROMs and tape drives used either proprietary interfaces (often implemented on sound cards), the floppy disk interface (which is slow and cumbersome) or SCSI. In the early 1990s it became apparent that there would be enormous advantages to using the standard IDE/ATA interface to support devices other than hard disks, due to its high performance, relative simplicity, and universality. The intention was not to replace SCSI of course, but rather to get rid of the proprietary interfaces (which nobody really likes) and the slow floppy interface for tape drives.

Unfortunately, because of how the ATA command structure works, it wasn't possible to simply put non-hard-disk devices on the IDE channel and expect them to work. Therefore, a special protocol was developed called the *AT Attachment Packet Interface* or *ATAPI*. The ATAPI standard is used for devices like optical, tape and removable storage drives. It enables them to plug into the standard IDE cable used by IDE/ATA hard disks, and be configured as master or slave, etc. just like a hard disk would be. When you see a CD-ROM or other non-hard-disk peripheral advertised as being an "IDE device" or working with IDE, it is really using the ATAPI protocol.

Internally, however, the ATAPI protocol is not identical to the standard ATA (ATA-2, etc.) command set used by hard disks at all. The name "packet interface" comes from the fact that commands to ATAPI devices are sent in groups called *packets*. ATAPI in general is a much more complex interface than regular ATA, and in some ways resembles SCSI more than IDE in terms of its command set and operation. (At the time it was created, SCSI was the interface of choice for many CD-ROM and higher-end tape drives.)

A special ATAPI driver is used to communicate with ATAPI devices. This driver must be loaded into memory before the device can be accessed (most newer operating systems support ATAPI internally and in essence, load their own drivers for the interface). The actual transfers over the channel use regular PIO or DMA modes, just like hard disks, although support for the various modes differs much more widely by device than it does for hard disks. For the most part, ATAPI devices will coexist with IDE/ATA devices and from the user's perspective, they behave as if they are regular IDE/ATA hard disks on the channel. Newer BIOSes will even allow booting from ATAPI CD-ROM drives.

The first standard that described ATAPI wasn't actually even developed by the people who maintain the ATA standards. It was defined by the *Small Form Factor* committee, an industry group that traditionally defined standards for physical issues like PC cables and screw hole locations, but somehow got involved in storage interfacing. The first ATAPI standard document produced by this group was called SFF-8020 (later renamed INF-8020), which is now quite old and obsolete. In the late 1990s, the T13 Technical Committee took over control of the ATAPI command set and protocol, combining it with ATA into the ATA/ATAPI-4 standard.

Next: ATA/ATAPI-4

ATA/ATAPI-4

The next significant enhancement to the ATA standard after ATA-2 saw the ATA Packet Interface (ATAPI) feature set merged with the conventional ATA command set and protocols to create *ATA/ATAPI-4*. This standard was published by ANSI in 1998 as NCITS 317-1998, *AT Attachment with Packet Interface Extensions*. (Note the change to "NCITS" in the document number, from the "X3" used in earlier ATA standards; see here for an explanation of these organization names.)

Aside from combining ATA and ATAPI, this standard defined several other significant enhancements and changes:

- **Ultra DMA Modes:** High-speed Ultra DMA modes 0, 1 and 2, defining transfer rates of 16.7, 25 and 33.3 MB/s were created.
- **High-Performance IDE Cable:** An improved, 80-conductor IDE cable was first defined in this standard. It was thought that the higher-speed Ultra DMA modes would require the use of this cable in order to eliminate interference caused by their higher speed. In the end, the use of this cable was left "optional" for these modes. (It became mandatory under the still faster UDMA modes defined in ATA/ATAPI-5.)
- **Cyclical Redundancy Checking (CRC):** This feature was added to ensure the integrity of data sent using the faster Ultra DMA modes. Read more about it here.
- **Advanced Commands Defined:** Special command queuing and overlapping protocols were defined.
- **Command Removal:** The command set was "cleaned up", with several older, obsolete commands removed.

Of course, the Ultra DMA modes were the most exciting part of this new standard. Ultra DMA modes 0 and 1 were never really implemented by hard disk manufacturers, but UDMA mode 2 made quite a splash, as it doubled the throughput of the fastest transfer mode then available. Ultra DMA mode 2 was quickly dubbed "Ultra DMA/33", and drives conforming to ATA/ATAPI-4 are often called "Ultra ATA/33" drives, which technically does not exist.

Next: ATA/ATAPI-5

ATA/ATAPI-5

Not content to rest on their laurels with the adoption of ATA/ATAPI-4, the T13 committee immediately began work on its next generation, *ATA/ATAPI-5*. This standard was published by ANSI in 2000 as NCITS 340-2000, *AT Attachment with Packet Interface - 5*.

The changes defined in ATA/ATAPI-5 include:

- **New Ultra DMA Modes:** Higher-speed Ultra DMA modes 3 and 4, defining transfer rates of 44.4 and 66.7 MB/s were specified.
- **Mandatory 80-Conductor IDE Cable Use:** The improved 80-conductor IDE cable first defined in ATA/ATAPI-4 for optional use, is made mandatory for UDMA modes 3 and 4. ATA/ATAPI-5 also defines a method by which a host system can detect if an 80-conductor cable is in use, so it can determine whether or not to enable the higher speed transfer modes.
- **Miscellaneous Command Changes:** A few interface commands were changed, and some old ones deleted.

Like ATA-3, not that many changes were made in ATA/ATAPI-5 (compared to ATA/ATAPI-4 and ATA-2, for example). *Unlike* ATA-3, the main change made here was a high-profile one: another doubling of the throughput of the interface to 66.7 MB/second. Unsurprisingly, the same companies that called ATA/ATAPI-4 drives "Ultra ATA/33" labeled ATA/ATAPI-5 drives running Ultra

DMA mode 4 as "Ultra ATA/66". During 1999 and early 2000, new IDE/ATA drives conforming to this standard were the most common on the market.

Next: ATA/ATAPI-6

ATA/ATAPI-6

At the time that I write this in late 2000, the *T13 Technical Committee* is working on he next version of the ATA standard, *ATA/ATAPI-6*. It is likely that this standard will be completed in 2001 and published sometime later that year or early in 2002.

Since this standard is still in development, it is impossible to be sure exactly what features and changes it will include. One addition to the standard does seem almost certain: the new Ultra DMA mode 5, which increases transfer throughput to 100 MB/s. Since this is already the standard on currently-shipping drives, I can't imagine it not making the next standard! Beyond that, only the T13 folks know, and at this point, perhaps not even them. :^) Aside from Ultra DMA mode 5, some of the rumored possible new features for the next standard include:

- **LBA Address Size Expansion:** Hard disk sizes are now approaching the maximum that can be represented under the traditional 28-bit LBA scheme; this is the 137 GB size barrier. Most people don't know anything about this "size barrier of the future" now, but I predict that in 2001 or 2002 it's going to be a hot topic of conversation. :^) To get around this limitation, an addressing mode will probably be included in ATA/ATAPI-6 that expands the address width from 28 bits to either 48 or 64 (either of which will keep us busy for quite a while...)
- **Hard Disk Noise Reduction ("Acoustic Management"):** Some hard disk companies are working on technologies to allow the mechanics of the hard disk to be modified under software control, letting the user choose between higher performance or quieter operation. Commands to cover this feature may make the next standard, as they are mentioned in the current draft.
- **Audio and Video Streaming:** Some manufacturers have apparently suggested new commands related to multimedia streaming, but I don't know any more about them than that right now.

I will update this page as more becomes known about the ATA/ATAPI-6; or at least, as *I* come to know more about it. :^)

Next: Summary of IDE/ATA Standards

Summary of IDE/ATA Standards

With the creation of several new ATA standards over the last few years, there are now quite a few of them "out there". The table below provides a quick summary of the different official IDE/ATA interfaces, showing their key attributes and features.

Interface Standard	ANSI Standard Number (includes date)	PIO Modes Added	DMA Modes Added	Ultra DMA Modes Added	Notable Features or Enhancements Introduced
ATA-1	X3.221-1994	0, 1, 2	Single word 0, 1, 2; multiword 0	--	--
ATA-2	X3.279-1996	3, 4	Multiword 1, 2	--	Block transfers, Logical block addressing, Improved identify drive command
ATA-3	X3.298-1997	--	--	--	Improved reliability, SMART, Drive security
ATA/ATAPI-4	NCITS 317-1998	--	--	0, 1, 2	Ultra DMA, 80-conductor IDE cable, CRC
ATA/ATAPI-5	NCITS 340-2000	--	--	3, 4	--

ATA/ATAPI-6	Under Development	--	--	5?	LBA expansion? Acoustic management? Multimedia streaming?

Note: Since the columns only show which modes and features were *added* in a given standard, the table should be viewed as *cumulative*. This means that all standards shown include all the modes and features of all prior standards as well. Standards are also backward-compatible with drives designed to earlier standards.

Next: Unofficial IDE/ATA Standards and Marketing Programs

Unofficial IDE/ATA Standards and Marketing Programs

There are a number of official IDE/ATA standards that have defined the characteristics of the IDE/ATA interface over the years. The existence of so many standards can cause such confusion unless there is a good explanation of what they all are, which is why I wrote one. :^) Unfortunately, the situation is actually much *more* confusing than one would expect merely from the evolution of a formal standard over a period of a decade or so. The real confusion comes from all of the *unofficial* standards that are sometimes created by manufacturers, and the *marketing terms* that they think up to try to position and sell their products.

Unofficial standards often arise due to impatience on the part of a manufacturer that does not want to wait for the next version of a real standard. Sometimes a company has a desire to try to lead the market by extending a formal standard themselves, in the hopes that other companies will follow. A manufacturer may take an existing standard and add features or capabilities in an attempt to win part of the market from other companies that are still adhering to the "old standard". All of this of course prompts the other companies to respond with their own "extensions" to existing standards, and the result is often a chaos of competing and incompatible interface variations. This is why the formal standards development process now used for the IDE/ATA interface was created, and why it is so important.

Marketing programs cause confusion by creating different names for official standards or feature sets defined in the official standards. This may be done because the formal name for a feature is not considered sufficiently "exciting", or in an attempt to build name-brand recognition of, or market preference for, a particular hard disk brand. Here again, the result is often confusion, because two companies may be selling drives with the identical interface, but using two very different names.

In an attempt to shed a little light on all of these terms and "unofficial" standards, I describe each of the common ones here. Where relevant, I provide links to the official standards and features that correspond to the unofficial terms. If you have not yet done so, you may wish to first read the section describing official IDE/ATA standards.

Next: Integrated Drive Electronics (IDE)

Integrated Drive Electronics (IDE)

Drives that use the interface officially known as AT Attachment or ATA are also often called something else entirely: *Integrated Drive Electronics* or *IDE* drives. In fact, the term "IDE" is probably more widely used than the correct name for the interface! (This is changing, however, as new terms such as "Ultra ATA" grow in popularity.) IDE can be considered the unofficial "overall name" for this hard disk interface; it has been used since the earliest days of these drives, and will probably always be used by a large segment of the industry.

The reason for the name, of course, is that the IDE interface was the first where the logic board was integrated on the hard disk itself. As described in some detail in the overview and history of the interface, drives prior to this point had the hard drive control logic on a separate controller card that plugged in a system bus slot. This led to a number of compatibility and reliability problems that were corrected by mating the logic board to the hard disk itself.

The name "IDE" really reflects this design decision and has nothing to do with the interface per se. Today, all drives have integrated logic boards, including those that use interfaces quite different from ATA, such as SCSI or USB. However, habits are hard to break in the computer world, and use of the name persists. This has the potential for confusion, though most people today know that "IDE" refers to the IDE/ATA interface specifically. (One of the reasons that the name "IDE" was never adopted for the formal standards is that the developers of the standards considered it confusing.)

It's important to remember that today, being told that a particular drive is an "IDE drive" tells you only that it uses some variant of the IDE/ATA interface. "IDE" by itself is a generic term that does not tell you anything about the drive, such as what modes it supports or what official standard it adheres to. You need to find out more about the drive to understand the details of its interface.

Next: Enhanced IDE (EIDE)

Enhanced IDE (EIDE)

Enhanced IDE, also called *EIDE*, is a term that Western Digital coined in 1994 to represent a particular set of extensions it devised to the original AT Attachment standard. At that time, the official ATA standard was rather limiting, and work was progressing towards the new ATA-2 standard. Western Digital decided that it did not want to wait for the new standard, and also that it could better position itself as a market leader by creating a new feature set for (then) future drives. The name "Enhanced IDE" was presumably selected to build upon the common name for ATA then in popular use: IDE.

The original Enhanced IDE program included the following improvements over ATA:

- **ATA-2 Enhancements:** EIDE includes all (most?) of the improvements that are defined as part of the ATA-2 standard, including the higher-speed transfer modes.
- **ATAPI:** The EIDE definition includes support for non-hard-disk ATAPI devices on the IDE/ATA channel. Note that at that time, ATAPI was not part of the ATA standard at all.
- **Dual IDE/ATA Host Adapters:** The EIDE standard specifically includes support for dual IDE/ATA channels, allowing four IDE/ATA/ATAPI devices to be used. (In fact, the ATA standard at the time never precluded the use of two IDE/ATA channels; it just was not commonly done.)

EIDE has become a widely-accepted term in the industry, which would be great if not for the fact that it is so incredibly confusing. Objections to EIDE include the following issues:

- **Proprietary Standard:** EIDE is not an official standard, and it competed with other non-standard IDE/ATA terms like Fast ATA. Of course, that criticism applies not just to EIDE.
- **Scope:** Much of the criticism of the original EIDE program is that its scope was too wide, and that it encompassed features that are really the domain of the BIOS. For example, support for dual IDE/ATA host adapters, meaning a secondary IDE/ATA channel, has nothing to do with the interface or the hard disk itself. And ATAPI is a standard that is defined for use with optical drives and other non-hard-disk devices, which again requires BIOS and driver support and really has nothing to do with the hard disk. At the time, other hard disk manufacturers not only excluded these from their own standard proposals (such as Fast ATA), they made a point of criticizing Western Digital for bringing these issues into the interface discussion.
- **The Word "Enhanced":** The choice of the word "enhanced" was unfortunate, as it led to confusion in another area. At around the same time that EIDE was introduced, the 504 MB hard disk size barrier became a big issue. To work around this required an "enhanced BIOS". Because of the fact that both of these phrases use the word "enhanced", and because EIDE defines BIOS support standards, many people have come to think of the terms as interchangeable when they really are not. This has lead to claims that you need an enhanced IDE interface to support disks over 504 MB, when you don't--you just need an enhanced *BIOS*. As if this weren't bad enough, some companies advertised add-in cards with enhanced BIOSes as "enhanced IDE cards"! :^)
- **Redefinition:** Since EIDE is Western Digital's term, they have the right to change its meaning, and unfortunately, they do this on a regular basis. At first, EIDE included only PIO modes up to mode 3; then mode 4 was added. When the new Ultra DMA modes came out, WD of course added support for them to their newest models, but they kept calling the drives "EIDE"! Today other drive manufacturers also say things like "EIDE compatible", leaving you wondering what exactly this means.

Some people in the hard disk industry apparently feel that the creation of "Enhanced IDE" was one of the worst things to ever happen to the IDE/ATA interface! I think that is probably a bit over-stated, though I *do* agree that it is probably one of the most *confusing* things to ever happen to the IDE/ATA interface. :^) Much of the criticism is valid, but some of it is just the usual conflicts between rivals in a very competitive industry. And I do think Western Digital's goal of expanding IDE/ATA capabilities was a laudable one, even if the implementation of the program left a bit to be desired.

Of all the criticisms leveled at Western Digital, there's one that I personally agree with strongly, and that's the issue of redefining the term. Every time the IDE/ATA interface standards change, Western Digital changes the actual interface specifics of its drives, but continues to list the interface of the drive as just "EIDE". A term that is constantly redefined is a term that is utterly meaningless. As a result, I can only tell people at this point that if they see a drive labeled as being "EIDE", to keep digging to find out the specifics of the modes and official standards it supports, because "EIDE" by itself doesn't tell you anything (other than the generic interface of the drive, as the terms "IDE" or "ATA" do.) It would be nice if Western Digital would just drop the term entirely, but I doubt this will happen since they have spent so many years promoting it.

Next: Fast ATA and Fast ATA-2

Fast ATA and Fast ATA-2

At around the same time that Western Digital began promoting Enhanced IDE, Seagate Technology created its own unofficial extension to the original AT Attachment standard. Like Western Digital, they wanted to improve the performance of newer hard drives; they also didn't want to wait for the next official standard. And they certainly weren't going to just follow Western Digital! So they created a new "standard" that they called *Fast ATA*. Quantum joined Seagate in supporting this program. Later, *Fast ATA-2* was created, adding more functionality to Fast ATA.

While at the time Western Digital got the lion's share of the criticism for its Enhanced IDE program, these two terms fare only a bit better in the confusion department. The term "Fast" is meant to connote that the interface runs at higher speed than regular ATA; which it did. The names are confusing, however, because they make it sound like Fast ATA-2 is related to ATA-2, and Fast ATA is related to regular ATA. This is not so, as both are really subsets of ATA-2, including the higher speed transfer modes and some of the other features of the official standard. Specifically, Fast ATA includes PIO mode 3 and multiword DMA mode 1. Fast ATA-2 includes PIO modes 3 and 4, and multiword DMA modes 1 and 2.

I will give Quantum and Seagate credit for one thing: they at least stopped using these confusing terms in the late 1990s, while Western Digital keeps painting fresh bulls-eyes around the term "EIDE". ;^) As the IDE/ATA interface evolved, these "Fast" terms were eventually dropped, in favor of terms like "Ultra ATA" (which are still not technically correct, but at least are universally used in the industry).

Unfortunately, the "Fast" business persists in some places; I recently saw a datasheet for a Maxtor drive that claimed it was "Fast ATA-4 compatible". I have no idea what in the blazes "Fast ATA-4" is supposed to be, but the same spec sheet says that drive is capable of Ultra DMA/66 transfers. That's Ultra DMA mode 4, which was defined in the ATA/ATAPI-5 standard, not ATA/ATAPI-4. Perhaps this odd "Fast ATA-4" term was supposed to mean that the drive was using the newer DMA mode that was anticipated to make the ATA/ATAPI-5 standard, and therefore was faster than ATA-4? Argh, these marketing people drive me crazy sometimes. :^)

Next: Ultra ATA (Ultra ATA/33)

Ultra ATA (Ultra ATA/33)

With the introduction of the new Ultra DMA modes in what became the ATA/ATAPI-4 standard, manufacturers quickly adopted the new technology. The doubling of the maximum interface transfer rate was big news, and companies wasted no time trumpeting the new capabilities of the drives. One of the most common terms used to refer to these new drives was *Ultra ATA*, which appears to be a combination of the terms "Ultra DMA" and "ATA". Sometimes the drives were called *Ultra ATA/33*, referring to the maximum speed of the new Ultra DMA modes, 33 MB/s. Others called the drives just *ATA/33*. Still others called the drives late for dinner. ;^)

All of these are unofficial marketing terms, and not real standards. The bottom line is that even though you will see it used all the time, there really is no such thing as "Ultra ATA", at least in the official ATA standards. A drive that is marketed as using "Ultra ATA/33" is actually using Ultra DMA mode 2, which provides throughput at 33 MB/s. Some drives are sold as "Ultra DMA/33", which is a bit more accurate, but even that is a "slang" term. The correct designation for the interface of such drives would probably be "ATA-4, using Ultra DMA mode 2". That's a bit cumbersome and kind of boring, of course; I would never make a good marketing person. :^)

Note: When Ultra DMA mode drives first started being sold, they all operated at 33 MB/s, and thus "Ultra DMA" or "Ultra ATA" (without a number) implied mode 2 (33 MB/s). Today, there are faster Ultra DMA modes, so be careful about interpreting what those terms mean if no number is provided. They may be older drives using the Ultra DMA/33 standard, or the term may be being used generically to refer to drives using any of the Ultra DMA modes.

Next: Ultra ATA/66

Ultra ATA/66

Within a year or two of the introduction of Ultra DMA modes in ATA/ATAPI-4, two new Ultra DMA modes were created. These new modes allowed data transfer at 44 MB/s and 66 MB/s respectively (actually, the latter should really be 67 MB/s, since the number is really two-thirds of 100, but anyway...) The 44 MB/s speed never caught on, but new drives appeared on the market implementing Ultra DMA mode 4 at 66 MB/s. Just as the first 33 MB/s Ultra DMA mode drives were dubbed Ultra ATA/33, the new drives of course were called *Ultra ATA/66* or *ATA/66*.

As with the original ATA/33, there is no real standard called "Ultra ATA/66"; this is a slang term for ATA drives using 66 MB/s Ultra DMA (UDMA mode 4). Still, this is the common term given to drives that use the new Ultra DMA modes defined in the true standard, ATA/ATAPI-5. In most cases you can assume that drives sold as Ultra ATA/66 are compatible with ATA/ATAPI-5, though early "Ultra ATA/66" drives may not include all the features defined in the formal standard.

Note: At some point in your travels, you might also stumble upon some documentation that refers to the following mysterious specification: "Ultra ATA/66+". See the discussion of Ultra ATA/100 for the amusing story behind this term. :^)

Next: Ultra ATA/100

Ultra ATA/100

If you've already read my discussions of Ultra ATA/33 and Ultra ATA/66, you can probably guess what I am going to say on this page already. :^) Yes, *Ultra ATA/100* is the marketing term for hard disks that use the new Ultra DMA mode 5, supporting interface transfers at 100 MB/s. This new Ultra DMA mode is the fastest available as of this writing in late 2000. It has not been codified in any formally approved and published ATA standard, but will be documented in the new ATA/ATAPI-6 standard that is currently under development.

Pretty much all new ATA drives are now sold using some variant of the "Ultra ATA/100" moniker. Note that for a few weeks in 2000, some IBM "Ultra ATA/100" Deskstar drives were advertised as the somewhat strange "Ultra ATA/66+". The story behind this is rather amusing. Earlier in that year, work was progressing on drafts for the new ATA/ATAPI-6 standard. The key feature of this new standard was Ultra DMA mode 5, allowing interface speeds of 100 MB/s. Even though all hard drive companies are now selling drives that operate at this speed (despite the fact that ATA/ATAPI-6 is still in development), there apparently was some sort of agreement in place that no drives would be marketed at the 100 MB/s speed before a particular date. Someone at IBM obviously jumped the gun, and specifications for IBM Ultra ATA/100 drives began showing up early on the Internet. Unsurprisingly, people started asking what exactly this was all about!

So IBM apparently had to stop selling the drives as "Ultra ATA/100". They still wanted to send everyone the message, however, that these drives would be compatible with the new higher-speed transfer mode. So some marketing genius decided to label the drives as "Ultra ATA/66+". :^) I often praise IBM in my hard disk materials for their industry leadership role, but this was a really silly move in my opinion. Since of course nobody knew what on earth "Ultra ATA/66+" was supposed to mean, this led to even more confusion than the too-early use of "Ultra ATA/100" did. Fortunately, a couple of months later the magic date (whatever it was) passed and the "Ultra ATA/66+" nonsense went away, replaced with "Ultra ATA/100" again. You can still find mention of "66+" around, if you do some searching for that term on the web.

Next: Plug and Play ATA

Plug and Play ATA

A final "standard" that you may occasionally see reference to is something called *Plug and Play ATA*. This appears to be a proposal made by Quantum and Hewlett-Packard to standardize the connection and configuration of ATA hard disks in order to eliminate the need for master and slave configuration jumpers. The idea is based on the cable select feature that has been used by a number of manufacturers to get around the need for setting specific drives as master or slave.

This specification has never gained any widespread acceptance; after reading it several times I can understand why. The document leaves me puzzled as to what exactly the point was in introducing it in the first place, because it appears to have almost nothing in terms of new concepts. There's nothing novel about using cable select to automatically select which drive in a dual-drive configuration is the master and which the slave. The other main requirement seems to be support for the "Identify Drive" command to allow autodetection of drive size and other parameters on the part of the system BIOS. But again, this is already supported by every IDE/ATA drive made since at least the mid-1990s.

Perhaps the main point was to exploit the buzzword potential of the phrase "plug and play", to help convince consumers that a great breakthrough in configuration ease had been made. I don't really know, but based on the lack of substance in the specification, that's my bet. At any rate, I have not seen any other hard disk manufacturers make reference to it, and it does not appear to be destined to play an important role in the market at this time. For more details, you may be able to find the original white paper on *Quantum's web site*.

If you run into a system that is advertised as using "Plug and Play ATA", this most likely just means that the drive(s) are set to the cable select method of configuration, and a cable select cable is being used.

Next: Making Sense of IDE/ATA Standards and Compatibility

Making Sense of IDE/ATA Standards and Compatibility

I hope that in my effort to reduce all the confusion surrounding real and unofficial IDE/ATA standards, that I have not made you even more confused. :^) There are a lot of buzzwords being tossed around, and the marketing people are hard at work introducing new ones every year. ;^)

Fortunately, the technology itself is pretty easy to use, even if the labels given to it often stink. So one useful way of dealing with all the standards and labels is simply to ignore them! Look past the hype, and focus on what the drive's actual capabilities are. If you want to really understand what a drive can do and what it supports, you should look at its specification sheet and see what features and transfer modes it is designed to use. Ignore labels like "EIDE" or "Ultra ATA/whatever" and find out what modes and functions the drive supports. Getting the real scoop on the drive means you don't need to worry about the pretty stickers slapped all over the box, or whatever the manufacturer is trying to claim.

It's also important to realize that despite all of the various names and flavors, to some extent IDE/ATA is IDE/ATA. Especially since the late 1990s, virtually all IDE/ATA drives and controllers will work together with a minimum of fuss. The folks who create the IDE/ATA standards always ensure backwards compatibility. This means that a drive corresponding to a newer standard will still work on an IDE channel in an older PC. Similarly, older drives will work on newer systems, in most cases. When older and newer hardware are mixed, the newer hardware will just run at whatever the maximum speed is of the older hardware.

For example, suppose you want to use an older hard disk that does not support Ultra DMA, on a newer PC with Ultra DMA support. This will work fine, but the drive won't run at Ultra DMA speed. As a second example, suppose you want to install a new hard drive that runs Ultra DMA/100 on an older PC with a hard disk controller only supporting Ultra DMA/33. This will also work, but the drive's throughput will be limited to 33 MB/s.

Note: There can be issues with using some newer drives on some older systems, if they come enabled to run at higher-speed Ultra DMA modes by default. A utility may be needed to change the default transfer mode of the drive. See here for more details. Similarly, using new drives on older systems may cause problems related to the larger sizes of new drives (which isn't an interface issue).

One issue that many people have when upgrading an older system with a newer hard disk is that the older system may not support the highest transfer rate supported by the drive (as in the second example I just gave). This often causes great consternation and anguish, because the hard disk manufacturers hype interface transfer speeds to the high heavens. In fact, running an Ultra DMA/100 drive on an Ultra DMA/66-capable controller will produce no noticeable difference in performance compared to running it on an "Ultra ATA/100 controller". Even a 33 MB/s "regular" Ultra ATA channel will not result in a huge performance hit. For a full explanation of the reasons why, see this page.

Next: IDE/ATA Transfer Modes and Protocols

IDE/ATA Transfer Modes and Protocols

Since performance is of utmost concern when using a hard disk, the different transfer modes and protocols that a drive (and interface) supports are very important. In fact, they get more attention than any other issues and features associated with the interface! Most of the advances in newer IDE/ATA standards are oriented around creating faster ways of moving data between the hard disk and the PC system. Since the IDE/ATA interface is in essence a communication channel, support for a given transfer mode or protocol requires support from the devices on both ends of the channel. This means that both the hard disk and the system chipset and BIOS must support the mode in question.

In this section I describe all of the transfer modes and protocols used for the IDE/ATA interface. First I discuss programmed I/O (PIO) modes, the oldest IDE/ATA transfer mode. I then talk about direct memory access (DMA) modes, and then the new Ultra DMA modes that superceded regular DMA and are the transfer modes of choice for newer hardware. I also talk about a a few features and protocols related to IDE data transfer.

Next: Programmed I/O (PIO) Modes

Programmed I/O (PIO) Modes

The oldest method of transferring data over the IDE/ATA interface is through the use of *programmed I/O*. This is a technique whereby the system CPU and support hardware directly control the transfer of data between the system and the hard disk. There are several different speeds of programmed I/O, which are of course called *programmed I/O modes*, or more commonly, *PIO modes*.

Through the mid-1990s, programmed I/O was the only way that most systems ever accessed IDE/ATA hard disks. Three lower-speed modes were defined as part of the original ATA standards document; two more were added as part of ATA-2 (as well as part of several unofficial standards.) The table below shows the five different PIO modes, along with the cycle time for each transfer and the corresponding throughput of the PIO mode:

PIO Mode	Cycle Time (nanoseconds)	Maximum Transfer Rate (MB/s)	Defining Standard
Mode 0	600	3.3	ATA
Mode 1	383	5.2	ATA
Mode 2	240	8.3	ATA
Mode 3	180	11.1	ATA-2
Mode 4	120	16.7	ATA-2

A few things about this table bear mention. First of all, the PIO modes are defined in terms of their cycle time, representing how many nanoseconds it takes for each transfer to occur. The maximum transfer rate is the reciprocal of the cycle time, doubled because the IDE/ATA interface is two bytes (16 bits) wide. (For a basic explanation of cycle time and related issues, check out this fundamentals section). Also, conspicuous by its absence from the table above is the so-called "PIO mode 5", which does not exist and was never implemented in any IDE/ATA hard disks. Apparently, at one point some discussion occurred about creating a faster PIO mode, which was tentatively called "PIO mode 5". This mode was to support a transfer rate of 22.2 MB/s, but it was never implemented (probably because the much faster 33 MB/s Ultra DMA mode 2 was on the horizon). Some motherboard manufacturers made a point of providing early support for this proposed mode in their BIOS setup programs, so you may occasionally see it mentioned.

Obviously, faster modes are better, because they mean a higher theoretical burst transfer rate over the interface. This transfer rate represents the external data transfer rate for the drive. Remember that this is the speed of the *interface* and not necessarily the sustained transfer rate of the drive itself, which is almost always slower (and should be). Of course today all new drives have sustained transfer rates well in excess of what even the fastest PIO mode can handle, which is one reason why PIO has fallen out of favor. Also worth mention is that very old systems using ISA hard disk controllers cannot use even PIO modes 3 or 4, because their transfer rate exceeds the capacity of the ISA bus!

As I mentioned, programmed I/O is performed by the system CPU; the system processor is responsible for executing the instructions that transfer the data to and from the drive, using special I/O locations. This technique works fine for slow devices like keyboards and modems, but for performance components like hard disks it causes performance issues. Not only does PIO involved a lot of wasteful overhead, the CPU is "distracted" from its ordinary work whenever a hard disk read or write is needed. This means that using PIO is ideally suited for lower-performance applications and single tasking. It also means that the more data the system must transfer, the more the CPU gets bogged down. As hard disk transfer rates continue to increase, the load on the CPU would have continued to grow. This is the other key reason why PIO modes are no longer used on new systems, having been replaced by DMA modes, and then later, Ultra DMA.

As discussed in detail here, each IDE channel supports the use of two devices, designated as master and slave. Modern systems allow the use of master and slave devices running at different PIO modes on the same channel; this is called independent device timing and is a function of the system chipset and BIOS. When this feature is not supported, both devices may be limited to the slower of the two devices' maximum PIO mode, but this hasn't been a big issue since the mid-to-late 1990s.

PIO modes do not require any special drivers under normal circumstances; support for them is built into the system BIOS. This universal support, along with their conceptual simplicity, is why they were traditionally the default way that most drives are used. Today, however, PIO is just not up to handling modern drives, which use Ultra DMA to keep the load on the CPU down and to allow access to Ultra DMA's much higher performance. Support for PIO modes is still universal on almost all systems and drives made since the mid-1990s, for backwards compatibility. It is used, for example, as a "last resort" when driver or software issues cause problems with Ultra DMA accesses.

Next: Direct Memory Access (DMA) Modes and Bus Mastering DMA

Direct Memory Access (DMA) Modes and Bus Mastering DMA

As described in the page describing programmed I/O, that method of transferring data between the hard disk and the rest of the system has a serious flaw: it requires a fair bit of overhead, as well as the care and attention of the system's CPU. Clearly, a better solution is to take the CPU out of the picture entirely, and have the hard disk and system memory communicate directly. *Direct memory access* or *DMA* is the generic term used to refer to a transfer protocol where a peripheral device transfers information directly to or from memory, without the system processor being required to perform the transaction. DMA has been used on the PC for years over the ISA bus, for devices like sound cards and the floppy disk interface. Conventional DMA uses regular DMA channels which are a standard system resource. DMA is discussed in full detail here.

Several different DMA modes have been defined for the IDE/ATA interface; they are grouped into two categories. The first set of modes are *single word* DMA modes. When these modes are used, each transfer moves just a single word of data (a word is the techie term for two bytes, and recall that the IDE/ATA interface is 16 bits wide). There are (or were!) three single word DMA modes, all defined in the original ATA standard:

DMA Mode	Cycle Time (nanoseconds)	Maximum Transfer Rate (MB/s)	Defining Standard
Single Word Mode 0	960	2.1	ATA
Single Word Mode 1	480	4.2	ATA
Single Word Mode 2	240	8.3	ATA

(As I discussed in the page on PIO, maximum transfer rate is double the reciprocal of the specific cycle time for each mode.) Obviously, these are not impressive transfer rate numbers by today's standards. Further, performing transfers of a single word at a time is horribly inefficient--each and every transfer requires overhead to set up the transfer. For that reason, single word DMA modes were quickly supplanted by *multiword* DMA modes. As the name implies, under these modes a "burst" of transfers occurs in rapid succession, one word after the other, saving the overhead of setting up a separate transfer for each word. Here are the multiword DMA transfer modes:

DMA Mode	Cycle Time (nanoseconds)	Maximum Transfer Rate (MB/s)	Defining Standard

Multiword Mode 0	480	4.2	ATA
Multiword Mode 1	150	13.3	ATA-2
Multiword Mode 2	120	16.7	ATA-2

Since multiword DMA transfers are more efficient, and also have higher maximum transfer rates, single word DMA modes were quickly abandoned after ATA-2 was widely adopted--they were actually removed from the ATA standards in ATA-3. So all DMA accesses today (including Ultra DMA) are actually multiword; the term "multiword" is now often assumed and no longer specifically mentioned.

Another important issue with DMA is that there are in fact two different ways of doing DMA transfers. Conventional DMA is what is called *third-party DMA*, which means that the DMA controllers on the motherboard coordinate the DMA transfers. (The "third party" is the DMA controller.) Unfortunately, these DMA controllers are old and very slow--they are basically unchanged since the earliest days of the PC. They are also pretty much tied to the old ISA bus, which was abandoned for hard disk interfaces for performance reasons. When multiword DMA modes 1 and 2 began to become popular, so did the use of the high-speed PCI bus for IDE/ATA controller cards. At that point, the old way of doing DMA transfers had to be changed.

Modern IDE/ATA hard disks use *first-party* DMA transfers. The term "first party" means that the peripheral device itself does the work of transferring data to and from memory, with no external DMA controller involved. This is also called bus mastering, because when such transfers are occurring the device becomes the "master of the bus". Bus mastering allows the hard disk and memory to work without relying on the old DMA controller built into the system, or needing any support from the CPU. It requires the use of the PCI bus--older buses like MCA also supported bus mastering but are no longer in common use. Bus-mastering DMA allows for the efficient transfer of data to and from the hard disk and system memory. Bus mastering DMA keeps *CPU utilization* low, which is the amount of work the CPU must do during a transfer.

Interestingly, despite the obvious advantages of bus mastering DMA, the use of bus-mastering multiword DMA mode 2 never really caught on. There are several reasons for this. The most important was the poor state of support for the technology for the first couple of years. Using PIO required no work and was very simple; DMA was not supported by the first version of Windows 95, so special drivers had to be used. Problems with implementing bus mastering DMA on systems in the 1996 to 1998 time frame were numerous: issues with buggy drivers, software the didn't work properly, CD-ROM drives that wouldn't work with the drivers, and so on. In the face of these problems, DMA didn't offer much incentive to make the switch. Sure, the lower CPU utilization was good, but since the maximum DMA mode's speed was the same as that of the highest PIO mode (16.7 MB/s) there wasn't a great perception that

DMA offered much of an advantage over PIO. Given little upside potential, many people (including this author) stayed away from using DMA, to avoid the compatibility and stability problems that sometimes resulted.

Bus mastering DMA finally came into its own when the industry moved on to Ultra DMA. Once Ultra DMA/33 doubled the interface transfer rate, DMA had an obvious speed advantage over PIO in addition to its other efficiency improvements. Support for DMA was also cleaned up and made native in Windows 9x, and most of the problems with the old drivers were eliminated. Today, the use of Ultra DMA is the standard in the industry. See here for details on the Ultra DMA modes.

Next: Ultra DMA (UDMA) Modes

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. (For a complete description of clocked data transfer and double transition clocking, see this fundamentals section.) 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 (see here for more on clocking.) 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.

I discuss more issues and considerations for implementing Ultra DMA in the section on configuration.

Next: 16-Bit and 32-Bit Access

16-Bit and 32-Bit Access

One of the options on some chipsets and BIOSes is so-called *32-bit access* or *32-bit transfers*. In fact, the IDE/ATA interface always does transfers 16 bits at a time, reflecting its name ("AT attachment"--the original AT used a 16-bit data bus and a 16-bit ISA I/O bus). For this reason, the name "32-bit" access or transfer is somewhat of a misnomer.

Since modern PCs use 32-bit I/O buses such as the PCI bus, doing 16-bit transfers is a waste of half of the potential bandwidth of the bus. Enabling 32-bit access in the BIOS (if available) causes the PCI hard disk interface controller to bundle together two 16-bit chunks of data from the drive into a 32-bit group, which is then transmitted to the processor or memory. This results in a small performance increase.

Note: Some BIOSes (or add-in controller cards) may automatically and permanently enable this feature, and therefore not bother to mention it in the BIOS setup program.

Note: It should be noted that this has nothing to do at all with the very similar sounding "32-bit disk access" and "32-bit file access" that are options within Windows 3.x. These have more to do with how Windows and its drivers function than anything to do with the hard disk itself.

Next: Block Mode

Block Mode

On some systems you will find an option in the system BIOS called *block mode*. Block mode is a performance enhancement that allows the grouping of multiple read or write commands over the IDE/ATA interface so that they can be handled on a single interrupt.

Interrupts are used to signal when data is ready to be transferred from the hard disk; each one, well, interrupts other work being done by the processor. Newer drives, when used with a supporting BIOS allow you to transfer as many as 16 or 32 sectors with a single interrupt. Since the processor is being interrupted much less frequently, performance is much improved, and more data is moving around with less command overhead, which is much more efficient than transferring data one sector at a time.

Note: Some systems can have trouble running disks in block mode, even if they are supposed to allow it. You may have better luck with the drive or system if it is disabled.

Next: IDE/ATA Configuration and Cabling

IDE/ATA Configuration and Cabling

In most respects, IDE/ATA devices are relatively easy to install and configure, especially if you are only using one, or maybe two. However--you knew it was coming--the issues get more involved when many devices are to be used, when older drives are being configured, when mixing hard disks and ATAPI devices like CD-ROMs, or when using advanced transfer protocols like bus mastering DMA. The rise to prominence of Ultra DMA in the last few years has also made configuration and cabling just a bit much more complex, particularly with the introduction of a new 80-conductor IDE/ATA interface cable.

This section discusses issues relating to how IDE/ATA devices are set up and configured. This includes a complete look at how IDE/ATA hard disks are set up and configured, a discussion of IDE/ATA channels and resources, descriptions of the interface signals and cables used, and a look at software driver issues. I discuss various issues and options related to configuring multiple devices; choosing a better IDE/ATA hard disk configuration can result in performance improvements for the system, something that many people don't realize. I also talk a bit about notebook hard disks.

For specific instructions on configuring IDE/ATA devices, check out this procedure; for directions on physically installing the drive, this one; and for instructions on connecting the drive to the motherboard, this one.

Next: IDE/ATA Controllers

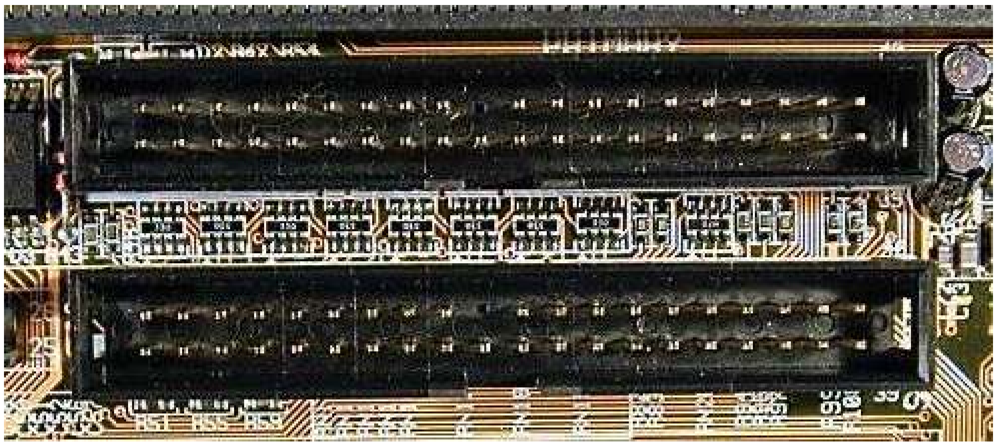
IDE/ATA Controllers

Every PC system that uses the IDE/ATA interface has at least one *IDE/ATA controller*. Now, as soon as you read that, a question probably formed in your mind: isn't the drive controller built into the drive in IDE, and in fact, wasn't that the whole point of how the name "IDE" came about? And you're absolutely right. Unfortunately, naming conventions in the PC world often leave much to be desired. A device that resides within the system and interfaces with a peripheral device is often commonly called a "controller", even though this isn't technically accurate. (As just one other example, the circuit on the motherboard that interfaces to the keyboard is called a "keyboard controller" even though keyboards also have their own built-in controllers.) At any rate, "a rose is a rose" and all that... :^)

So what exactly does this so-called IDE/ATA controller do, if not control the hard disk? Well, it acts as the middleman between the hard disk's internal controller and the rest of the system. As such, its less common name is the more accurate one: *IDE/ATA interface controller*. The controller (whatever its name) is what manages the flow of information over the IDE/ATA channels, allowing the hard disk to talk to the rest of the PC.

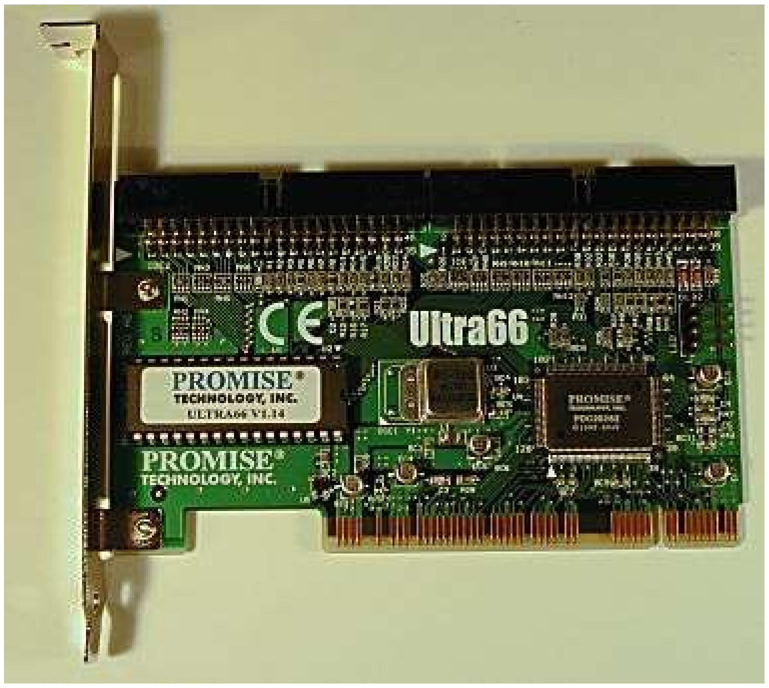
Traditionally--meaning: "back in the olden days" :^)--the IDE/ATA controller was a discrete interface card that plugged into a system expansion slot. The first IDE/ATA controllers were ISA bus cards. These were functional, and appropriate for technology of the late 1980s and early 1990s, but the ISA bus is very slow--throughput was limited to a maximum of about 8 MB/s. Other uncommon buses of that era such as EISA and MCA also had IDE/ATA controllers to suit, but those technologies never caught on. The creation of the VESA local bus meant a great improvement in performance for hard disks using controllers on that bus. In some cases, multi-function controller cards were used, providing both IDE/ATA interfacing, floppy drive interfaces and serial and parallel ports as well.

IDE/ATA controllers were transformed with the creation of the PCI bus in the mid-1990s, and Intel's decision to integrate the interface control functions into their new chipsets. Since that time, virtually every new motherboard has come with the interface controller built in, saving the cost of a separate controller card and also saving a PCI bus slot. The IDE/ATA cables simply connect to appropriate connectors on the motherboard; a much simpler arrangement.


A pair of IDE/ATA interface connectors on a typical motherboard. If you look closely, you can see the word "PRIMARY" above the top connector, identifying it as the connector for the primary IDE/ATA channel.

Of course, discrete PCI controllers continue to be made, and they are relatively inexpensive. These are commonly used for two main reasons: for expansion, if more IDE/ATA devices need to be used on a system, or to get around BIOS limitations of the built-in IDE/ATA controllers, enabling access to larger drives or faster transfer modes. In most cases, it is possible to use both the built-in and added-in controllers, as long as they are properly configured--or the built-in controllers can be disabled to free up system resources.

Tip: Another reason that sometimes add-in controllers are installed in a system is to provide support for IDE/ATA RAID. You can read more about RAID in general here.



An after-market, PCI-based IDE/ATA controller (the Promise Ultra66). Note the PCI interface connector on the bottom. The two IDE/ATA channel connectors on the top of the card are where IDE/ATA cables from the drives are attached to the card.

Some time in the next few years, even PCI will eventually become too limiting to handle the maximum throughput of the fastest IDE/ATA drives. The practical limit of the standard 32-bit, 33 MHz PCI bus is about 100 MB/s, and that's already the interface speed of current Ultra DMA/100 drives; maximum sustained transfer rates are about half that. It won't be long before regular PCI is just not up to the task. In fact, this is already happening with SCSI, because SCSI drives are faster and more than one can transfer data at a time. SCSI host adapters ("controllers") are now showing up using enhancements to the PCI bus such as 64-bit PCI, or PCI-X.

Note: Add-in ATA controllers may appear to the system as if they were actually SCSI controllers. This means that they may be listed as SCSI controllers in the Windows device manager. To enable booting from the device you may also have to set your BIOS's boot sequence to be SCSI first. See this page for more on configuration issues.

Next: IDE/ATA Channels and Resource Usage

IDE/ATA Channels and Resource Usage

The data pathway over which information flows in the IDE/ATA interface is called a *channel*. Each IDE channel is capable of communicating with up to two IDE/ATA devices (including ATAPI devices if they are supported by the BIOS). Despite Western Digital going so far as to define "dual IDE channels" as part of its enhanced IDE "standard", there has never really been anything barring the use of more than one IDE/ATA channel in a PC. It just wasn't regularly done prior to the mid 1990s. Before the popularity of ATAPI CD-ROMs and removable drives, and plentiful and cheap hard disk storage, the vast majority of PCs used one or two hard disks on a single IDE channel, and so a single channel was sufficient.

In fact, it is theoretically possible to configure and use as many as four (or even more) different IDE/ATA interface channels on a modern PC. There is nothing inherently different in concept between these channels, although there can be a difference in terms of how they are implemented. In theory, they are independent system devices, each using their own set of system resources. If configured correctly (so they don't try to use the same resources and therefore conflict), each IDE channel can behave basically independently.

This table shows the names of the four standard IDE channels, and the resources used by each under "classical" configuration:

Channel	IRQ Used	I/O Addresses Used	Popularity and Support
Primary	14	1F0-1F7h and 3F6-3F7h	Used by all PCs using IDE/ATA
Secondary	15 (10)	170-177h and 376-377h	Present on all modern PCs; usually used for an ATAPI optical drive
Tertiary	11 (12)	1E8-1EFh and 3EE-3EFh	Used uncommonly, can have software support problems
Quaternary	10 or 11	168-16Fh and 36E-36Fh	Very rarely used, can have software support problems

Note: The I/O address range for the slave device on the primary IDE channel actually overlaps with the standard address range for the floppy disk controller. This is in fact not a conflict since this overlap is well known and accounted for.

As you can see, each IDE/ATA channel has traditionally required both an interrupt request (IRQ) line and two ranges of I/O addresses. The first two IDE channels are pretty much standard among all newer PCs, as are the resources that they use. IRQs 14 and 15 are generally reserved on most systems for use by the primary and secondary IDE channels, and most newer

PCI motherboards have support for both of these channels built into the chipset and BIOS, so there are two IDE connectors on the motherboard, one for the primary channel and one for the secondary. Most operating systems and other software "know" about these two channels, and software problems or resource conflicts with them are rare. With the exception of SCSI host adapters, it was hard to even *find* an expansion card that will use IRQ 14, until recently. (SCSI host adapters can replace IDE entirely in a PC so having them be able to use IRQ 14 can make some sense. Similarly, add-in IDE/ATA controllers can use IRQ 14 or 15.)

While all modern systems have the primary and secondary IDE controllers built into the motherboard, some older systems didn't implement both channels identically. The better systems included full transfer mode support and bus mastering for both the primary and secondary channels, but some systems--especially early Pentiums from the mid-1990s--wimped out. In order to save a few bucks, they included support for the faster PIO modes (3 and 4) only on the primary channel, meaning that the secondary channel would only run at the lower PIO modes (0, 1 and 2). The idea was that the primary channel would be used by the main hard disks (fast) and the secondary channel by extra, older hard disks and ATAPI devices (slow). Really, having full support on both channels is a much smarter idea, and this practice has fortunately been abandoned.

The tertiary and quaternary channels have always been far less frequently used, and software issues with them are far more frequent. There are also resource issues to be addressed; the IRQs used by the third and fourth channels can also be "claimed" by other peripherals such as sound cards, network cards and even PS/2 style mice; see this section on system configuration. The I/O addresses they use can conflict with network cards, COM ports, and other devices.

Using the tertiary and quaternary channels requires that additional controllers be added to the two built into the motherboard (or provided by the existing controller or controllers). Through the mid-to-late 1990s, the most common way that a tertiary channel was introduced into a PC was through the use of a sound card. Many SoundBlaster and compatible sound cards included support for an IDE/ATA controller that could be configured to implement an IDE/ATA channel. There are two reasons for this: first, sound cards were commonly sold in "multimedia kits" that include ATAPI CD-ROM drives, and so this provides a place for them to be attached. Second, the very first CD-ROMs were attached to proprietary (non-IDE) ports on early sound cards, so the trend has continued for historical reasons even into the ATAPI CD-ROM era.(See here for more on CD-ROM interfacing issues.)

The use of sound cards for IDE/ATA channels causes resource issues, as mentioned, and just generally isn't the best way to connect a high-speed device to the system. For starters, many sound cards use the ISA bus, meaning the IDE/ATA controller is also on the slow ISA bus. This practice has diminished in recent years, though if you have a sound card and want to run a slower ATAPI device on it, you can certainly do so. Bear in mind that software support for the tertiary and quaternary IDE channels is not nearly as consistent as it is for the primary and secondary channels. For example, Windows NT 4.0 will not recognize an IDE device attached to a tertiary IDE channel on a sound card. Most BIOSes also only have space to set up four IDE/ATA/ATAPI devices. This means that using a tertiary IDE requires some sort of driver or add-in BIOS support.

Over the last few years, the way that channels are added to a PC has changed greatly. I said earlier that each channel "traditionally" required separate resources, but cards like Promise's "Ultra" series support two IDE/ATA channels while using just a single IRQ line. Further, modern systems running Windows 95 OEM SR2 or later support the use of PCI IRQ Steering, which allows several PCI devices to share an interrupt. The combination of these two has caused the whole issue of system resources for IDE/ATA channels to be much improved, which is why my discussion of tertiary and quaternary channels above is largely in the past tense. :^)

Today, the best way of providing support for up to eight IDE/ATA devices (four channels) is through the addition of one of these PCI-based cards. Doing so allows you the use of four channels, while taking only three IRQs (two by the original motherboard channels and one for the card). Support for the card is provided through a driver provided by the maker of the card. As an additional bonus, these cards usually support the latest Ultra DMA transfer modes, and support for large hard disks as well. If you have an older system, the best way to go is to put the hard disks on the add-in card to get support for high-speed transfers, and then use the motherboard's channels for slower ATAPI devices. And as if that weren't enough, these cards cost under $50!

Tip: If you are tight on IRQs in a system, you can add a Promise-type controller card and disable both of the on-board channels. This leaves you with support for four devices as before, but only one IRQ is required for the add-in controller. This is a net gain of one IRQ!

Next: Single, Master and Slave Drives and Jumpering

Single, Master and Slave Drives and Jumpering

Each IDE/ATA channel can support either one or two devices. IDE/ATA devices of course each contain their own integrated controllers, and so in order to maintain order on the channel, it is necessary to have some way of differentiating between the two devices. This is done by giving each device a designation as either *master* or *slave*, and then having the controller address commands and data to either one or the other. The drive that is the target of the command responds to it, and the other one ignores the command, remaining silent.

Note that despite the hierarchical-sounding names of "master" and "slave", the master drive does not have any special status compared to the slave one; they are really equals in most respects. The slave drive doesn't rely on the master drive for its operation or anything like that, despite the names (which are poorly-chosen--in the standards the master is usually just "drive 0" and the slave "drive 1"). The only practical difference between master and slave is that the PC considers the master "first" and the slave "second" in general terms. For example, DOS/Windows will assign drive letters to the master drive before the slave drive. If you have a master and slave on the primary IDE channel and each has only one regular, primary partition, the master will be "C:" and the slave "D:". This means that the master drive (on the primary channel) is the one that is booted, and not the slave.

Devices are designated as master or slave using jumpers, small connectors that fit over pairs of pins to program the drive through hardware. Each manufacturer uses a different combination of jumpers for specifying whether its drive is master or slave on the channel, though they are all similar. Some manufacturers put this information right on the top label of the drive itself, while many do not; it sometimes takes some hunting around to find where the jumper pins are on the drive even once you know how the jumpers are supposed to go. The manufacturers are better about this now than they have been in the past, and jumpering information is always available in the manual of the hard disk, or by checking the manufacturer's web site and searching for the model number. I describe (and illustrate) the jumpers on IDE/ATA disks in detail in the section on hard disk construction. For a fundamental description of what jumpers are, see here.

ATAPI devices such as optical, Zip and tape drives are jumpered in pretty much the same way as hard disks. They have the advantage of often having their jumpers much more clearly labeled than their hard disk counterparts. Most optical drives, for example, have three jumper blocks at the back, labeled "MA" (master), "SL" (slave) or "CS" (cable select).

If you are using two drives on a channel, it is important to ensure that they are jumpered correctly. Making both drives the master, or both the slave, will likely result in a very confused system. Note that in terms of configuration, it makes no difference which connector on the standard IDE cable is used in a standard IDE setup, because it is the jumpers that control master and slave, not the cable. This does not apply when cable select is being used, however. Also, there can be electrical signaling issues if one connects a single drive to only the *middle* connector on a cable, leaving the end connector unattached. In particular, the use of Ultra DMA is not supported in such a configuration; see the discussion of the 80-conductor Ultra DMA cable for more information.

As long as one drive is jumpered as master and the other as slave, any two IDE/ATA/ATAPI devices *should* work together on a single channel. Unfortunately, some older hard disks will fail to work properly when they are placed on a channel with another manufacturer's disk. One of the reasons why drives don't always "play nicely together" has to do with the *Drive Active / Signal Present* (*/DASP*) signal. This is an IDE/ATA interface signal carried on pin #39, which is used for two functions: indicating that a drive is active (during operation), and also indicating that a slave drive is present on the channel (at startup). Some early drives don't handle this signal properly, a residue of poor adherence to ATA standards many years ago. If an older slave drive won't work with a newer master, see if your master drive has an "SP" (slave present) jumper, and if so, enable it. This may allow the slave drive to be detected.

Drive compatibility problems can be extremely frustrating, and beyond the suggestion above, there usually is no solution, other than separating the drives onto different channels. Sometimes brand X won't work as a slave when brand Y is the master, but X will work as a master when Y is the slave! Modern drives adhere to the formal ATA standards and so as time goes on and more of these older "problem" drives fall out of the market, making all of this less and less of a concern. Any hard disk bought in the last five years should work just fine with any other of the same vintage or newer.

When using only a single drive on a channel, there are some considerations to be aware of. Some hard disks have only a jumper for master or slave; when the drive is being used solo on a channel it should be set to master. Other manufacturers, notably Western Digital, actually have three settings for their drives: master, slave, and *single*. The last setting is intended for use when the drive is alone on the channel. This type of disk should be set to single, and not master, when being used alone.

Also, a single device on an IDE channel "officially" should not be jumpered as a slave. In practice, this will often work despite being formally "illegal". Many ATAPI drives come jumpered by default as slave--because they are often made slaves to a hard disk's master on the primary IDE channel, this saves setup time. However, for performance reasons they are sometimes put on the secondary channel, and often the system assemblers don't bother to change the jumpers. It will work, but I don't recommend it; if nothing more, it's confusing to find a slave with no master when you or someone else goes back into the box a year or two later to upgrade.

For performance reasons, it is better to avoid mixing slower and faster devices on the same channel. If you are going to share a channel between a hard disk and an ATAPI device, it is generally a good idea to make the hard disk the master. In some situations there can be problems slaving a hard disk to an optical drive; it will usually work but it is non-standard, and since there is no advantage to making the ATAPI device the master, the configuration is best avoided.

There are many more performance considerations to take into account when deciding how to jumper your IDE devices, if you are using several different ones on more than one channel. Since only one of the master and slave can use any channel at a time, there are sometimes advantages to using more than one IDE/ATA channel even if not strictly necessary based on the number of devices you are trying to support. There can also be issues with using a drive that has support for a fast transfer mode like Ultra DMA with older devices that don't support these faster modes. See this section for a discussion of performance issues and IDE/ATA configuration.

Next: Configuration Using Cable Select

Configuration Using Cable Select

An alternative to the standard master/slave jumpering system used in the vast majority of PCs is the use of the *cable select* system. As the name implies, with this system the cable--or more correctly, which connector on the cable a device is attached to--determines which device is master and which is slave. The goal of cable select is to eliminate having to set master and slave jumpers, allowing simpler configuration.

To use cable select, both devices on the channel are set to the "cable select" (CS) setting, usually by a special jumper. Then, a special cable is used. This cable is very similar in most respects to the regular IDE/ATA cable, except for the *CSEL* signal. CSEL is carried on wire #28 of the standard IDE/ATA cable, and is grounded at the host's connector (the one that attaches to the motherboard or controller). On a cable select cable, one of the connectors (the "master connector") has pin #28 connected through to the cable, but the other (the "slave connector") *has an open circuit* on that pin (no connection). When both drives on the channel are set cable select, here's what happens:

- **Master:** The device that is attached to the "master connector" sees the CSEL signal as grounded, because its connector has pin #28 attached to the cable, and the host's connector has that signal grounded. Seeing the "zero value" (grounded), the device sets itself to operate as master (device 0).
- **Slave:** The drive that is attached to the "slave connector" does not see the CSEL signal as grounded, because its connector is not attached to the CSEL signal on the cable. Seeing this "no connection", the device configures itself as a slave (device 1).

If you switch the devices between the two connectors, they swap configuration, the master becoming the slave and vice-versa. Not a very complicated arrangement, and a good idea, it would seem. In fact, *if cable select had actually caught on*, it would have been great. The problem is that it has never been widely used, and this lack of universality has made cable select unattractive, which is a bit of a chicken and egg situation. Since cable select was never accepted in the industry, most drives come, by default, with the drive jumpered as a master or single drive. This means that to enable cable select, you have to change a jumper anyway, which obviously negates some of the advantage.

But the biggest reason why cable select never caught on was the cable itself. From the very beginning, all 40-conductor IDE/ATA cables *should* have been made so that they would work with cable select. There's actually no need to have different cable types, because if you set a drive to "master" or "slave" explicitly, it just ignores the CSEL setting. So a cable select cable can be used either way: regular jumpering or cable select.

Unfortunately, regular 40-conductor IDE/ATA cables don't support cable select. (Why this came about I do not know, but I suspect that some bean counter determined they could save five cents on each PC by doing this.) So to use cable select you need a special cable, and these are of course non-standard, making them a special purchase. Also, many people don't understand cable select, nor do they realize it needs a special cable. If you set both drives to "CS" and then use them on a regular (non-cable-select) IDE cable, both drives will configure themselves as "master", causing a configuration conflict.

Making matters worse, the 40-conductor IDE/ATA cable select cables have the "master connector" as the *middle* device and the "slave connector" as the device at the end of the cable, farthest from the host. For signaling reasons, it's best to put a single drive at the end of a cable, not put it in the middle leaving a "stub" of wire hanging off the end of the channel. But if you do this, that single drive sets itself as a slave with no master, a technically illegal configuration. Worse, suppose you do this, and your hard disk sets itself as a slave, and the system boots from it without problem, as most would. Then, you decide to add a new hard disk. You set it to cable select and attach it to the middle connector. The new drive then becomes the master, and thus moves ahead of the old drive in precedence! The system will try to boot from it instead of your old drive (which some people might want, but many do not.)

To get around this problem, a *second* type of 40-wire cable select cable was created, the so-called "Y-shaped" cable. On this one, the connector to the system is in the middle, and the slave and master connectors are on the two opposite ends of the cable. This certainly makes things less confusing, but has its own difficulties. For starters, IDE/ATA cables are very limited in length, which means this "Y-shaped" cable was hard to use in large tower systems. All your drives had to be mounted very close to the motherboard or controller card so the cable would reach. And again, the cable was a special item.

As you can see, the traditional way of doing cable select was a total mess, which was why it was never widely adopted. The key reason for this mess was--once again--lack of standardization. I rather expected cable select to eventually wither away. However, when the 80-conductor Ultra DMA cable was introduced, the cable select feature was *much* improved, changing the potential of this feature. The two key changes were:

- **Drive Position:** Unlike the old cables, with the 80-conductor cable, the master connector is at the end of the cable, and the slave is in the middle. As I explained above, this is a much more sensible arrangement, since a single drive placed at the end of the cable will be a master, and a second drive added in the middle a slave.
- **Universality:** All 80-conductor IDE/ATA cables support cable select (or at least, all of the ones that are built to meet the ATA standards). This means there's no confusion over what cables support the feature, and no need for strange "Y-cables" and other non-standard solutions.

These two changes mean a world of difference for the future of cable select. Since these cables will eventually completely replace all of the 40-conductor cables, all systems will be capable of running cable select without any special hardware being needed. As I mentioned before, you can still explicitly set drives to master or slave if you want to, and the CSEL signal will be ignored by the drives. So the bottom line is that these cables work either way, cable select or not. What will finally make cable select catch on? If drive manufacturers and systems integrators widely agree to use it, and the manufacturers start shipping drives with the "CS" jumpers on by default. We'll have to see if this happens.

Warning: 80-conductor IDE/ATA cables are often said to be compatible with 40-conductor cables. That's true of *normal* 40-conductor cables with drives jumpered as master and slave, but not cable select cables. If you swap a regular (non-"Y-shaped") 40-conductor cable select cable with an 80-conductor IDE cable, the master and slave drives will swap logical positions. If you don't that to happen, you'll need to change the order that the devices connect to the cable.

Note: A special thanks to Hale Landis of *www.ata-atapi.com* for his assistance in deciphering the mysteries of cable select, especially with the 80-conductor cable.

Next: IDE/ATA Connectors and Signals

IDE/ATA Connectors and Signals

Standard IDE/ATA hard disks and ATAPI devices use two different connectors. The first is the *data connector*, to which the IDE/ATA cable attaches. The second is the *power connector*, which comes from the power supply, and of course, provides power to the drive. The power connectors are standardized and discussed in more detail in this construction page, and this power supply page. The data connectors and signals I will describe below.

Let's begin with the signals themselves. There are 40 wires in a regular IDE/ATA cable, so it's no surprise that there are 40 corresponding signals. (Incidentally, the newer 80-conductor cable uses the same pins and signals. For compatibility, and because the 40 extra conductors that were added are just grounds, the pin assignments are the same.) The table below lists the names of the signals, along with the pin number of the standard connector that each uses:

Pin #	Signal	Pin #	Signal
1	-RESET	2	GROUND
3	DD7	4	DD8
5	DD6	6	DD9
7	DD5	8	DD10
9	DD4	10	DD11
11	DD3	12	DD12
13	DD2	14	DD13
15	DD1	16	DD14
17	DD0	18	DD15

19	GROUND	**20**	(key)
21	DMARQ	**22**	GROUND
23	-DIOW: STOP	**24**	GROUND
25	DIOR:-HDMARDY:HSTROBE	**26**	GROUND
27	IORDY:-DDMARDY:DSTROBE	**28**	CSEL
29	-DMACK	**30**	GROUND
31	INTRQ	**32**	(reserved)
33	DA1	**34**	-PDIAG:-CBLID
35	DA0	**36**	DA2
37	-CS0	**38**	-CS1
39	-DASP	**40**	GROUND

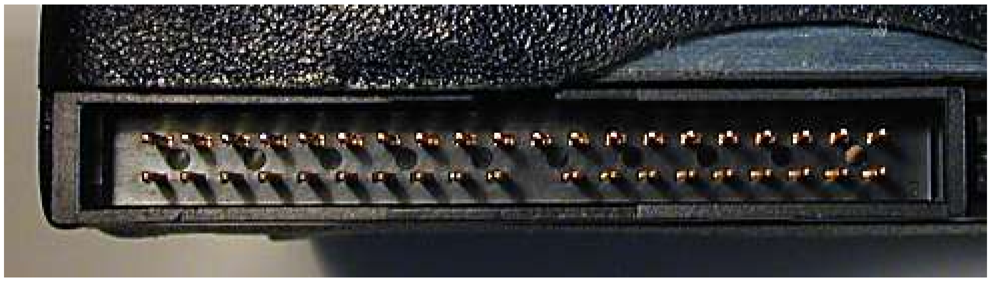
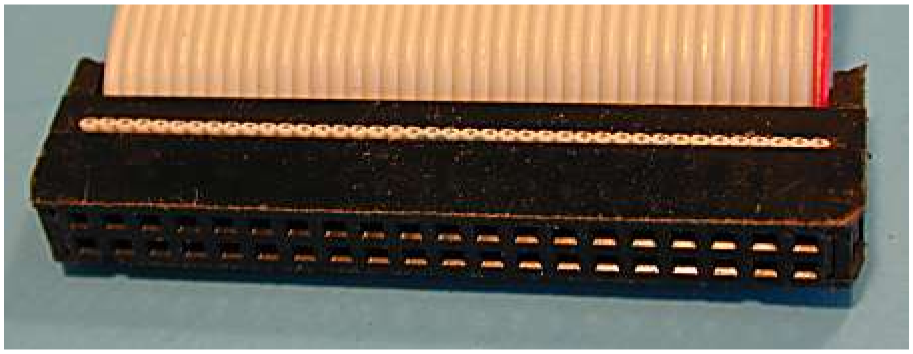
(For a general description of what signals are, and what the "dash" ("-XXX") notation means, see this page.)

Now, I'm not going to describe all of these signals in detail; if you are interested in learning all about them, you should order the latest IDE/ATA standard and read up all about the signaling. However, I do have a few explanatory notes:

- **Pins 3 through 18:** These are the 16 data lines used for transferring data over the interface.
- **Pin 20:** This is a "key" location, used for orientation; see below for more.
- **Pin 28:** This is the cable select signal used for cable select operation.
- **Pin 32:** This was once known as "/IOCS16" but is not currently used.
- **Pin 34:** This pin is used (in part) to detect the presence of an 80-conductor IDE/ATA cable for Ultra DMA operation; see here for more.

The data connectors for IDE/ATA are standardized. Drives and hosts (controllers) have male connectors consisting of two rows of 20 pins, with a plastic "fence" surrounding them. Cables have female connectors with two rows of 20 holes for the pins. There are two ways that these connectors are supposed to be keyed for proper orientation (to prevent the cable from being inserted into the drive or controller upside-down). On the male connectors, pin #20 is supposed to be missing, and on the cable, the hole for pin #20 is supposed to be blocked. Also, the female (cable) connectors are supposed to have a tab in the middle on top that matches a gap in the plastic surrounding the male pins. If you tried to put a cable in upside-down, these keyings would prevent insertion.

Unfortunately, yet again, these measures were never standardized. Some drives and controllers were produced that had pin #20 in place, even if it was not used, and some did not have the "gap" in the plastic surrounding the pins. If you used a properly-designed cable with an improperly-constructed drive or controller, the cable wouldn't fit. To avoid this, many IDE/ATA cable makers just said "to heck with it" and made the cables with no plastic tab on the connectors, and no block for hole #20. As a result, the entire orientation scheme fell apart, so one must be careful to line up pin #1 on the cable with pin #1 on the drive. There is usually a red stripe on the edge of the cable on the side where pin #1 is, but it's still easy to get the cable backwards. Fortunately, the standard data connector has no live power signals, so damage is not typical if the cable is inserted upside-down (though the drive won't work that way, of course!)

An IDE/ATA interface connector on a hard disk (above) and on a regular 40-conductor IDE/ATA cable (below). Note the keying features on the hard disk connector: pin #20 is missing, and there's a gap in the plastic surrounding the pins (top middle). However, these features are not matched on the cable: hole #20 has not been plugged, and there's no notch on the top of the connector. This is a common situation, which is why IDE/ATA cables can be and often are inserted improperly, unless you are careful to install them the right way. You must look for the red stripe on the cable (see it?) that marks pin #1, and then find which is pin #1 on the hard disk drive or motherboard connector.

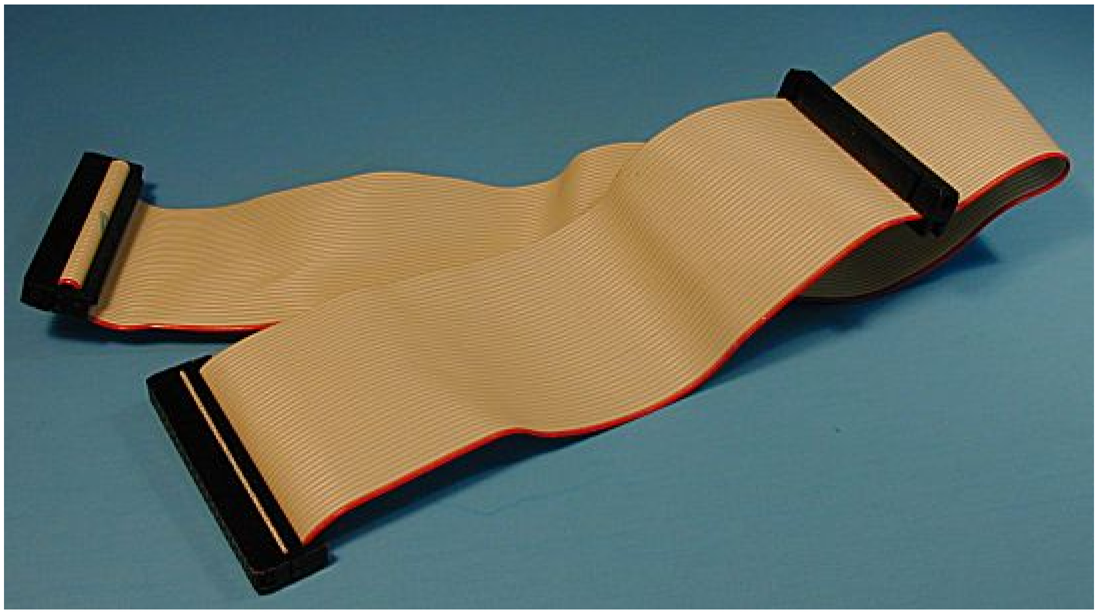
Note: The 2.5" form factor drives used in notebook PCs use different connectors and are attached differently. See this page for details.

Next: Standard (40-Conductor) IDE/ATA Cables

Standard (40-Conductor) IDE/ATA Cables

Each IDE/ATA channel uses one IDE/ATA cable. The cable that has been used for over a decade on this interface was once just called "an IDE cable", since there was only one kind (with the exception of special cable select cables.) Today, however, there is also the new 80-conductor Ultra DMA cable; to avoid confusion, I refer to the old cable as a *standard 40-conductor cable*.

A standard IDE cable is a rather simple affair: a flat ribbon cable, normally gray in color, with a (usually red) stripe running down the edge. The cable has 40 wire connectors in it, and usually has three identical female connectors: one is intended for the IDE controller (or motherboard header for PCs with built in PCI ATA controllers) and the other two are for the master and slave devices on the interface. The stripe is used to line up pin 1 on the controller (or motherboard) with pin 1 on the devices being connected, since the techniques used for keying the cables are not standardized. For more information on IDE/ATA connectors and signals, see this page.


A standard, 40-wire IDE/ATA cable. Note the presence of three black connectors, and the 40 individual wires in the ribbon cable. (Go ahead, get that magnifying glass out and count 'em. :^)) Also note the red wire that marks wire #1 and hence pin #1 on each connector.

Some cheapskate PC makers that ship PCs with only one IDE device per channel save a few pennies by using an interface cable that has only two connectors. This means you can't use two IDE devices on a channel, unless you replace the cable with one that has three connectors. Fortunately this is easy to do, and the cables are cheap and readily available at most any store that sells computer supplies and parts (this is also a great component to buy cheap at a computer show). Even more fortunately, most companies don't do this any more.

In many ways, the cable is the weak link in the IDE/ATA interface. It was originally designed for very slow hard disks that transferred less than 5 MB/s, not the high-speed devices of today. Flat ribbon cables have no insulation or protection from electromagnetic interference. Of course, these are reasons why the 80-conductor cable was developed for Ultra DMA. However, even with slower transfer modes there are limitations on how the cable can be used.

The main issue is the length of the cable. The longer the cable, the more the chance of data corruption due to interference on the cable and uneven signal propagation, and therefore, it is often recommended that the cable be kept as short as possible. According to the ATA standards, the official maximum length is 18 inches, but if you suspect problems with your hard disk you may find that a shorter cable will eliminate them. Sometimes moving where the disks are physically installed in the system case will let you use a shorter cable.

Warning: There are companies that sell 24" and even 36" IDE cables. They are not recommended because they can lead to data corruption and other problems. Many people use these with success, but many people do a *lot* of things they shouldn't and get away with it. :^)

In terms of its mechanics, the IDE cable could certainly be much better designed as well. As described here, a keying mechanism exists to prevent incorrect cable insertion, but it's not universally implemented. This means there is the risk of inserting the cable backwards--the red stripe on the cable should be used to align pin 1 of the IDE/ATA device with pin 1 of the controller's connector port. Also, the cable has no latching mechanism, so it is not very securely attached to the hard disk or the motherboard. If you work inside the box and apply any pressure on the cable by accident, it can easily come loose (sometimes only partially) which will lead to device failure. Fortunately, neither backwards insertion or a partially loose cable usually cause any permanent damage--your hard disk just won't work.

Assuming cable select is not being used, any connector on a standard 40-conductor cable can go to any device, because all 40 wires are connected "straight through" to all three connectors. Since two of the connectors are closer to each other than the third, the distant connector is normally attached to the motherboard (or hard disk controller card). The other two devices can be used for either the master or the slave, and it doesn't matter which is which. If a single device is used, it should be attached to the connector at the end of the cable, and the connector in the middle of the cable left unattached. Using the middle connector and leaving the end connector unattached is technically allowed for regular PIO and DMA transfer modes, but leaves part of the cable "dangling". This is called a *stub* and creates much worse electrical characteristics on the cable, due to reflections from the unterminated ends of the cable wires. It is not recommended.

Tip: The new 80-conductor cable is compatible with regular 40-conductor cables (not cable select cables). If you are having integrity problems with an older system using a 40-conductor cable, try replacing it with one of the newer cables, which are superior electrically to the older design. Many people now use the 80-conductor cables exclusively on all systems, even ones not using Ultra DMA.

Next: Ultra DMA (80-Conductor) IDE/ATA Cables

Ultra DMA (80-Conductor) IDE/ATA Cables

There are a lot of issues and problems associated with the original 40-conductor IDE cable, due to its very old and not very robust design. Unterminated flat ribbon cables have never been all that great in terms of signal quality and dealing with reflections from the end of the cable. The warts of the old design were tolerable while signaling speeds on the IDE/ATA interface were relatively low, but as the speed of the interface continued to increase, the limitations of the cable were finally too great to be ignored.

In the ATA/ATAPI-4 standard that introduced the Ultra DMA transfer mode set, a new cable was introduced to replace the old standby: the *80-conductor IDE/ATA cable*. The name is important: the new cable has 80 *conductors* (wires)--it does *not* have 80 pins on each connector, though, just 40. This means that the new cable is pin-compatible with the old drive. No change has been made to the IDE/ATA connectors, aside from the color-coding issue (see below).

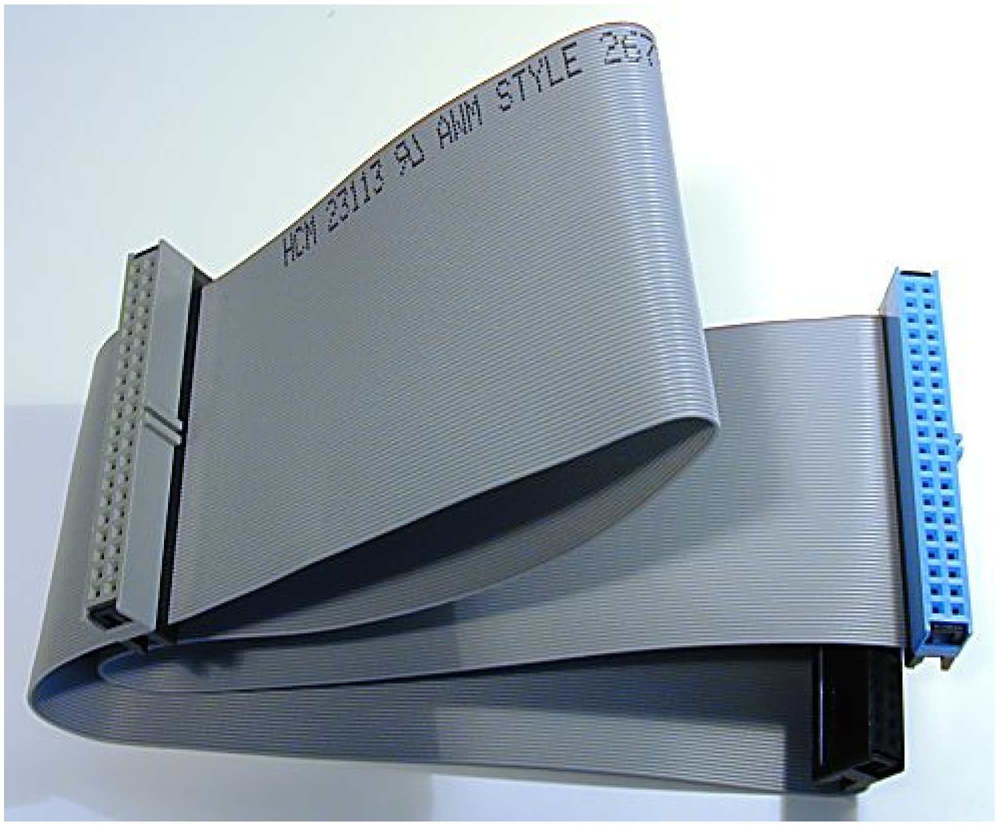
The obvious question, of course, is this: what's the point of adding 40 extra wires to a cable if they aren't connected to anything? :^) Well for starters, the 40 wires *are* connected to something, just not their own pins on the interface connectors. The extra 40 wires don't carry new information, they are just used to separate the "real" 40 signal wires, to reduce interference and other signaling problems associated with higher-speed transfers. So the 40 extra conductors are connected to ground, interspersed between the original 40 conductors of the old cable. Any stray signals that would "cross-talk" between adjacent wires on the 40-conductor cable are "absorbed" by these extra ground wires, improving signal integrity. The extra ground wires can be either all of the even-numbered wires, or all of the odd-numbered wires in the cable.

There are a number of other attributes and characteristics of the 80-conductor cable, which I'm going to list in bullet form for easier absorption:

- **Requirement:** The 80-conductor cable was first defined with the original Ultra DMA modes 0, 1 and 2, covering transfer speeds up to 33.3 MB/s. The cable is considered "optional" for those modes. However, for any Ultra DMA modes above mode 2, the 80-conductor cable is *mandatory*.
- **Detection:** Since the cable is mandatory for high-speed modes, the system has to have some way of knowing it is installed. This is done by having the /PDIAG:/CBLID signal, carried on pin #34 of the interface, grounded in the connector that attaches to the motherboard. Since the older 40-conductor cable would not have this pin grounded, by looking for the grounding on this pin at startup the host can determine if the 80-conductor cable is installed.

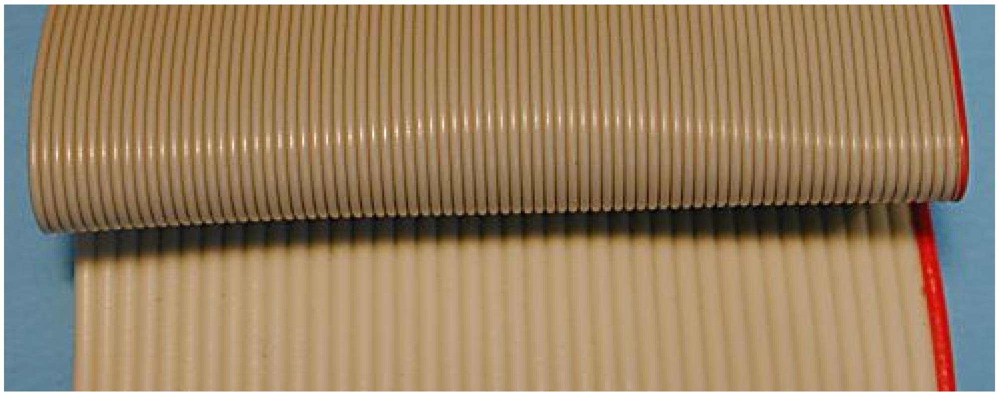
- **Cable Select Support and Drive Assignment:** All 80-conductor cables that meet the ATA specifications support the cable select feature automatically. This is accomplished by special connection of the CSEL signal on pin #28. The cable can still be used with drives that have been manually configured as master or slave, of course. See the discussion of cable select for more.
- **Connector Assignments and Color Coding:** For the first time, the 80-conductor cable defines specific roles for each of the connectors on the cable; the older cable did not. Color coding of the connectors is used to make it easier to determine which connector goes with each device:
    - **Blue:** The blue connector attaches to the host (motherboard or controller).
    - **Gray:** The gray connector is in the middle of the cable, and goes to any slave (device 1) drive if present on the channel.
    - **Black:** The black connector is at the opposite end from the host connector and goes to the master drive (device 0), or a single drive if only one is used.

  There are a couple of reasons why this coding was done. The main one is the issue mentioned in the discussion of the 40-conductor cable: it is not a good idea to connect a single drive to the middle connector on a ribbon cable, because the "stub" of left-over, unconnected cable causes signaling problems. With Ultra DMA this "stub" connection is not just "not recommended", it is illegal: a single device must be at the end of the cable. The other reason is that since these cables support cable select inherently, the position of each drive on the cable matters if cable select is being used. With these two needs combined, it just made sense to design the cable so that drive positioning was explicitly clear.



A standard 80-conductor Ultra DMA IDE/ATA interface cable. Note the blue, gray and black connectors, and the 80 thin wires. The red marking on wire #1 is still present (but hard to see in this photo.)

- **Width:** Despite the extra 40 wires, the 80-conductor cable is about the same width as a 40-conductor cable--which is good, because the current width is difficult enough to work with. :^) This bit of "magic" is accomplished by using thinner, lower-gauge wires within the cable.



A comparison of the wires used in 80-conductor and 40-conductor cables. The 80-conductor cable is about the same width as the older style because thinner gauge wires are used to make up the ribbon.

Aside from the above, the cable can be treated the same way as a 40-conductor cable. Since it is of higher quality, it can be used in place of a 40-conductor cable in older systems without any issues. However, it does *not* directly replace a 40-conductor cable select cable. Note also that the 18" length restriction associated with the original 40-conductor cable applies to this variation as well.

Next: Notebook IDE/ATA Configuration

Notebook IDE/ATA Configuration

Notebook PCs are very similar to desktop PCs in terms of overall architecture, but very different in implementation. Hard disks and hard disk interfaces are certainly no exception to this general rule of thumb. Virtually all notebooks have built-in hard disks using the IDE/ATA interface, but the way they are actually connected the machine is not the same as for desktop PCs.

Hard disks are installed in notebooks in two very different ways--which method is used on a particular model depends on the decisions made by the engineers that created it. The first is a *proprietary* installation, where the hard disk is just rammed "in there somewhere"; the second is an *open* or *removable* installation. The differences between the two are pretty obvious. If a notebook has a proprietary hard disk installation, it probably still uses the regular IDE/ATA interface, but the drive is attached using special cables and connectors, and is not intended to be touched by the user. Needless to say, this is a very inflexible arrangement, because if you ever need or want to replace or upgrade the drive, you have to refer to qualified service personnel for service. This was the most common way of putting hard disks into notebooks during their early years.

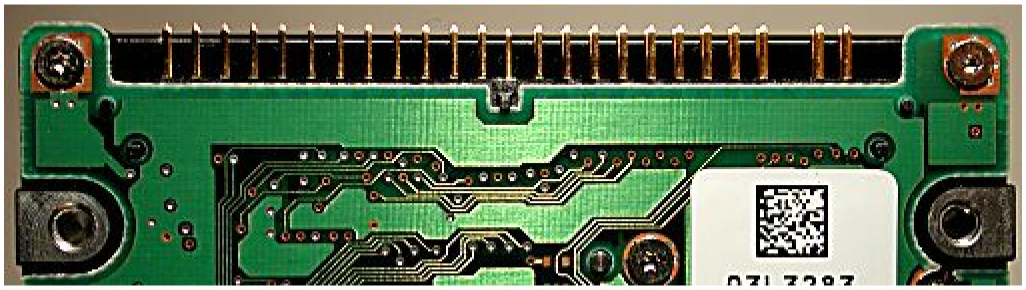
Most modern hard disks today use the open arrangement; unlike the proprietary installations these are probably *easier* to install and remove than the hard disks in a regular PC! A typical hard disk will use a special hard disk bay that has room for a standard, 2.5" form factor hard disk. Bays differ between models, especially in terms of the height of the drive, but the form factor itself is pretty much standardized at this point; you can read more about it here. The advantages of a standard interface are obvious: interchangeability and competition (allowing you choices between drive makers).

The usual method of attachment of notebook hard disks is through a special 44-pin connector that includes all of the signals needed by the drive, instead of the usual 40-pin data connector and 4-pin power connector. This provides the basis for allowing the drives to be quickly and easily removed and replaced. The drive itself uses a regular set of straight pin connectors, just like a desktop drive. Since these pin connectors are not well-suited for easy insertion and removal from a notebook, the drive is mounted into a special carrier or "caddy". This device includes as part of its hardware an adapter which converts the regular pins into a single connector designed for drive swapping. The technique used for this attachment is very similar in concept to the SCA connectors used for some types of SCSI drives (though the two are obviously very different in all but this concept.)

The first 40 signals on a notebook's connector are the same as those of the regular 40-pin connector; the additional four signals are defined as follows:

Pin #	Signal	Pin #	Signal
41	+5 V (logic)	**42**	+5 V (motor)
43	GROUND	**44**	(reserved)

You may immediately notice that there is no +12 V connection as exists for regular drives, because 2.5" form factor drives have 5-volt motors. Two separate +5 lines are provided; one for the motor and the other for the hard disk's circuit board.


Underside of a 2.5" form factor notebook hard drive. You can see the main connector, with its two rows of 22 pins (the second row is hard to see). On the right are two more pins which are used for jumpers. To see what this drive looks like in a caddy, complete with its single attachment connector, see this page

Notebooks are of course very limited in space, so "expansion" is usually not an option, at least not using the built-in IDE/ATA interface. The hard disk usually is assigned as the single device on the primary IDE/ATA channel, and the drive's optical drive (if any) assigned to the secondary. There's no way to decide to add a second device to these channels on a typical notebook. Expanding a notebook to add a second hard disk is usually done using one of the specialty interfaces such as USB or PCMCIA.

Next: Independent Master/Slave Device Timing

Independent Master/Slave Device Timing

Since the transfer modes associated with the IDE/ATA interface are constantly being improved, new devices support faster transfer modes than older ones do. In addition, hard disks often support faster transfer modes than ATAPI devices such as optical drives do. Yet, these devices can be combined on the same IDE/ATA channel, raising the question of compatibility when the devices are together.

The ability of an IDE/ATA channel to operate a master and slave device using different transfer modes is called *independent device timing*. The hard disk controllers integrated on modern chipsets all pretty much support independent timing, as do modern add-in controllers, but this was not always the case. Independent timing can be an issue if, for example, you upgrade an older PC and get a new, high-speed drive, but want to continue to be able to use the older one on the same channel with the new one.

If your system does not support independent device timing, and you use a newer hard disk that supports PIO mode 4 on the same channel as an older hard disk that operates only at PIO mode 0, the system will knock down the PIO mode to 0 for *both drives*. This will hamper the performance of the newer hard disk. The lack of independent device timing on many older systems is one reason why placing ATAPI devices like CD-ROMs on the same channel as a fast hard disk is usually not recommended. (It should be pointed out that we are talking here about the interface or *external* transfer speed of these devices. Reducing the speed of the interface only causes big issues if the effective interface speed becomes less than the sustained transfer rate of the disk. For a full discussion of these issues, please see this page.)

Again, today's chipsets all pretty much support independent timing, so this is less of an issue than it once was. However, there are still other good reasons to be careful about how you assign drives to the IDE/ATA channels in your system; see this page for a full discussion of these issues.

Note: It is not possible to use PIO modes to control one device on a channel and DMA (or Ultra DMA) modes to control the other one. Do not mix devices that don't support DMA with ones that do on the same channel, if you want to use DMA.

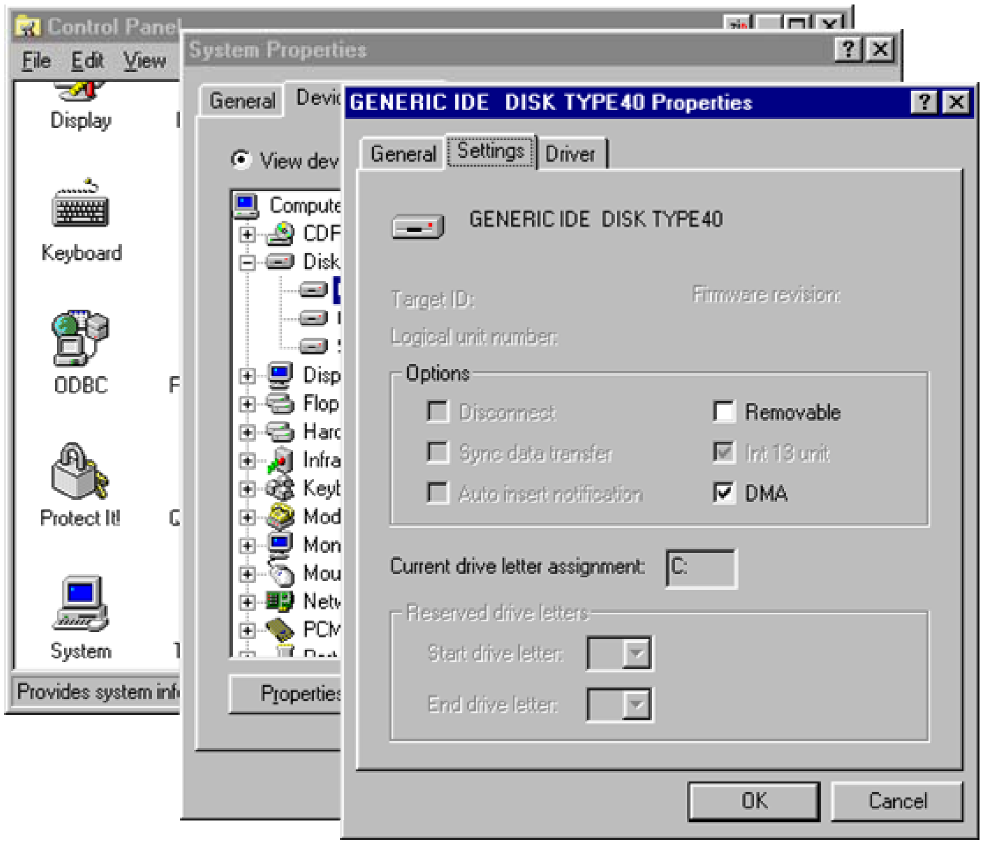
Next: Windows Drivers and DMA Support

Windows Drivers and DMA Support

Traditionally, support for IDE/ATA hard disks has been provided by the system BIOS and operating system. This support was at first limited the simple programmed I/O (PIO) access modes. Over time, the IDE/ATA world has moved away from PIO towards DMA modes. At first this was because DMA modes allow hard disk transfers to occur without excessive *CPU utilization*--PIO modes drag down the CPU while DMA modes allow transfers to occur while letting the CPU do other things. With the introduction of Ultra DMA, DMA support became that much more important.

DMA began to rise in popularity at around the time the first version of Windows 95 was released. Unfortunately, Windows 95 "A" did not have native support for first-party bus mastering DMA, so special drivers had to be added to support DMA. These frequently caused compatibility and stability issues, and it took several years for DMA support to "mature". Starting with the second OEM-only release of Windows 95 ("OEM SR2", "Windows 95b") DMA support was provided by the operating system. Windows 98 and Windows ME also provide built-in support for DMA operation, and DMA works quite well today.

On modern systems, DMA support should be enabled by default. You can check for DMA support by looking in the Device Manager. From the Control Panel, open "System", then click the "Device Manager" tab. Open the icon for "Disk drives", and then highlight the drive you are interested in. Click "Properties" and then the "Settings" tab. Among the other settings you should find a "DMA" check box, which should be checked. If it is not, try checking it to enable DMA support. If you then reboot and the check box does not stay checked, this probably means your system does not support Ultra DMA; try looking on this page to be sure you meet all the requirements.



GENERIC IDE DISK TYPE40

Target ID:
Firmware revision:
Logical unit number:

Options
[ ] Disconnect
[ ] Removable
[ ] Sync data transfer
[✓] Int 13 unit
[ ] Auto insert notification
[✓] DMA

Current drive letter assignment: C:

Reserved drive letters
Start drive letter:
End drive letter:

The "Settings" page for a DMA-capable hard disk under Windows 95, accessed through the Control Panel. Note the DMA checkbox.

Note: Hard drives connected to some Ultra DMA add-in controllers will not display a "DMA checkbox". Since these cards are designed primarily to *provide* Ultra DMA support, they don't bother showing it as an explicit option.

Next: Performance Factors and Tradeoffs in Configuring for Multiple Devices

Performance Factors and Tradeoffs in Configuring for Multiple Devices

Configuring a single IDE/ATA or ATAPI device is very simple. You pretty much just need to jumper it, and then connect it with an IDE/ATA cable, and you are usually done. Setting up more than one device is really not much more complicated. Except for some special cases, you just jumper pairs as master and slave (or use cable select) and use both the primary and secondary channels to get support for up to four devices. You can also add additional IDE/ATA channels to get support for up to eight devices, although these involve a bit more work and slightly more risk of resource issues or other complications.

While setting up multiple devices isn't difficult, there are real performance reasons why it makes sense to put some thought into how you decide to allocate different drives to different IDE channels, and which to make master and which slave. As time goes on, the trend in more and more systems is towards more and more drives. As more removable drives are implemented using ATAPI, and as PC users add new hard disks to their systems in greater numbers, many PC users are finding that they have to figure out how to arrange all these drives in a way that makes sense. A common occurrence is a PC user who starts out with a new PC that has just a hard disk and a CD-ROM drive. Over a couple of years, it's easy to add to this a CD-RW drive for backups or making music, and a new hard disk to improve performance or expand capacity. Suddenly, whammo, you're looking at four IDE/ATA devices, and probably only the original two IDE/ATA channels. Some are surprised to find themselves juggling five or even *six* IDE/ATA drives in a single PC!

The following are some of the issues that you should take into account when configuring multiple IDE/ATA devices, to maximize the performance of your system:

- **Master/Slave Channel Sharing:** By its very nature, each IDE/ATA channel can only deal with one request, to one device, at a time. You cannot even begin a second request, even to a different drive, until the first request is completed. This means that if you put two devices on the same channel, they must share it. In practical terms, this means that any time one device is in use, the other must remain silent. In contrast, two disks on two different IDE/ATA channels can process requests simultaneously on most motherboards. The bottom line is that the best way to configure multiple devices is to make each of them a single drive on its own channel, if this is possible. (This restriction is one major disadvantage of IDE compared to SCSI). An add-in controller like the Promise "Ultra" series is a cheap way of adding extra IDE/ATA channels to a modern PC.
- **Boot Devices and Boot Order:** The boot device of the PC is usually the first hard disk that the operating system "sees" when it boots up. By default, this means the master drive on the primary IDE/ATA channel. If you want to boot from a drive on an add-in controller, you may actually need to set up your BIOS boot sequence to boot from SCSI first, so the add-in controller is seen before the ones on the motherboard. (These controllers often "look" like SCSI controllers to the system.) This is only an issue if you have hard disks on both the

existing and the add-in controller; if you have hard disks on the add-in controller but not on the built-in motherboard controllers, this won't be necessary.

- **Independent Master/Slave Device Timing:** Hard disk controllers on modern systems support running the master and slave device at different speeds, if one supports faster transfer modes than the other. Some systems, however, especially older ones, do not. If you are using two devices with radically different maximum transfer rates, and the chipset doesn't support independent timing, you will slow down the faster device to the speed of the slower one.
- **Hard Disk and ATAPI Device Channel Sharing:** There are several reasons why optical drives (or other ATAPI devices) should not be shared on the same channel as a fast hard disk. ATAPI allows the use of the same physical channels as IDE/ATA, but it is not the same protocol; ATAPI uses a much more complicated command structure. Opticals are also generally much slower devices than hard disks, so they can slow a hard disk down when sharing a channel. Finally, some ATAPI devices cannot deal with DMA bus mastering drivers, and will cause a problem if you try to enable bus mastering for a hard disk on a channel they are using.
- **Older Hard Disks:** Older hard disks, typically those three years in age or more, typically only support the older, low-speed PIO modes for transfers (check the drive's specifications to be sure). In this case, the caveats about hard disk performance matter much less, since the drive is relatively slow anyway.
- **Interface Bus Type:** High-speed transfer modes require the use of a local bus (PCI or VLB) hard disk controller. ISA-based controllers, such as regular sound cards that are sometimes used as tertiary IDE channels, are not suitable for use with high-speed modern disks.
- **IRQ Resource Usage:** Each conventional IDE channel requires an IRQ line, and in some PCs these are a scarce commodity. In some cases devices can be combined on a single channel with minimal impact on performance, allowing the reclamation of one or more IRQs for use by other peripherals. Add-in controllers can also provide you with two channels on a single IRQ; see here for more.

For information on how to use these factors to choose configurations for different combinations of IDE/ATA and ATAPI devices, see this page.

Next: Recommended IDE Device Configurations

Recommended IDE Device Configurations

Based on the IDE/ATA configuration performance factors outlined in this section, I have identified what I think are sensible configuration options for setting up IDE channels for different hard disk and ATAPI device combinations. These recommendations are oriented towards maximizing overall system performance, and are based primarily on my experience and understanding of the IDE/ATA interface and the way most people use their systems.

The following is a list of assumptions and notes regarding these recommended configurations:

- Systems vary more today in terms of how many IDE/ATA channels they have, and of what type, than ever before. I cannot list every possible configuration, and I certainly am not going to bother with the ones that obviously make no sense.
- The best setup depends to a great extent on the needs of the individual, so use the recommendations on this page as *guidelines only*. Do what works for you.
- If at all possible, the best configuration is always to have each device on a separate channel. Of course, not all channels are created equal. Hard disks and faster ATAPI devices should always be on a channel connected to the PCI bus (or VLB for 486-class systems). Older or slower ATAPI devices like slower CD-ROMs or tape drives can be connected to an ISA-bus tertiary channel such as the one on a sound card, though this is still usually not optimal.
- I am assuming that the channels on the motherboard, and any channels on an add-in controller, have the same support for transfer modes and hard disk size. If an add-in card is installed in an older system to provide support for faster transfer modes not supported by the integrated controllers, or to get around a hard disk size barrier, then newer, larger drives should be connected to those channels, and the built in controllers used for older devices. Note that booting from a drive connected to an add-in card may require a change to the BIOS boot sequence setting. See here for more.
- Some people are most concerned with absolute performance, and are willing to use extra controllers to get it, even if it costs them extra IRQs. Other people don't have IRQs to burn and may not even want to use the secondary IDE controller. I assume that most people will not disable the secondary IDE/ATA channel just to save an IRQ, though this is an option if you have only two devices.
- Finally, remember that choosing an ideal drive configuration is really an optimization; in most cases the differences between various alternatives are not enormous.

These are my recommendations for the more common mixtures of up to five hard disks and/or ATAPI (optical, tape, removable storage) drives:

Hard Disks	**ATAPI Devices**	**Notes**
1	**0**	This is fairly uncommon today, since almost all systems have at least one optical drive. The best configuration is to use the primary master for the hard disk and disable the secondary controller (to save one IRQ).
1	**1**	The most common default configuration. Unless system resources are very tight, put the hard disk as a single drive on the master channel, and the ATAPI drive on the secondary channel.

2	**1**	The best configuration is to put each device on a separate channel through the use of a third IDE/ATA channel. If only two channels are available, it is generally best to put the fastest drive as a single device on the primary channel, the second hard disk as the master on the secondary channel, and the ATAPI as the slave on the secondary.
1	**2**	This is a common configuration when a second ATAPI device is added to a new system; the best configuration depends on what that device is. In general, put the hard disk by itself and share the secondary channel between the two ATAPI drives. However, if your two drives are a CD-ROM and a CD-RW drive, and you are doing a lot of copying from the CD-ROM drive to the CD-RW, you may have better luck separating those devices onto separate channels.
2	**2**	For optimal performance, buy an add-in controller card and use all four channels for the four devices. If you have three channels, then put the boot drive by itself on the primary channel, and then split the remaining three devices on the secondary and tertiary channels as described in the row just above. If you have only two channels, there are several options. I would put the fastest drive as the boot device on the primary master, and whatever drive is the least used as the slave on that channel. Then put the other two devices on the secondary channel. Avoid putting drives that can't be run in Ultra DMA with the boot device.
3	**1**	Fairly similar configuration to the two rows above. Use four channels if possible. If there are three put the two slowest devices together. If there are only two channels, configure as for the 2+2 option above.
1	**3**	Same as for 3+1 just above.
3	**2**	If you have more than four devices, you will need to add a controller to get a third IDE/ATA channel--or get rid of one of your devices. ;^) The best option is to add a PCI-based IDE/ATA controller card, which will give you four channels. If this is the case, put the three fastest or more used devices on their own channels, and the two slowest devices together. If you have three channels, put the boot drive by itself and split the other four drives up over the remaining two channels as described in the discussion for the 2+2 case.
2	**3**	Same as for 3+2 just above.

Next: Small Computer Systems Interface (SCSI)

Small Computer Systems Interface (SCSI)

The second-most popular hard disk interface used in PCs today is the *Small Computer Systems Interface*, abbreviated *SCSI* and pronounced "skuzzy". SCSI is a much more advanced interface than its chief competitor, IDE/ATA, and has several advantages over IDE that make it preferable for many situations, usually in higher-end machines. It is far less commonly used than IDE/ATA due to its higher cost and the fact that its advantages are not useful for the typical home or business desktop user.

In terms of standards, SCSI suffers from the same problem that IDE/ATA does: there are too many different ones and it can be hard to understand what is what. Fortunately, this situation is coming under control now. Also, SCSI standards aren't as much of a problem as they are for IDE/ATA, because in the SCSI world, each SCSI protocol has a name that indicates rather clearly what its capabilities are, and there is much less reliance on using the name of the standard to infer transfer rates and other characteristics. Unfortunately, there is still a lot of confusion if you try to figure out the standards themselves and what each one means. And there are still manufacturers playing fast and loose with how they label their drives.

SCSI is a much higher-level protocol than IDE is. In fact, while IDE is an interface, SCSI is really a system-level bus, with intelligent controllers on each SCSI device working together to manage the flow of information on the channel. SCSI supports many different types of devices, and is not at all tied to hard disks the way IDE/ATA is--ATAPI supports non-hard-disk IDE devices but it is really a kludge of sorts. Since it has been designed from the ground up as almost an additional bus for peripherals, SCSI offers performance, expandability and compatibility unmatched by any other current PC interface.

In this section of the site I describe the SCSI interface in detail. I begin with an overview of the interface and a brief history. I then discuss SCSI standards, focusing on the ones of most relevance to today's PC SCSI user. I then examine the various protocols and features of the interface, followed by the popular transfer modes and feature sets used. I then discuss practical implementation matters, including SCSI host adapters, cable and connector types, and configuration issues.

Note: Despite the fact that IDE stands for "integrated drive electronics", and that IDE and SCSI are "competing" interfaces, SCSI devices all have integrated drive controllers. IDE is really a misnomer for the IDE/ATA interface.

Note: Most of the attention in this section is oriented towards "regular" parallel SCSI, as commonly used on PCs. The SCSI-3 standard also defines several other "cousins" of parallel SCSI; these are mentioned briefly, but parallel SCSI is the focus, since it continues to be the most widely implemented.

Next: Overview and History of the SCSI Interface

Overview and History of the SCSI Interface

What we currently know of as the SCSI interface had its beginnings back in 1979. Shugart Associates, led by storage industry pioneer Alan Shugart (who was a leader in the development of the floppy disk, and later founded Seagate Technology) created the *Shugart Associates Systems Interface* (*SASI*). This very early predecessor of SCSI was very rudimentary in terms of its capabilities, supporting only a limited set of commands compared to even fairly early "true" SCSI, and rather slow signaling speeds of 1.5 Mbytes/second. For its time, SASI was a great idea, since it was the first attempt to define an intelligent storage interface for small computers. The limitations must be considered in light of the era: we are talking about a time when 8" floppy drives were still being commonly used. :^)

Shugart wanted to get SASI made into an ANSI standard, presumably to make it more widely-accepted in the industry. In 1981, Shugart Associates teamed up with NCR Corporation, and convinced ANSI to set up a committee to standardize the interface. In 1982, the X3T9.2 technical committee was formed to work on standardizing SASI. A number of changes were made to the interface to widen the command set and improve performance. The name was also changed to SCSI; I don't know the official reason for this, but I suspect that having Shugart Associates' name on the interface would have implied that it was proprietary and not an industry standard. The first "true" SCSI interface standard was published in 1986, and evolutionary changes to the interface have been occurring since that time. (You can read more about SCSI standards here.)

It's important to remember that SCSI is, at its heart, a *system* interface, as the name suggests. It was first developed for hard disks, is still used most for hard disks, and is often compared to IDE/ATA, which is also used primarily for hard disks. For those reasons, SCSI is sometimes thought of as a hard disk interface. (I must admit that placing my SCSI coverage in my own hard disk interfaces section certainly suggests this as well!) However, SCSI is not an interface tied specifically to hard disks. Any type of device can be present on the bus, and the very design of SCSI means that these are "peers" of sorts--though the host adapter is sort of a "first among equals". :^) My point is that SCSI was designed from the ground up to be a high-level, expandable, high-performance interface. For this reason, it is frequently the choice of high-end computer users. It includes many commands and special features, and also supports the highest-performance storage devices.

Of course, these features don't come for free. Most PC systems do not provide native, "built in" support for SCSI the way they do for IDE/ATA, which is one of the key reasons why SCSI isn't nearly as common as IDE/ATA in the PC world. Implementing SCSI on a PC typically involves the purchase of a storage device of course, but also a special card called a host adapter. Special cables and terminators may also be required. All of this means that deciding between SCSI and IDE/ATA is an exercise in tradeoffs.

SCSI began as a parallel interface, allowing the connection of devices to a PC or other systems with data being transmitted across multiple data lines. Today, parallel or "regular" SCSI is still the focus of most SCSI users, especially in the PC world. SCSI itself, however, has been broadened greatly in terms of its scope, and now includes a wide variety of related technologies and standards, as defined in the SCSI-3 standard.

Next: SCSI Standards

SCSI Standards

There was a time that SCSI standards were relatively few, and not that difficult to understand. That time is now long past. :^) In some ways, the best way I could describe the current situation regarding SCSI standards, feature sets and marketing terms is that it makes the standards and terms associated with IDE/ATA seem simple by comparison. That would really be a rather strong indictment, however, so I won't say that. ;^) Still, understanding all of the documents and labels associated with SCSI can be very baffling at times.

It's not that the standards are poorly written, or that the technology is all that hard to understand. The main issue with SCSI today is that it has become so broad, and includes so many different protocols and methods, that it's hard to get a handle on all of it. The confusion surrounding SCSI standards has increased since the creation of SCSI-3, which is really a collection of different standards, some of them rather different from each other. The situation is made worse by manufacturers that like to create funky new "unofficial names" for transfer modes or feature sets, or apply overly-broad labels to specific hardware.

As described in the page describing the history of SCSI, the first organization that was charged with developing the first SCSI standard was ANSI technical committee X3T9.2. Today, SCSI standards are developed, maintained and approved by a number of related organizations, each playing a particular role. Here's how they all fit together:

- **American National Standards Institute:** *ANSI* is usually thought of as an organization that develops and maintains standards, but in fact they do neither. They are an oversight and accrediting organization that facilitates and manages the standards development process. As such, they are the "high level management" of the standards world. They qualify other organizations as *Standards Developing Organizations* or *SDOs*. ANSI also publishes standards once they have been developed and approved.
- **Information Technology Industry Council:** *ITIC* is a group of several dozen companies in the information technology (computer) industry. ITIC is the SDO approved by ANSI to develop and process standards related to many computer-related topics.
- **National Committee for Information Technology:** *NCITS* is a committee established by ITIC to develop and maintain standards related to the information technology world. NCITS was formerly

known under the name "Accredited Standards Committee X3, Information Technology", or more commonly, just "X3". It maintains several sub-committees that develop and maintain standards for various technical subjects.

- **T10 Technical Committee:** *T10* is the actual technical standards committee responsible for the SCSI interface.

Note: If this description looks similar to the one on the page where I defined the structure of the organizations supporting the T13 technical committee that develops ATA standards, that's because it is. T10 and T13 are sibling committees.

If you boil all of this down, T10 is the group that actually does the work of developing new SCSI standards. ;^) The other organizations support their activities. The T10 group is comprised primarily of technical people from various hard disk and other technology companies, but the group (and the development process itself) is open to all interested parties. Comments and opinions on standards under development are welcomed from anyone, not just T10 members. The standards development process is intended to create consensus, to ensure that everyone who will be developing hardware and software agrees on how to implement new technology.

Once the T10 group is done with a particular version of a standard, they submit it to NCITS and ANSI for approval. This approval process can take some time; which is why the official standards are usually published several years after the technology they describe is actually implemented. While approval of the standard is underway, companies develop products using technology described in the standard, confident that agreement has already been reached. Meanwhile, the T10 group starts work on the next version of the standard. With SCSI-3 now including a number of different "sub-standards" (hmm, bad name :^)), it is in some ways constantly "under development".

There are also other organizations that are involved in the creation and maintenance of SCSI-related standards. Since SCSI-3 has a broad scope, it defines and structures certain standards that are in fact "owned" by other groups. In particular, the documents describing the physical layer for Fibre Channel are developed by the *T11 technical committee*, and the IEEE-1394 interface is of course an *IEEE standard*.

In this section I describe the three main standards that define SCSI. They are listed in chronological order, and SCSI-3 is expanded into its own full section, reflecting its new status as an "umbrella" standard containing several others.

Note: Standards that have been approved and published by ANSI are available for purchase in either print form or electronic format from *ANSI's web site*. Draft standards that are under development (as well as older drafts of approved standards) can be found at the *T10 Technical Committee web site*.

Tip: If there's a SCSI term or "standard" that you are looking for information on but can't find in this section, it might in fact be a transfer mode or feature set.

Warning: You may occasionally see a hardware device being sold based on the name of a standard; for example, a "SCSI-3 drive". Be aware that this is a meaningless label, because it is very vague. With the possible exception of SCSI-1, the standards define several different transfer speeds and signaling methods, so just giving the name of a standard is insufficient information to properly describe a SCSI device. With SCSI-3 especially, the label could mean just about anything--always ask for specifics.

Next: SCSI-1

SCSI-1

SCSI evolved from the Shugart Associates Systems Interface or SASI, which was originally created in 1979. The first SCSI standard was approved by ANSI in 1986 as standard X3.131-1986. To avoid confusion when subsequent SCSI standards came out, the original specification was later renamed "SCSI-1".

SCSI-1 defines the basics of the first SCSI buses, including cable length, signaling characteristics, commands and transfer modes. It was quite limited, especially by today's standards, and defined only the most fundamental of SCSI features and transfer modes. Devices corresponding to the SCSI-1 standard use only a narrow (8-bit) bus, with a 5 MB/s maximum transfer rate. Only single-ended transmission was supported, with passive termination. There were also difficulties associated with the standard gaining universal acceptance, due to the fact that many manufacturers implemented different subsets of its features. The standard did not call for all devices to implement support for the same commands, so there was no guarantee that any given device would work with any other!

SCSI-1 is now obsolete, and the standard has in fact been withdrawn by ANSI. Devices that adhere to the SCSI-1 standard can in most cases be used with host adapters and other devices that use the higher transfer rates of the more advanced SCSI-2 protocols, but they will still function at their original slow speed.

Note: Since all SCSI-1 devices are single-ended, they may cause performance degradation if placed onto a multimode LVD SCSI bus. If you want to run LVD devices to their full potential, you will want to avoid mixing them with single-ended devices.

Next: SCSI-2

SCSI-2

In 1985, a year before the SCSI-1 standard was formally approved, work began on the SCSI-2 specification. Important goals of this evolution of the SCSI standard were to improve performance, enhance reliability, and add features to the interface. However, the most important objective was to formalize and properly standardize SCSI commands. After the confusion that arose from the non-standardized implementations of original SCSI, a working paper was created to define a set of standard commands for SCSI hard disks, called the *common command set* or *CCS*. This paper eventually formed the basis for the new SCSI-2 standard. SCSI-2 was approved by ANSI in 1994 and released as document X3.131-1994.

Note: The SCSI-2 standard was originally released in 1990 as X3.131-1990, but it was retracted for further changes and didn't actually get formally approved until four years later. You may see reference to the 1990 version of the standard on occasion; there are actually few differences between it and the 1994 version.

SCSI-2 is an extensive enhancement of the very limited original SCSI. The command set used for SCSI devices was standardized and enhanced, and several confusing "options" removed. In addition, the standard defines the following significant new features as additions to the original SCSI-1 specification:

- **Fast SCSI:** This higher-speed transfer protocol doubles the speed of the bus to 10 MHz, meaning 10 MB/s transfer rate with 8-bit regular SCSI cabling or even higher when used with Wide SCSI.
- **Wide SCSI:** The width of the original SCSI bus was increased to 16 (or even 32) bits. This permits more data throughput at a given signaling speed. Wide SCSI eventually replaced original "narrow" SCSI buses for the fastest drives.
- **More Devices per Bus:** On buses that are running with Wide SCSI, 16 devices are supported (as opposed to 8 with regular SCSI).
- **Improved Cables and Connectors:** As discussed in detail here, SCSI uses a large number of different cable and connectors. SCSI-2 defined new higher-density connections, extending the basic 50-pin connectors defined in SCSI-1.
- **Active Termination:** Termination is an important technical consideration in setting up a SCSI bus. SCSI-2 defined the use of active termination, which provides more reliable termination of the bus.
- **Differential Signaling:** To allow longer cable lengths, differential signaling was introduced. (This was later renamed "high-voltage differential" to distinguish it from low voltage differential (LVD) signaling.)
- **Command Queuing:** One of SCSI's strengths is its ability to allow multiple outstanding requests between devices on the bus, simultaneously. Command queuing was introduced in SCSI-2.
- **Additional Command Sets:** SCSI-2 added new command sets to support the use of more devices such as CD-ROMs, scanners and removable media. The older command set focused more on hard disks.

There were also several other minor changes to the standard, mostly low-level technical changes that I don't really need to get into. It is important to note that one of the major design criteria in the creation of SCSI-2 was backward compatibility with SCSI-1. SCSI-2 devices will in most cases work with older SCSI-1 devices on a bus. This is not always done, however, because the older devices have no ability to support the SCSI-2 enhancements and faster transfer protocols.

Note: SCSI-2 is not the same as Ultra2 SCSI, which is a much newer and higher-performance feature set.

Next: SCSI-3

SCSI-3

Work on the next version of the SCSI standard, called *SCSI-3* of course, began in 1993. At the time, a large number of different technologies, command sets and features were being considered for SCSI. At the time SCSI-3 work started, the SCSI-2 standard was eight years old and had not yet been formally released; SCSI-2 was also a *much* bigger document than SCSI-1 had been. Considering how much more technology was vying for inclusion in SCSI-3, and how many different parties would have been involved in defining it, trying to do everything in one large document as had been done with SCSI-1 and SCSI-2 would have been a bad idea. The standard would have been much too long, would have required too many people to work on it, and development of hardware would have been held up during discussions of the standard. Standards that take too long to develop, or that are too cumbersome, get ignored by impatient companies that start developing their own proprietary extensions. If this happens, it causes a lot of confusion in the industry.

Recognizing the potential for problems here, the decision was made to make SCSI-3 not one huge standards document, but rather a collection of different, but related standards. Splitting up SCSI into these different sub-documents has allowed for a "divide and conquer" strategy that enables multiple standards to be worked on at once by different groups. In addition, it lets popular technologies advance at a faster rate than ones where changes are needed less frequently. Keeping all the standards together under one "banner" helps to ensure that the commands and other common attributes used between technologies remain synchronized.

Unfortunately, SCSI-3 tries to bite off a lot; some would say, more than it can chew. Within this umbrella standard are over a dozen other standards that define command sets, protocols and signaling methods related to SCSI and "SCSI-like" interfaces such as IEEE-1394 and Fibre Channel. Each of these documents has its own standards name and is revised independently of the others. For example, the most implemented form of SCSI, that which was formerly known as just "SCSI" in the earlier standards, became the *SCSI-3 Parallel Interface* (*SPI*) under SCSI-3. There are now several versions of SPI, each defining new features and transfer speeds for conventional, parallel SCSI devices. (In some cases, other organizations are involved in maintaining the documents, such as the *T11 technical committee* for Fibre Channel.) The large number of different technologies that all fall under the name "SCSI-3" has caused many people to be confused by that term. And since all of these constituent standards documents are constantly changing, it's unclear when (or even if) the overall "SCSI-3" standard will be formally approved!

In this section I discuss SCSI-3 in a fair bit of detail. I start by describing the basic architecture of the standard, and then describe very briefly the various standards that make up SCSI-3. I then focus most of my attention on several revisions of the *SPI* parallel specification, because these are the standards used by PC SCSI hard disks and other devices.

Note: As this section will amply demonstrate, saying that a device is "SCSI-3" or "SCSI-3 compatible" can mean anything--which of course is the same as saying it means nothing. Two devices labeled "SCSI-3" may not even use the same physical interface or commands! If you ever see a device advertised in this manner, be sure to get clarification on what is meant: specific details on the interface type and signaling method.

Note: It seems that the standards committee is trying to drop the "-3" designation from all of the SCSI-3 standards documents. The stated reason is that each standard is revised, it gets its own "dash number" extension. For example, the second SPI parallel specification is *SCSI-3 Parallel Interface - 2*. Some people thought the "-3" in "SCSI-3" would be confused with the dash number at the end of the document name, so they are dropping it and just calling SCSI-3 "SCSI" within these documents. Frankly, I wish they wouldn't do this--every time folks in the computer industry rename documents to "avoid confusion", they create more. :^)

Next: SCSI-3 Architecture

SCSI-3 Architecture

Since SCSI-3 defines a number of different standards, each covering different aspects of SCSI, it is necessary to organize these into a format that defines how they relate to each other, and the goals of the interface as a whole. This structure is called the *architecture* of SCSI-3. SCSI-3 architecture is defined by a document called the *SCSI-3 Architecture Model* or *SAM*, which has been approved as ANSI standard X3.270-1996. The *T10 technical committee* is also currently working on a revision to this document, called the *SCSI-3 Architecture Model - 2* or *SAM-2*. (SAM-2 is actually on its 14th revision at the time of this writing; it's been in progress for over four years!)

The SCSI-3 Architecture Model has several functions. An important one is to organize and categorize the various other standards that fall under SCSI-3. This serves to structure these standards in a way that makes sense to SCSI standards developers, hardware designers and users. The structure defines broad, generic requirements at a high level, which are refined to more specific low-level requirements through the use of particular implementation standards. Most of the different SCSI-3 documents fall into the following three general categories:

- **Commands:** These are standards that define specific command sets for either all SCSI devices, or for particular types of SCSI devices.
- **Protocols:** These standards formalize the rules by which various devices communicate and share information, allowing different devices to work together. These standards are sometimes said to describe the *transport layer* of the interface.
- **Interconnects:** These are standards that define specific interface details, such as electrical signaling methods and transfer modes. They are sometimes called *physical layer* standards as well.

The protocols and interconnects are often closely related, with a particular interconnect document typically being associated with a specific protocol standard. In some cases, especially lately, the protocol and interconnect standards are being combined into a single document. All of these standards are tied together by the architecture model, and also by the *Common Access Method* or *CAM*, which defines software services for host systems (computers) to interface with SCSI devices.

The architecture model is also responsible for providing much of the "foundation" for the other standards. This is important, because while the standards are developed and enhanced independently of each other, they must share certain common features if SCSI-3 is to remain a coherent standard as a whole. The SAM documents define high-level models of how SCSI works, requirements that apply to all SCSI implementations, and some of the fundamentals of how SCSI devices should be identified and addressed. The architecture model documents also serve as a single, unified place where common terms and concepts are defined. This is useful from a practical standpoint, to avoid the confusion that would result if the various SCSI-3 standards used inconsistent definitions for key terms and acronyms.

For a list and brief description of the standards defined in the SCSI-3 architecture, see the next page.

Next: Overview of SCSI-3 Standards

Overview of SCSI-3 Standards

The SCSI-3 standard is a family of other standards, as I have mentioned. These standards are organized into a structure as defined by the SCSI-3 architecture documents, which categorize the other documents into several groups. On this page I will list, and briefly describe, the documents that make up the SCSI-3 standard family. I am not going to get into too much detail here, as most of these standards will be of only passing interest to the PC user--even one who uses SCSI hardware. :^)

Note: These documents are constantly being updated as a result of ongoing work. While current as of this writing in late 2000, every month documents are revised and change status. For the latest status, or to download drafts of any documents under development, see the *T10 technical committee web site* or the other web sites referenced below.

The first group are the *command sets*, which define commands used for various SCSI commands. These include the following:

Command Set	Description	Document	Abbreviation and Generation	Status	Standard or Project
Shared	Commands defined for all SCSI devices	*SCSI-3 Primary Commands*	*SPC*	Published	X3.301-1997
			SPC-2	Pending Publication	T10 1236-D
			SPC-3	Development	T10 1416-D
Block	Commands defined for random-access devices that transfer data in blocks, such as hard disks	*SCSI-3 Block Commands*	*SBC*	Published	NCITS.306-1998
			SBC-2	Development	T10 1417-D
Block (Reduced)	A "simplified" version of the block command set	*SCSI-3 Reduced Block Commands*	*RBC*	Published	NCITS.330-2000
Stream	Commands for	*SCSI-3 Stream*	*SSC*	Published	NCITS.335-2000

	streaming, sequential-access devices such as tape drives	*Commands*	*SSC-2*	Development	T10 1434-D
Medium Changer	Commands for medium-changing devices such as tape or disk "jukeboxes"	*SCSI-3 Medium Changer Commands*	*SMC*	Published	NCITS.314-1998
			SMC-2	Development	T10 1383-D
Multimedia	Commands for "multimedia devices" (typically, optical drives)	*SCSI-3 Multimedia Commands*	*MMC*	Published	X3.304-1997
			MMC-2	Published	NCITS.333-2000
			MMC-3	Development	T10 1363-D
Multimedia (Reduced)	A "simplified" version of the multimedia command set	*SCSI-3 Reduced Multimedia Commands*	*RMC*	Development	T10 1364-D
Controller	Commands for RAID controllers	*SCSI-3 Controller Commands*	*SCC*	Published	X3.276-1997
			SCC-2	Published	NCITS.318-1998
Enclosure Services	Commands for SCSI device enclosures	*SCSI-3 Enclosure Services*	*SES*	Published	NCITS.305-1998
Object Based Storage Devices	Defines an object-oriented command set for accessing data	*Object Based Storage Device Commands*	*OSD*	Development	T10 1355-D

The other two groups of standards are *protocols* and *interconnects*. Protocols define how data is interchanged, and transports describe the physical ways that protocols are implemented. They are closely related and in some cases now are defined by a single document, which makes them a bit difficult to present in an organized way. I will start by discussing the various protocols (also called *transports* by some):

Protocol	Description	Document	Abbreviation and Generation	Status	Standard or Project
Interlocked (Parallel Bus)	Defines the protocol for "regular" parallel SCSI	*SCSI-3 Interlocked Protocol*	*SIP*	Withdrawn; now incorporated into later versions of the SCSI-3 Parallel Interface	--
Fibre Channel	Defines the protocol for running SCSI on the Fibre Channel interface	*SCSI-3 Fibre Channel Protocol*	*FCP*	Published	X3.269-1996
			FCP-2	Pending Publication	T10 1144-D
Serial Bus	Defines the protocol for transporting commands over the IEEE-1394 (serial) interface	*Serial Bus Protocol*	*SBP*	Withdrawn	--
			SBP-2	Published	NCITS.325-1998
Serial Storage Architecture	Defines the transport layer for Serial Storage Architecture, an advanced interface used in servers and enterprise hardware; there are two documents that specify the protocol	*Serial Storage Architecture SCSI-3 Protocol*	*SSA-S3P*	Published	NCITS.309-1998
		Serial Storage Architecture Transport Layer	*SSA-TL2*	Published (replaced SSA-TL1)	NCITS.308-1998

Finally, this table shows the interconnect standards, also sometimes called *physical layer* documents (since they describe how devices are physically connected):

Interconnect	Description	Document	Abbreviation and Generation	Status	Standard or Project
Parallel Bus	Describes the electrical signaling, connectors and related issues associated with "regular" parallel SCSI; starting with SPI-2 these include the formerly separate SIP protocol document	*SCSI-3 Parallel Interface*	*SPI*	Published	X3.253-1995
			Fast-20 (addendum to SPI)	Published	X3.277-1996
			SPI-2	Published	X3.302-1999
			SPI-3	Pending Publication	T10 1302-D
			SPI-4	Development	T10 1365-D
Fibre Channel	Several documents define alternative physical layer standards for Fibre Channel; these are maintained by the *T11 technical committee* and include Fibre Channel Arbitrated Loop (FC-AL) and several revisions of the Fibre Channel Physical Interface (FC-PHx)				
Serial Bus	The physical layer standards for the serial bus (IEEE-1394) are developed by the *IEEE High Performance Serial Bus Bridges Working Group* (P1394)				
Serial Storage Architecture	Defines the physical connections for the Serial Storage Architecture interface	*Serial Storage Architecture Physical Layer*	*SSA-PH*	Published	X3.293-1996
			SSA-PH2	Published	NCITS.307-1998

Whew. Well, you can certainly understand now why some people find SCSI-3 "a bit confusing". :^) I hope that this "road map" helps to make things a bit more clear for my readers though!

Next: SCSI(-3) Parallel Interface (SPI)

SCSI-3 Parallel Interface (SPI)

When the decision was made to expand the scope of SCSI-3 to include a number of different physical interfaces and protocols, what had been "just SCSI" had to be given a more specific name. Since "regular SCSI" uses a parallel bus (many wires transferring data in parallel), this technology became known as the *SCSI-3 Parallel Interface* or *SPI*. The first description of the parallel interface was accomplished in a rather confusing way, through the use of three different documents (see this page for more details on the standards documents):

- **Protocol:** The protocol for parallel SCSI was defined in a document entitled *SCSI-3 Interlocked Protocol* (*SIP*).
- **Physical Layer:** The physical layer was defined in the *SCSI-3 Parallel Interface* or *SPI* document, ANSI standard X3.253-1995. This specification only called for bus speeds of up to 10 MHz, which is so-called "Fast SCSI", first defined in SCSI-2.
- **Fast-20:** This is an addendum to the original SPI document, published as ANSI standard X3.277-1996. It defined faster 20 MHz bus signaling, increasing maximum throughput to as much as 40 MB/s on the SCSI bus.

Taken collectively, these are sometimes called Ultra SCSI or Wide Ultra SCSI, which are really informal or marketing terms; sometimes, Ultra SCSI refers specifically to the faster signaling rates themselves. Aside from the faster signaling, which allows for speeds of up to 20 MB/s on narrow (8-bit) SCSI buses or 40 MB/s on wide (16-bit) buses, the other main change associated with SPI is the creation of new cabling. Wide buses previously required two cables, a cumbersome solution that was never widely accepted. SPI introduced the high-density, 68-pin "P" cable and connectors now widely used for faster SCSI buses.

This collection of documents was "retired" in 1999 and replaced with a single document, SPI-2.

Next: SCSI(-3) Parallel Interface - 2 (SPI-2)

SCSI(-3) Parallel Interface - 2 (SPI-2)

The second generation of the SCSI-3 parallel interface standard is called the *SCSI(-3) Parallel Interface - 2*, or *SPI-2*. (The "-3" was dropped from "SCSI-3" to uh... reduce confusion. Sure. :^)) This ANSI standard, document X3.302-1999, replaced the older SPI standard, and also incorporated the SCSI-3 Interlocked Protocol (SIP) document. Thus, SPI-2 included everything from the earlier SCSI-2, SPI, SIP and Fast-20 documents--as well as adding some new features of course. SPI-2 formally replaced the earlier collection of SPI documents in 1999, and in doing so simplified matters significantly, since at least now everything associated with parallel SCSI was back in one document.

Several important new technologies and features were defined as part of SPI-2; the most important changes are the following:

- **Fast-40 Data Transfer:** SPI-2 defines another doubling of the maximum speed of the SCSI bus, from 20 MHz to 40 MHz, allowing maximum throughput of 40 MB/s on a narrow (8-bit) channel or 80 MB/s on a wide (16-bit) channel. The document also defines several restrictions associated with these faster signaling speeds, such as the use of differential signaling.
- **Low Voltage Differential Signaling:** A new type of signaling for the SCSI bus, called *low voltage differential* or *LVD* signaling, was specified as part of SPI-2. LVD is an attempt to blend the best attributes of conventional single-ended (SE) signaling and the older type of differential signaling that is now called high voltage differential (HVD). LVD (or the older HVD) is required to run the SCSI bus at Fast-40 speeds; you can read more about it here.
- **Multimode Operation:** Specification is provided for a way to create devices that will automatically work on both LVD and regular single-ended buses; such units are called *multimode devices*. They are also discussed in the section on LVD.
- **SCA-2 Single Connector Attachment Connectors:** An improvement to the original SCA connectors, called SCA-2, was defined.
- **Very High Density Connectors:** SPI-2 defined a smaller version of the older high-density 68-pin connectors. This new standard is called *Very High Density Cable Interconnect*, abbreviated *VHDCI*. Read more about them here.

Note: The features defined as part of SPI-2 are sometimes referred to by the informal (marketing) terms Ultra2 SCSI.and Wide Ultra2 SCSI.

Next: SCSI(-3) Parallel Interface - 3 (SPI-3)

SCSI(-3) Parallel Interface - 3 (SPI-3)

The third generation of the SCSI parallel interface is, unsurprisingly, called the *SCSI(-3) Parallel Interface - 3* or *SPI-3*. This document builds upon the physical and protocol definitions of the SPI-2 document. It is in the process of approval at the time of this writing, so it has no ANSI standard number yet, but should be published by early 2001 (it's T10 project 1302-D).

Five main features were added to parallel SCSI in the SPI-3 standard:

- **Fast-80(DT) Data Transfer:** Reflecting the continuing appetite for speed on the SCSI bus, data transfer rates were again doubled, this time to 160 MB/s on a wide bus. This was accomplished not by increasing the speed of the bus from 40 MHz to 80 MHz, but rather through the use of double transition clocking; thus the "DT" sometimes found in the name for this signaling speed. See here for more.
- **Cyclic Redundancy Check (CRC):** This is a common error checking protocol used to ensure data integrity. It was added as a safety measure since transfer speeds were being increased, leading to the possibility of data corruption.

- **Domain Validation:** This feature improves the robustness of the process by which different SCSI devices determine an optimal data transfer rate; read more about it here.
- **Quick Arbitration and Selection (QAS):** This feature represents a change in the way devices determine which has control of the SCSI bus, providing a small improvement in performance.
- **Packetization:** Another small change to improve performance, packetization reduces the overhead associated with each data transfer; it is described here.

Other, smaller changes were also made. SPI-3 also does some "cleanup" of the parallel SCSI standard, by making obsolete several older features that either never caught on in the industry, or were replaced with superior ways of accomplishing the same tasks:

- **High Voltage Differential:** With the widespread adoption of low voltage differential, the older "high voltage" differential became unnecessary. Since it was never very popular, it was removed from the standard.
- **32-Bit Bus Width:** Introduced in SCSI-2, the 32-bit parallel SCSI option never caught on in the industry and was finally removed from the specification in SPI-3.
- **SCAM:** SPI-3 removed the "SCSI Configured AutoMatically" (SCAM) feature, which was a good idea but never was universally adopted and sometimes led to configuration problems. In doing so, the SCSI world was mercifully rid of one of the worst acronyms in the history of the computer industry. :^)
- **Narrow High-Speed Transfers:** Narrow (8-bit) SCSI hasn't been technically "made obsolete", but 8-bit transfers are not defined for Fast-80 transfers. (Considering that faster transfer modes are used to get more throughput, increasing data transfer speeds while staying on an 8-bit bus never really made much sense.)

Unfortunately, despite the lessons that should have been learned in the past regarding what happens when standards aren't kept universal, the SCSI industry managed to create another mess out of the SPI-3 standard. The *SCSI Trade Association* defined the marketing term "Ultra3 SCSI" to correspond to the features introduced in SPI-3. However, they allowed a device that implemented *any* sub-set of the five main new features to be called "Ultra3 SCSI"; this "optionality" meant that there was no guarantee that any two devices labeled "Ultra3 SCSI" had the same features! Hardware manufacturers didn't like this, so they decided to market alternative names for more concrete subsets of the Ultra3 features, and we were off to the competing standards races yet again. The results were Ultra160 and Ultra160/m SCSI, and Ultra160+ SCSI.

(Someday, all these companies will get their act together and these things won't happen any more--and I'll probably die of shock. ;^))

Next: SCSI(-3) Parallel Interface - 4 (SPI-4)

SCSI(-3) Parallel Interface - 4 (SPI-4)

The latest revision of SPI will be called the *SCSI(-3) Parallel Interface - 4* or *SPI-4*. It is still in development within T10, and goes by the project name "T10 1365-D". Since it is still in the fairly early stages of development, it is unclear at this time exactly what it will include, beyond one feature.

The one feature that we now know *will* be included in SPI-4 is yet another doubling of maximum throughput on the SCSI bus. This will be accomplished through the implementation of what is being called *Fast-160(DT)*. Like Fast-80, this transfer mode uses double transition clocking, but increases the speed of the bus from 40 MHz to 80 MHz. This results in a maximum theoretical throughput of 320 MB/s on a wide bus. Like the Fast-80 transfers defined in SPI-3, Fast-160 is only supported on 16-bit buses and requires the use of LVD. The marketing people are calling this Ultra320, and despite SPI-4 only being in a draft form at present, products are being readied for the market using the new Fast-160 timing.

Beyond Fast-160, the crystal ball gets a bit hazy. I have seen mention of something called *Intersymbol Interference Compensation* (*ISI*), but can't find any information on it at present. I have also read that the packetization feature introduced in SPI-3 may be required for Fast-160 transfers. All of this information should be considered just rumor until the standard gets closer to being approved. Since new standards often change frequently before being finalized, I will probably wait until the dust settles before updating this page.

Next: SCSI Data Transfer Modes and Feature Sets

SCSI Data Transfer Modes and Feature Sets

The technologies used in SCSI are defined by the formal standards created and maintained by industry groups and standards associations. However, SCSI hardware is usually not sold labeled with the name of the standard whose features it implements. This is probably a good thing; considering the number of different protocols and transfer modes that are covered by both the SCSI-2 and SCSI-3 names, being told a drive is a "SCSI-2 drive" or "SCSI-3 device" doesn't really tell you much of anything at all!

Instead, SCSI devices are usually sold using specific names that define particular "flavors" of SCSI. Each of these is a particular intersection of various important SCSI characteristics, such as bus speed, bus width and signaling type. In most cases new feature sets are created when new transfer modes are created by the adoption of new standards; they are then given cute names by various hardware makers or manufacturers' associations and used to promote the new products. The various feature set names can be hard to understand since some of them are similar to each other despite different-sounding names.

In this section I provide a description of each of the common feature sets used for regular parallel SCSI. For each one I provide the following information in a standardized format:

- **Description:** A brief description of the particular SCSI "flavor".
- **Defining Standard:** The specific SCSI standard that introduced the technology used in this type of SCSI.
- **Special Features:** Specific features or requirements of a hardware device of this particular SCSI type.
- **Bus Width:** The width of the bus for this SCSI type, either narrow (8-bit) or wide (16-bit).
- **Signaling Method:** What combination of single-ended (SE), differential (HVD) and low voltage differential (LVD) signaling is used by this type of SCSI. Note that HVD has been withdrawn from the SCSI standard, but some HVD hardware is still "out there" and will of course still work with HVD host adapters.
- **Signaling Speed and Bus Throughput:** The speed that this SCSI bus runs at, and the data throughput in MB/s.
- **Number of Devices Supported:** The number of devices that can be put on a single SCSI chain. This is the total number of devices; remember that the host adapter counts as one device.
- **Termination:** The type of SCSI bus termination required or normally used.
- **Cabling and Maximum Cable Length:** The type of cables supported, and the maximum length of the SCSI chain supported, in meters. Some types of SCSI support different cable lengths depending on the signaling method. Oh, for you folks still stuck in the land of Imperial measure, 1 meter is about 3.28 feet. :^)

Next: "Regular" SCSI (SCSI-1)

"Regular" SCSI (SCSI-1)

Description: With the creation of so many new types of SCSI over the years, the oldest SCSI implementations really have no specific name; they are sometimes called "regular" SCSI or other similar names. Since the only type of SCSI defined by the original SCSI-1 standard is this "plain" variety, some people call this variety of SCSI "SCSI-1".

Defining Standard: SCSI-1.

Special Features: None.

Bus Width: Narrow (8-bit).

Signaling Method: SE or HVD.

Signaling Speed and Bus Throughput: 5 MHz bus speed; 5 MB/s.

Number of Devices Supported: 8.

Termination: For SE, any type (passive, active or forced perfect termination). For HVD, HVD termination.

Cabling and Maximum Cable Length: "A" cable (50 pins). Maximum of 6m for SE, 25m for HVD.

Next: Wide SCSI

Wide SCSI

Description: *Wide SCSI* refers to devices that use a standard-speed, 5 MHz SCSI bus but on a wide, 16-bit bus. It is also sometimes called *Wide SCSI-2* after the standard that defined it.

Defining Standard: SCSI-2.

Special Features: None.

Bus Width: Wide (16-bit).

Signaling Method: SE or HVD.

Signaling Speed and Bus Throughput: 5 MHz bus speed; 10 MB/s.

Number of Devices Supported: 16.

Termination: For SE, any type (passive, active or forced perfect termination). For HVD, HVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). (Formerly, "A" cable plus "B" cable.)Maximum of 6m for SE, 25m for HVD.

Next: Fast SCSI

Fast SCSI

Description: *Fast SCSI* refers to devices that use 10 MHz SCSI signaling speed on a narrow, 8-bit bus. It is also sometimes called *Fast SCSI-2* after the standard that defined it.

Defining Standard: SCSI-2.

Special Features: None.

Bus Width: Narrow (8-bit).

Signaling Method: SE or HVD.

Signaling Speed and Bus Throughput: 10 MHz bus speed; 10 MB/s.

Number of Devices Supported: 8.

Termination: For SE, either active or forced perfect termination. For HVD, HVD termination.

Cabling and Maximum Cable Length: "A" cable (50 pins). Maximum of 3m for SE, 25m for HVD.

Next: Fast Wide SCSI

Fast Wide SCSI

Description: *Fast Wide SCSI* combines the features of Wide SCSI and Fast SCSI, so it uses 10 MHz SCSI bus speed on a 16-bit wide bus. This doubles the maximum transfer rate to 20 MB/s, the fastest bus defined under the SCSI-2 standard. This protocol is sometimes called *Fast Wide SCSI-2*.

Defining Standard: SCSI-2.

Special Features: None.

Bus Width: Wide (16-bit).

Signaling Method: SE or HVD.

Signaling Speed and Bus Throughput: 10 MHz bus speed; 20 MB/s.

Number of Devices Supported: 16.

Termination: For SE, either active or forced perfect termination. For HVD, HVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). (Formerly, "A" cable plus "B" cable.)Maximum of 6m for SE, 25m for HVD.

Next: Ultra SCSI

Ultra SCSI

Description: *Ultra SCSI* is the term usually applied to devices that use 20 MHz SCSI bus speed on a narrow, 8-bit bus. This is sometimes called *Fast-20 SCSI* in reference to the Fast-20 signaling specification that defines Ultra SCSI speed. Note that "Ultra SCSI" is sometimes used vaguely; for example, some people call both Wide Ultra SCSI and (non-wide) Ultra SCSI just "Ultra SCSI" collectively.

Defining Standard: SCSI-3 / SPI (including Fast-20 addendum).

Special Features: None.

Bus Width: Narrow (8-bit).

Signaling Method: SE or HVD.

Signaling Speed and Bus Throughput: 20 MHz bus speed; 20 MB/s.

Number of Devices Supported: 8 if HVD signaling is used or SE signaling is used with a maximum cable length of 1.5m; 4 if SE signaling is used with a cable length of over 1.5m.

Termination: For SE, either active or forced perfect termination. For HVD, HVD termination.

Cabling and Maximum Cable Length: "A" cable (50 pins). Maximum of 3m for SE if no more than 4 devices are used, otherwise 1.5m; 25m for HVD.

Next: Wide Ultra SCSI

Wide Ultra SCSI

Description: *Wide Ultra SCSI* refers to devices that use 20 MHz SCSI bus speed on a wide, 16-bit bus, allowing for throughput of up to 40 MB/s. This is sometimes called *Fast-20 Wide SCSI* in reference to the Fast-20 signaling specification that defines Ultra SCSI speed. It is also sometimes called *Ultra Wide SCSI*.

Defining Standard: SCSI-3 / SPI (including Fast-20 addendum).

Special Features: None.

Bus Width: Wide (16-bit).

Signaling Method: SE or HVD.

Signaling Speed and Bus Throughput: 20 MHz bus speed; 40 MB/s.

Number of Devices Supported: 16 if HVD signaling is used; 8 if SE signaling is used with a maximum cable length of 1.5m; 4 if SE signaling is used with a cable length of over 1.5m.

Termination: For SE, either active or forced perfect termination. For HVD, HVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). Maximum of 3m for SE if no more than 4 devices are used, otherwise 1.5m; 25m for HVD.

Next: Ultra2 SCSI

Ultra2 SCSI

Description: *Ultra2 SCSI* is the marketing term for devices corresponding to the *SCSI-3 Parallel Interface - 2* (*SPI-2*) standard that run on a narrow (8-bit) bus. These units support a maximum throughput of 40 MB/s using Fast-40 signaling; they are occasionally (rarely) called *Fast-40 SCSI* devices. See the SPI-2 standard discussion and the details below for more on this type of SCSI.

Note: While narrow Ultra2 SCSI has been formally defined, it was never very popular; most devices using Ultra2 SCSI are in fact Wide Ultra2 SCSI devices, because "giving away" half the maximum throughput by using narrow cabling has become quite unpopular. The sparsity of narrow Ultra2 implementations means that many people use just "Ultra2 SCSI" to refer to what are really *wide* devices. Watch out for this.

Note: Don't confuse Ultra2 SCSI with SCSI-2, which is a totally different (and much older) standard.

Defining Standard: SCSI-3 / SPI-2.

Special Features: LVD signaling; multimode (LVD/SE) optional.

Bus Width: Narrow (8-bit).

Signaling Method: LVD or HVD. (HVD is officially supported for Ultra2 SCSI, though it is not generally used; LVD offers significant advantages over HVD and has become the standard for modern high-speed SCSI buses.) Note that multimode drives may optionally run in SE mode, but throughput will drop to Fast-20 (Ultra) levels if this is done.

Signaling Speed and Bus Throughput: 40 MHz bus speed; 40 MB/s.

Number of Devices Supported: 8 for HVD or LVD cables up to 12m in length; 2 for LVD cables over 12m.

Termination: For LVD, LVD termination; for HVD, HVD termination.

Cabling and Maximum Cable Length: "A" cable (50 pins). Maximum of 25m for LVD if no more than 2 devices are used, otherwise 12m; 25m for HVD.

Next: Wide Ultra2 SCSI

Wide Ultra2 SCSI

Description: *Wide Ultra2 SCSI* is the marketing term for devices corresponding to the *SCSI-3 Parallel Interface - 2* (*SPI-2*) standard that run on a wide (16-bit) bus. These units support a maximum throughput of 80 MB/s using Fast-40 signaling; they are occasionally (rarely) called *Wide Fast-*

40 SCSI devices. See the SPI-2 standard discussion and the details below for more on this type of SCSI.

Note: Most SCSI implementations that use Ultra2 signaling are in fact wide implementations. Narrow Ultra2 SCSI is supposed to be called just "Ultra2 SCSI" while the wide version is "Wide Ultra2"; however, since narrow never really caught on for Ultra2, some people say "Ultra2 SCSI" when they really mean "Wide Ultra2 SCSI".

Note: Don't confuse Wide Ultra2 SCSI with Wide SCSI-2; the latter is a less common name for Wide SCSI, a much older and slower transfer mode.

Defining Standard: SCSI-3 / SPI-2.

Special Features: LVD signaling; multimode (LVD/SE) optional.

Bus Width: Wide (16-bit).

Signaling Method: LVD or HVD. (HVD is officially supported for Wide Ultra2 SCSI, though it is not generally used; LVD offers significant advantages over HVD and has become the standard for modern high-speed SCSI buses.) Note that multimode drives may optionally run in SE mode, but throughput will drop to Fast-20 (Ultra) levels if this is done.

Signaling Speed and Bus Throughput: 40 MHz bus speed; 80 MB/s.

Number of Devices Supported: 16 for HVD or LVD cables up to 12m in length; 2 for LVD cables over 12m.

Termination: For LVD, LVD termination; for HVD, HVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). Maximum of 25m for LVD if no more than 2 devices are used, otherwise 12m; 25m for HVD.

Next: Ultra3 SCSI

Ultra3 SCSI

Description: *Ultra3 SCSI* is the marketing term created by the SCSI Trade Association to refer to devices that implement some or all of the key features defined in the *SCSI-3 Parallel Interface - 3* (*SPI-3*) standard. The units can support a maximum throughput of up to 160 MB/s using Fast-80 signaling with double transition clocking. See the SPI-3 standard discussion and the details below for more on this type of SCSI. Note that Ultra3 SCSI is wide (16-bit) only; 8-bit support was dropped for this version of the interface. This means the name is just "Ultra3 SCSI"; the "Wide" is implied and not explicitly mentioned.

Unfortunately, the decision by the SCSI TA to define the five main features of SPI-3 as *optional* for Ultra3 SCSI led to a revolt of sort on the part of hardware manufacturers. They made the (very valid) point that this definition of Ultra3 SCSI allows for devices with very different features to all be called "Ultra3 SCSI"--even devices that don't support Fast-80 signaling, the key feature of SPI-3! The hardware makers wanted a more concrete standard, and this led to the creation of the rather similar Ultra160(/m) SCSI and Ultra160+ SCSI feature sets.

Note: Don't confuse Ultra3 SCSI with SCSI-3, which is a generic term that can refer to any device corresponding to any of the myriad SCSI-3 standards (and is therefore meaningless.)

Defining Standard: SCSI-3 / SPI-3.

Special Features: Ultra3 SCSI devices include support for at least one of the following five features:

- Fast-80(DT) data transfer
- Cyclic Redundancy Check (CRC)
- Domain validation
- Quick Arbitration and Selection (QAS)
- Packetization

Bus Width: Wide (16-bit). Narrow mode is not supported.

Signaling Method: LVD only, if Fast-80 is being used. (Multimode drives may optionally run in SE mode, but throughput will drop to Fast-20 (Ultra) levels.)

Signaling Speed and Bus Throughput: Depends on implementation; assuming Fast-80, 40 MHz bus speed; 160 MB/s.

Number of Devices Supported: 16 for cables up to 12m in length; 2 for cables over 12m.

Termination: LVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). Maximum of 25m if no more than 2 devices are used, otherwise 12m.

Next: Ultra160 (Ultra160/m) SCSI

Ultra160 (Ultra160/m) SCSI

Description: The SCSI Trade Association created the Ultra3 SCSI marketing program to identify drives implementing some or all of the main features associated with the *SCSI-3 Parallel Interface - 3* (*SPI-3*) standard. This created a controversy, because hardware could be called "Ultra3 SCSI" even if it implemented only one of the five key features in the SPI-3 standard. Hardware makers feared this would create compatibility problems, especially since some "Ultra3 SCSI" devices might not support the 160 MB/s transfer speed that is the key feature of the new standard.

To address this possible source of confusion, several SCSI hardware companies decided to stay way from the "Ultra3 SCSI" label, and created a specific feature set called *Ultra160/m*, where the "160" refers to the maximum throughput of the interface--the "/m" means "maximum" or "manageability". The "/m" suffix was later dropped to "prevent confusion", yielding just *Ultra160*. Once again, this created *more* confusion, not less, because many people think Ultra160 and Ultra160/m are different. (Hardware makers! Stop confusing people by renaming standards to avoid confusing people! ;^))

Note: Ultra160+ *is* a different subset of Ultra3 SCSI, and is not the same as Ultra160 or Ultra160/m. Aren't standards great? ;^)

See the discussion of the SPI-3 standard and the items below for more details on Ultra160 SCSI. Incidentally, in 1999 the SCSI trade association recognized the Ultra160 feature set; it also seems to be sticking with that terminology going forward, given that the next generation of SCSI will be called Ultra320 and not "Ultra4".

Defining Standard: SCSI-3 / SPI-3.

Special Features: Ultra3 SCSI devices include support for the following three SPI-3 features:

- Fast-80(DT) data transfer
- Cyclic Redundancy Check (CRC)
- Domain validation

The following features are considered optional for Ultra160:

- Quick Arbitration and Selection (QAS)
- Packetization

Bus Width: Wide (16-bit). Narrow mode is not supported.

Signaling Method: LVD only. (Multimode drives may optionally run in SE mode, but throughput will drop to Fast-20 (Ultra) levels.)

Signaling Speed and Bus Throughput: 40 MHz bus speed; 160 MB/s.

Number of Devices Supported: 16 for cables up to 12m in length; 2 for cables over 12m.

Termination: LVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). Maximum of 25m if no more than 2 devices are used, otherwise 12m.

Next: Ultra160+ SCSI

Ultra160+ SCSI

Description: Ultra160 SCSI--a.k.a. "Ultra160/m"--defines a specific subset of the features associated with Ultra3 SCSI. It was created to avoid the confusion associated with the various options that make up the Ultra3 SCSI definition. *Ultra160+ SCSI* is another feature set based on the five key enhancements to the SCSI interface introduced in the *SCSI-3 Parallel Interface - 3* (*SPI-3*) standard. While Ultra160 SCSI defines three of the five key features as being mandatory, Ultra160+ SCSI refers to devices that implement *all five* of the key SPI-3 features.

To summarize the differences between these three related feature sets:

- **Ultra3 SCSI:** At least one, and as many as five, of the main SPI-3 features.
- **Ultra160(/m) SCSI:** Fast-80 signaling, CRC and domain validation are mandatory; QAS and packetization optional.
- **Ultra160+ SCSI:** All five features are mandatory.

Aside from these definitional differences, all these terms refer to the same basic interface. See the discussion of the SPI-3 standard and the items below for more details.

Defining Standard: SCSI-3 / SPI-3.

Special Features: Ultra3 SCSI devices include support for all of the following five SPI-3 features:

- Fast-80(DT) data transfer
- Cyclic Redundancy Check (CRC)
- Domain validation
- Quick Arbitration and Selection (QAS)
- Packetization

Bus Width: Wide (16-bit). Narrow mode is not supported.

Signaling Method: LVD only. (Multimode drives may optionally run in SE mode, but throughput will drop to Fast-20 (Ultra) levels.)

Signaling Speed and Bus Throughput: 40 MHz bus speed; 160 MB/s.

Number of Devices Supported: 16 for cables up to 12m in length; 2 for cables over 12m.

Termination: LVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). Maximum of 25m if no more than 2 devices are used, otherwise 12m.

Next: Ultra320 SCSI

Ultra320 SCSI

Description: *Ultra320 SCSI* is the current name given by the SCSI Trade Association to hardware being developed to the next generation of the SCSI parallel interface, as defined in drafts of the *SCSI-3 Parallel Interface - 4* (*SPI-4*) standard. As the name implies, these drives support a maximum throughput of 320 MB/s using 80 MHz bus signaling and double transition clocking. This technology was at one point known by the name "Ultra4 SCSI", but it appears that this terminology will not be used going forward.

For more information, see the discussion of the SPI-4 standard and the information below. Bear in mind that SPI-4 and Ultra320 are still under development. Not much is known about the specific features that will be agreed to in this new version of the interface, so details are not yet complete.

Defining Standard: SCSI-3 / SPI-4 (in development).

Special Features: Fast-160(DT) data transfer; others not yet confirmed at this time.

Bus Width: Wide (16-bit) only.

Signaling Method: LVD only. (Multimode drives may optionally run in SE mode, but throughput will drop to Fast-20 (Ultra) levels.)

Signaling Speed and Bus Throughput: 80 MHz bus speed; 320 MB/s.

Number of Devices Supported: 16 for cables up to 12m in length; 2 for cables over 12m.

Termination: LVD termination.

Cabling and Maximum Cable Length: "P" cable (68 pins). Maximum of 25m if no more than 2 devices are used, otherwise 12m.

Next: SCSI Protocol Compatibility

SCSI Transfer Mode and Feature Set Compatibility

I sometimes call the various types of SCSI "flavors"; well, if that's so, the way the interface is heading we'll soon be in Baskin Robbins territory. ;^) The sheer number of different kinds of SCSI can certainly make the interface seem overwhelming! How does a SCSI user make it all of this hardware work together? Fortunately, while the standards and feature sets can be quite confusing, the hardware is actually well-engineered, and the standards are designed to allow different hardware types to work together fairly readily.

It's important to remember that a key design goal of all SCSI standards is *backwards compatibility*. Few people want to buy new hardware that won't work with their older hardware. Therefore, in most cases, at least in theory, you can mix older, slower hardware with newer, faster hardware. You can, again *in theory*, put a brand-new Ultra160 SCSI hard disk on the same SCSI bus with a decade-old SCSI-1 host adapter (albeit with added hardware and suboptimal results.) This is generally true, but note the important qualifier: *in theory*. Since changes are always being made to the signaling and other aspects of the interface, there is no guarantee that any two very different pieces of SCSI hardware will work together.

There are no hard and fast rules regarding the compatibility of different SCSI transfer modes and feature sets, especially if they are very different in terms of key attributes. Here are some issues that you should keep in mind as you consider device compatibility:

- **Age:** The greater the difference in age between two devices, the greater the difficulties associated with getting them to work together. The extreme example I gave above of trying to get an Ultra160 drive to work with a SCSI-1 host adapter (or vice-versa) would probably not be much fun. :^) However, mixing Ultra160 and Ultra2 devices is fairly straightforward.
- **Drive and Host Speed Negotiation:** You can use faster drives on slower host adapters or vice versa, but communication will only occur as fast as the slowest device can handle. For example, you can connect a Wide Ultra SCSI drive to an Ultra160 host adapter, but the drive will only run at a maximum of 40 MB/s throughput, not 160 MB/s.
- **Signaling:** Mixing different types of signaling on the same bus can lead to problems ranging from slowdowns to disaster. :^) The older (high voltage) differential signaling is not electrically compatible with either single-ended or LVD devices, and should never be mixed with those types, or you risk *disaster* such as smoked hardware. Multimode LVD devices can be mixed with SE devices, but they won't function at Ultra2 or higher speeds if you do so.
- **Bus Width:** You can mix wide and narrow devices on the same SCSI bus, but there are specific requirements in doing this, to ensure that the bus functions properly.
- **Packages:** If you want to be sure that a particular SCSI implementation will work, buy a complete system or SCSI "package" (including a host adapter, drives, cables and terminators) from a reputable dealer.
- **Ask For Help:** If you are considering a particular hardware combination, ask for advice on it before trying the setup. Unless the setup is extremely strange, someone may have already done what you are contemplating. A good place to try is the *comp.periphs.scsi* newsgroup.

Next: SCSI Protocols and Interface Features

SCSI Protocols and Interface Features

In detailing the various standards, transfers modes and feature sets associated with the SCSI interface, I have introduced several important concepts that define the attributes of various types of SCSI buses. New technologies are often introduced specifically to change these traits, to improve the interface. It's important to understand how the various aspects of SCSI combine to create different specific SCSI varieties.

In this section I describe the most important characteristics of the SCSI interface. This includes a discussion of the three most important defining characteristics of any SCSI bus: signaling, bus speed and bus width. I then discuss several important SCSI bus features, many of which have been introduced to improve performance or reliability on the newest, highest-performance SCSI implementations. This includes a discussion of bus integrity protection, and advanced features such as command queuing and reordering, domain validation, quick arbitration and selection, and packetization.

Next: Single-Ended (SE) and Differential (High Voltage Differential, HVD) Signaling

Single-Ended (SE) and Differential (High Voltage Differential, HVD) Signaling

Conventional SCSI signaling is very similar to that used for most other interfaces and buses within the PC. Conventional logic is used: a positive voltage is a "one", and a zero voltage (ground) is a "zero" (see this fundamentals page on signaling for more). This is called *single-ended* signaling, abbreviated *SE*. Up until recently, single-ended SCSI had been by far the most popular signaling type in the PC world, for a simple reason: it is relatively simple and inexpensive to implement.

There's an important problem with SE signaling, however. SCSI is a high-speed bus capable of supporting multiple devices, including devices connected both inside and outside the PC. As with all high-speed parallel buses, there is always a concern about signal integrity on the bus; problems can arise due to bouncing signals, interference, degradation over distance and cross-talk from adjacent signals. The faster the bus runs, the more these problems manifest themselves; the longer the cable, the more the problems exist for any given interface speed. As a result, the length of a single-ended SCSI cable is rather limited, and the faster the bus runs, the shorter the maximum allowable cable length.

To get around this problem, a different signaling method was also defined for SCSI, which uses two wires for each signal that are mirror images of each other. For a logical "zero", zero voltage is sent on both wires. For a logical "one", the first wire of each signal pair contains a positive voltage, similar to the signal on an SE bus, but not necessarily at the same voltage. The second wire contains the electrical opposite of the first wire. The circuitry at the receiving device takes the *difference* between the two signals sent, and thus sees a relatively high voltage for a one, and a zero voltage for a zero. This method is *much* more resilient to signaling problems than regular SE signaling. It is called *differential* signaling, after the technique used to determine the value of each signal by the recipient. The two signals in each pair are usually named with "+" and "-" signs; for example, the signal carrying data bit 0 would use "+DB(0)" and "-DB(0)". See the section cabling for more information.

This table shows the great difference in cable length that exists between SE and differential devices, particularly as bus speed increases:

Signaling Speed	**Bus Speed (MHz)**	**Single-Ended SCSI Maximum Cable Length (m)**	**Differential SCSI Maximum Cable Length (m)**
Slow	5	6	25
Fast	10	3	25
Fast-20	20	1.5	25

As you can see, each doubling of the bus speed results in a halving of the maximum cable length for single-ended SCSI, but differential SCSI allows long (25m) cables for all three speeds. (Fast-20 buses allow a cable length of 3m if no more than four devices are used, but this is really a kludge of sorts to get around the limitations associated with a 1.5m cable restriction.)

Differential SCSI is a great idea in theory, and one might have thought it would become very popular. In fact, this never happened in the PC world, largely due to cost. The circuits needed to drive differential signals are more expensive and use more power than those for single-ended SCSI. For many years, single-ended SCSI was "good enough", and allowed cable lengths sufficient for the needs of most users, so little impetus was seen to move to the more expensive differential signaling. From there, "chicken-and-egg" syndrome kicked in: since differential was less popular, it was not produced in volume and so never saw its costs come down due to economies of scale.

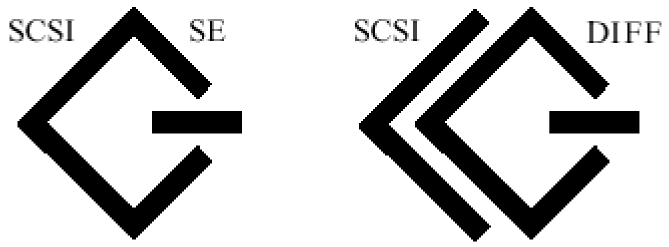
The end result of all of this is that the older type of differential signaling is rarely seen in the PC world. The concept of differential signaling, however, did not die out. As the SCSI bus was pushed to faster and faster speeds, the cable limits of SE were finally too great to be worked around. However, the cost of regular differential was unappealing, so a new type of differential signaling was created, called *low voltage differential* or *LVD*. See this page for more on LVD signaling. With the creation of LVD, the old name of "differential" for the higher-voltage version became vague, so the older style was renamed *high voltage differential* or *HVD*.

High voltage differential signaling has been around since the earliest SCSI-1 standard, so devices have been theoretically available as either SE or HVD since the start of SCSI use on the PC--of course, whether any particular model has been implemented in HVD is another story altogether. :^) With the creation of the SPI-3 specification and the standardization of LVD, HVD had no more *raison d'etre*, and has been removed from the SCSI standard entirely, leaving only LVD. There are no new drives being produced that use HVD.

Note: Since differential signaling uses complementary positive and negative voltages, it is sometimes called *balanced* signaling; single-ended signaling is similarly called *unbalanced*. This is just different terminology for the same technologies.

Warning: Since single-ended and HVD SCSI use very different voltage levels, they are incompatible at the electrical level. You should not mix single-ended (or low voltage differential) devices with high voltage differential SCSI devices on the same bus. If you do, actual physical damage could result--this is one of those cases where actually smoking your hardware is a distinct possibility, because of the high voltages that might be sent to the single-ended or LVD devices! To compound the matter, the cables and connectors used for single-ended and differential SCSI look the same.

To help reduce the chances that similar-looking SE and HVD hardware will be interconnected, special *icons* are imprinted on SCSI hardware that indicates the signaling method used by the device. Make sure you know what you have before putting together your SCSI bus, and look for these identifying symbols on devices to be sure they are electrically compatible. Note that slightly different symbols are used for LVD and LVD/SE devices, as described here. Also see this section for more on cables and connectors.



Icons for hardware using single-ended SCSI (left) and regular (high voltage) differential SCSI (right).

Images © Information Technology Industry Council (ITI) (See here for more on ITI.) *Images used with permission.*

Next: Low-Voltage Differential (LVD) Signaling

Low-Voltage Differential (LVD) Signaling

Differential signaling has been around since the first SCSI specification, SCSI-1. As described in detail on this page, high voltage differential (HVD) signaling uses two wires for each signal to improve signal integrity and allow long cables to be used without data loss or corruption. However, it is expensive to implement, and uses a great deal of power. For this reason, it has never caught on, and through the late 1990s, conventional single-ended (SE) signaling was the standard in the SCSI world.

Single-ended SCSI signaling worked fine for many years. However, when the SCSI industry was ready to increase the speed of the SCSI bus to 40 MHz, they had a serious problem. As the table on this page shows, each doubling of the bus speed results in a halving of the maximum cable length allowed under SE. Since 20 MHz speeds had already dropped maximum cable length to 1.5m, halving it again would have resulted in a maximum length of only 0.75m--that's just a little over two feet for an entire SCSI chain! The alternative was to go to the older type of (high voltage) differential signaling, with its high cost and catastrophic electrical incompatibility with single-ended hardware.

Instead, a third option was created, with the intention of marrying the best attributes of both SE and HVD signaling. This is a differential signaling method that was designed to use the advantages of differential signaling to allow long cable lengths, while reducing implementation cost and allowing for electrical compatibility with single-ended devices. This technology is called *low voltage differential* or *LVD* signaling. It was first defined in the SPI-2 standard and is rapidly becoming the signaling method of choice in the SCSI world. In fact, LVD signaling is required for Ultra2 or Wide Ultra2 SCSI (unless HVD is used), and LVD is the exclusive signaling method for all SCSI modes faster than Ultra2. Even the fastest LVD SCSI chains can be up to 12m in length, or 25m if only two devices are used on the chain (this is called *point-to-point* operation; remember that one of these must be the interface card, the host adapter.)

The concept behind LVD is relatively straight-forward: continue using two wires for each signal, but use lower voltage to create the complementary signal pairs. Using lower voltage allows cost to be reduced and power requirements to be kept under control. It also means that the dangers associated with mixing SE and differential devices is eliminated. In fact, single-ended devices are not just electrically compatible with LVD devices, some types of LVD devices can even function on single-ended SCSI buses.

A particular type of LVD device was defined when LVD was created; drives that correspond to this variant of LVD are called *multimode LVD device*. These are usually abbreviated *LVD/SE* or *LVD/MSE* (the "M" is for "multimode"). A multimode LVD device will automatically switch between LVD and single-ended operation by detecting whether the other devices on the chain are running in SE or LVD mode. (Note that only one or the other can be used at a time; the device won't use both simultaneously.)

In addition to the usual SCSI rules--such as unique IDs for each device and proper topology and termination--LVD operation requires the following:

1. All devices on the chain must be LVD-capable; if even one device is only SE, all devices "drop down" and run as single-ended.

2. All devices must not be set to run in SE mode; some multimode devices have a jumper to "force" SE operation, which will cause the entire SCSI chain to not work in LVD.
3. LVD (or multimode LVD/SE) terminators must be used.

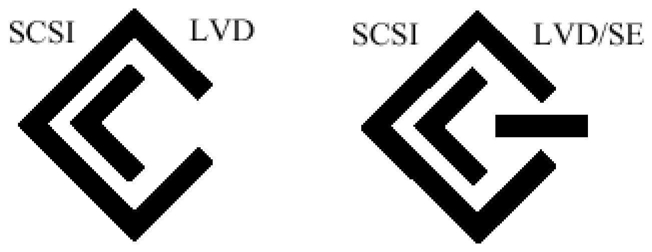
Remember that bus speeds over 20 MHz are not supported under single-ended operation. This means that a multimode LVD/MSE Ultra160 device will run at only a maximum of 40 MB/s if it is connected to a SCSI chain with single-ended devices.

Warning: As soon as multimode LVD devices begin running as single-ended, all the rules and restrictions of single-ended operation apply, including cable length. For example, suppose you have a 4m cable connecting an LVD Ultra160 host adapter to a multimode LVD Ultra160 device; this is perfectly fine. Now, let's say you decide to add to this cable a Wide Ultra single-ended device. As soon as this happens, the other devices will drop down to single-ended operation, and probably will try to run at Ultra speeds (Fast-20). Communication problems will then result due to the fact that a 4m cable is not supported at Ultra speeds in single-ended operation.

LVD signaling is rapidly taking over the SCSI world. Single-ended operation is not supported for bus speeds faster than 20 MHz ("Ultra" bus speeds), so to use Ultra2 or faster SCSI, differential must be used. HVD for its part was made obsolete in the SPI-3 standard, so for Ultra3, Ultra160, Ultra160+ and faster speeds, LVD is the only option.

Warning: Low voltage differential devices are not electrically compatible with high voltage differential hardware; do not mix them on the same SCSI cable or damage to the LVD devices may occur.

To help reduce the chances that similar-looking LVD and HVD hardware will be interconnected, special *icons* are imprinted on SCSI hardware that indicates the signaling method used by each device. Make sure you know what you have before putting together your SCSI bus, and look for these identifying symbols on devices to be sure they are electrically compatible. Note that slightly different symbols are used for SE and HVD devices, as described here. Also see this section for more on cables and connectors.



Icons for hardware using LVD SCSI (left) and multimode LVD/SE SCSI (right).

Images © Information Technology Industry Council (ITI) (See here for more on ITI.) *Images used with permission.*

Next: SCSI Bus Width

SCSI Bus Width

There are two commonly used SCSI bus widths: narrow and wide. Narrow SCSI uses a data pathway that is 8 bits wide, and was the first type of parallel SCSI defined in the original SCSI-1 standard. Wide SCSI uses a data pathway 16 bits wide, and was first defined as part of SCSI-2. Since its introduction, wide SCSI has been steadily increasing in popularity, since it allows a doubling of bus bandwidth for any given signaling speed. It also allows the use of 16 devices on the SCSI bus, compared to the standard 8 devices for narrow SCSI.

Wide SCSI originally required the use of two cables: a 68-pin "B" cable in addition to the regular 50-conductor "A" cable used for narrow SCSI. This use of two cables was expensive and cumbersome, and the "A+B" configuration was eventually replaced by a single 68-pin "P" cable. See here for more on cabling issues.

Today, narrow SCSI is actually being left behind, as the need for extra performance has led to the dominance of wide forms of SCSI, especially for hard disks. This has actually led to some terminology difficulties. Traditionally, the narrow SCSI bus has been considered the "regular" or default type, so "narrow" was not generally mentioned in the name of SCSI type. For example, saying "Ultra SCSI" implied narrow operation; wide buses running at Ultra speeds were called "Wide Ultra SCSI". However, at around the time that Ultra2 SCSI was created, narrow operation began to fall out of favor, and as a result most Ultra2 implementations are wide, and many people stopped bothering with explicitly saying "Wide Ultra2 SCSI", even though this is the technically accurate name.

Transfer modes faster than Ultra2 have done away with narrow buses altogether. Presumably, if one is designing a device that needs throughput enough to justify going to speeds faster than Ultra2, it would be silly to "give away" half the throughput by going narrow instead of wide. Fast-80(DT) and Fast-160(DT) signaling, as defined in the SPI-3 and SPI-4 standards respectively, are only for wide implementation. As a result, all relevant marketing terms such as Ultra3, Ultra160, Ultra160+ and Ultra320 have *wide bus operation implied*, reversing the way it was with the earlier SCSI flavors.

It is possible to mix narrow and wide SCSI on the same bus, but there are issues that must be overcome to do so. These typically revolve around cabling, which is different for narrow and wide SCSI, and also with termination. Adapters may be required to convert between the narrow and wide cables. See this discussion for more information. Note that there are also host adapters available that are specifically designed to support both wide and narrow devices.

Note: A "very wide" 32-bit form of SCSI was defined as part of the SCSI-2 standard, but was never accepted by the industry due to cost. It required the use of two 68-conductor cables, for one thing! It was also non-standard; with the extra costs involved, there was little interest in it. After laying the proverbial egg for several years, it was finally withdrawn from the SCSI

mehedi hasan
mehedi@joinme.com

standard in SPI-3.

Next: SCSI Bus Speed

SCSI Bus Speed

SCSI buses run at a variety of different speeds. Generally, newer buses run faster than older ones, reflecting the increased performance of newer hardware. In order to understand SCSI bus speeds, we must first tackle some terminology issues (yet again, sorry) There are several different ways that SCSI bus speeds are specified, which can lead to a tremendous amount of confusion. The situation is particularly bad because base SCSI speeds vary by powers of two, and the differences between the various ways of specifying SCSI bus speeds also differ by powers of two!

These are the three ways that SCSI bus speeds are commonly quoted:

- **Clock Speed:** This refers strictly to the frequency of the clock (strobe) used to control synchronous transfers of data on the SCSI bus. With current technology this can be 5, 10, 20, 40 or 80 MHz. For more on clocks and how clock speeds work, see this fundamentals page.
- **Transfer Rate:** This refers to the number of times per second that data is transferred across the interface. This is only the same as the clock speed of the bus if single transition (conventional) clocking is used. Faster SCSI implementations now use double transition clocking, and this means the transfer rate (in millions of transfers per second) will be double the clock speed in MHz.
- **Throughput:** This number represents the theoretical maximum amount of data that can be moved across the SCSI bus, and is measured in millions of bytes per second (MB/s). On a narrow bus, throughput and transfer rate are the same, because each transfer is of 8 bits (one byte). But for a wide bus, throughput is double transfer rate, because each transfer is of 16 bits--two bytes.

Now that we understand all of that--we *do* understand it, right? :^)--we can look at the various bus speeds used in the SCSI world and understand what they mean. The table below shows all of the bus speeds used for parallel SCSI. (You may also find that looking at the table makes more clear the relationship between clock speed, transfer rate and throughput):

Standard-Defined Bus Speed	**Common Signaling Speed Name**	**Clock Speed (MHz)**	**Clocking**	**Transfer Rate (Mtransfers/s)**	**Throughput (MB/s)**	
					Narrow (8-bit)	**Wide (16-**

						bit)
SCSI-1	"Regular"	5	Single	5	5	--
Fast	"Fast"	10	Single	10	10	20
Fast-20	"Ultra"	20	Single	20	20	40
Fast-40	"Ultra2"	40	Single	40	40	80
Fast-80(DT)	"Ultra3" or "Ultra160"	40	*Double*	80	--	160
Fast-160(DT)	"Ultra320"	80	*Double*	160	--	320

Note: The "(DT)" in "Fast-80(DT)" and "Fast-160(DT)" represents the fact that this suffix is sometimes attached to represent the use of double transition clocking for those interfaces.

As you can see, the use of double transition clocking and wide buses means that the numbers in the latest transfer modes do not refer to the actual speed of the bus at all. The "160" in "Ultra160" represents the maximum throughput of such devices, but the clock speed is "only" 40 MHz.

Finally, I must include my standard disclaimer: we are discussing *interface* transfer rates here. These represent only the maximums that data can be transmitted across the interface under theoretical conditions. The big numbers that are popularly discussed ignore command overhead and other inefficiencies, so you will not actually get a full 160 MB/s on an Ultra160 interface. Also, remember that true performance will be limited by the speed of the devices on the interface. Simply increasing the speed of the interface is not enough to really improve performance unless the interface was already the limiting factor (such as if multiple drives were saturating it). See here for more on this issue. Also remember that the maximum throughput of any SCSI device will be limited by the throughput of the host adapter's system bus interface.

Next: Bus Parity and Cyclic Redundancy Checking (CRC)

Bus Parity and Cyclic Redundancy Checking (CRC)

As mentioned in a few places in this discussion of SCSI, parallel buses can have signal integrity problems, especially if used on long cables or at high signaling speeds. To help ensure that the data sent from one device arrives intact at its destination, various SCSI buses use two different data protection methods.

The first technique is *SCSI bus parity*. The parity method uses an extra bit for each eight bits of data, which is computed by the sending device so that the sum of all the "ones" in the nine bits taken together is either *odd* or *even*--one is chosen as the standard for the interface, and for SCSI odd parity is used. At the receiving device, the data is checked to see if the total is still odd; if an even number of "ones" is seen, this means there was a data corruption problem (because one bit is the wrong value somewhere) and the sender is signaled to retransmit. This simple data protection method is not unique to SCSI; it has also been used for years for serial communications and in memory circuits. For more details on parity, including examples showing how it works, see this comprehensive discussion in the memory section.

Note: SCSI parity is almost universally supported, but some very early host adapters may not work with parity checking enabled; to allow for this, many drives include a jumper to disable parity operation.

SCSI parity is useful, but is limited in its effectiveness, especially for very high transfer rates. It cannot detect if two bits in a given byte of data flip, for example. To further safeguard data, the SPI-3 standard introduced *cyclic redundancy checking* or *CRC* to the SCSI world when double transition clocking was introduced, allowing 160 MB/s data throughput on the bus. CRC is another technique that is not new, just "new to SCSI". It has been used in a variety of places in the computing world for decades, for example, in modems.

CRC is a bit difficult to describe in brief terms, and a full explanation is beyond the scope of this site. In a nutshell, it is a more robust way of checking for data corruption that can occur anywhere in a transmitted data message. A special algorithm is used that calculates a binary code as a result of arithmetic operations on the data; this is called a *cyclic redundancy code* (also abbreviated *CRC*). This code is sent along with the data over the bus. The recipient runs the same computation on the data and checks to see if it gets the same value that the sender computed; if there is any difference then an error occurred. In fact, modern CRC implementations don't actually run the computations using formulas, but rather use pre-created tables to speed up the process. This is sort of like using "multiplication tables" as you might have memorized as a child, instead of doing the computation from scratch each time.

At any rate, the bottom line is that CRC does a much better job of protecting data transmitted on the bus, especially at high signaling speeds. It can in theory be used in conjunction with parity, as they are independent. However, once CRC is being used, parity is somewhat unnecessary, except perhaps for compatibility with older hardware. CRC is one of the "optional" features of Ultra3 SCSI, and is a required feature for hardware meeting the Ultra160 or Ultra160+ specifications.

Tip: If you really want the "full scoop" on CRC, *try this site*. Be warned that it is definitely not for the technically faint of heart. :^)

Next: Command Queuing and Reordering

Command Queuing and Reordering

SCSI is often described as being "advanced", or is called an "intelligent interface". One of the reasons for these descriptions is that SCSI hardware incorporates features that improve overall system performance, where simpler interfaces such as IDE/ATA do not. One of these techniques is a special feature that allows for concurrent, multiple requests to devices on the SCSI bus. This feature is called *command queuing and reordering*; sometimes the name is given as *tagged command queuing*. It was first introduced in the SCSI-2 standard.

Traditionally, a simple interface like SCSI-1 or IDE/ATA will allow only a single command to be outstanding at a time to any device. This means that once a particular command is sent to a device, any other commands must wait for the first one to be completed, which slows down performance. *Command queuing* allows a device to accept as many as 64 or even 256 concurrent commands. The commands can also come from different originating devices. *Command reordering* allows a device that has multiple commands outstanding to fill them "out of order", meaning, not necessarily in the order that they were received.

For a very simple SCSI bus, such as a single hard disk on a host adapter in a desktop PC, command queuing and reordering may not make a particularly huge difference in performance. The reason is simply that there aren't that many concurrent processes running, and not a great deal of activity on the bus. This feature really comes into its own in a multiple-device, multitasking environment, such as that experienced by a shared server. In that environment, command queuing and reordering will improve performance significantly, by allowing devices to accept multiple simultaneous requests from different users, and fill them in the most efficient manner.

This is very important for devices like hard disks, which are very slow compared to the rest of the system. If commands are processed only as they are received, a great deal of time may be wasted while the hard disk's mechanical components move past a physically close piece of data that will be needed one or two requests "down the road". For a more thorough explanation of how drives can improve performance by reordering commands, see this discussion.

Next: Negotiation and Domain Validation

Negotiation and Domain Validation

SCSI hardware supports many different speeds, and newer, faster hardware is generally backwards-compatible with older, slower devices. You can use a host adapter capable of 160 MB/s throughput with drives that can only support 20 MB/s transfers, or vice-versa. This leaves an obvious question: how does each device determine what speeds the others on the bus are capable of? Without knowing this, senders can't figure out how fast receivers can handle data being sent.

Since this is so important, the SCSI protocols build in support for a method by which the host adapter can interrogate all devices on the bus to find out what speeds they support. This process is called *negotiation*, and is one of the first tasks performed by the SCSI host adapter when the system power is applied. Under conventional SCSI rules, this negotiation is done with each device; the host adapter records the maximum transfer speed that each device claims to support, and then uses that information when the device is accessed.

This works great in theory, but there's a problem with it: theory doesn't always translate into practice, especially when the technology "pushes the envelope" with high-speed signaling. For example, even if the host adapter can support Ultra160 transfers and the device says it can as well, this doesn't mean that 160 MB/s signaling is actually possible on the bus. Perhaps the cabling being used is inferior or too long, or there's a problem with a terminator, or the system is in a particularly electrically noisy environment. Regular negotiation just "trusts" that everything will work at the speed the hardware decides is possible, but it may not actually work. If there are difficulties, they may manifest themselves in the form of data errors or reliability problems.

To improve negotiation, the SPI-3 standard introduced a new feature called *domain validation*, sometimes abbreviated *DV*. This feature basically adds a verification step to the normal negotiation procedure (note that "domain" is another word for a SCSI channel or bus). After a device tells the host adapter that it is capable of transfers at a particular speed, the host adapter tests the device by sending write requests to the device's internal buffer at that speed. The data just written is then read back and compared. If the data is different, or if parity or CRC errors occur during either the read or the write, the host adapter knows that communication at that speed is not reliable. It will then retry at the next lower speed, and continue until reliable operation is established. (If this sounds similar to the way that two regular analog modems determine a communications speed, that's because it is!)

Domain validation is one of the five "optional" features of Ultra3 SCSI, and is a required feature for hardware meeting the Ultra160 or Ultra160+ specifications. This feature may be expanded in the future to include more frequent validation during the operation of the system, since over time errors may occur on a channel that worked fine when the system was first powered up.

Next: Quick Arbitration and Selection (QAS)

Quick Arbitration and Selection (QAS)

During the time when the system is running, the SCSI bus is generally either active or idle. If active, the bus is busy transmitting data from one device to another; if idle, it is available for a device to begin sending a command or data. When a device decides it wants to use the bus, it "bids" for control of the bus. It is also possible that other devices on the bus will want to use it at the same time, so they too may "bid" for control. A specific method is used to resolve these requests and decide which device gets to use the bus first; this is based to some extent on the devices' respective priority levels. This process is called *arbitration*.

While arbitration works fine in regular SCSI configurations, it introduces overhead. During the time that arbitration is going on, no data is being transferred on the bus, so it makes sense that doing this faster will allow improved performance of the entire SCSI subsystem. To this end, the SPI-3 standard defined a feature that reduces the overhead required for arbitration. This feature is called *quick arbitration and selection* or *QAS*. You may also see it called by the name it carried during development, *quick arbitration and select*; IBM calls it *quick arbitration select* and Adaptec, simply *quick arbitrate*. These are all different names for the same feature.

In a nutshell, QAS works by reducing the number of times arbitration must occur on the bus. When the feature is used, a device waiting for the bus can grab it more quickly after the last device on the bus sends the signal that it is done, without having to begin a new arbitration process. Provision is made in the specification to ensure that one device does not "dominate" the bus by "unfairly" blocking out other devices that may be of a lower priority or may not implement QAS.

Quick arbitration and selection is one of the five "optional" features of Ultra3 SCSI. It was not included as one of the required features for hardware meeting the Ultra160 specification, but is present in Ultra160+ devices.

Next: Packetization

Packetization

While the SCSI interface is widely implemented on high-end hardware due to its flexibility and high performance, its complexity does mean that some of its potential performance is lost to overhead. In an effort to improve SCSI bus performance by reducing overhead, the SPI-3 SCSI standard describes a new feature that is generally called *packetization* or *packetized SCSI*.

Packetization is a technique whereby some of the phases that are involved in setting up a command request and data transfer are combined. For example, under traditional SCSI interfacing, several different types of information are sent over the bus separately: commands, data, status messages and so on. With packetization, these are grouped together into *packets* (also called *information units*) and sent as a single entity. This reduces some of the wasted bus cycles normally sent on managing all the individual transfers in regular SCSI.

Packetization is one of the five "optional" features of Ultra3 SCSI. It was not included as one of the required features for hardware meeting the Ultra160 specification, but is present in Ultra160+ devices. It may also be part of the requirements for Ultra320 SCSI when that specification is complete.

Next: SCSI Protocol Map

SCSI Protocol Map

The table below shows how the two most important protocol characteristics--bus width and bus speed--map into the different names for SCSI transfer modes and feature sets that are commonly seen in the SCSI world. Links are included to each of the SCSI transfer modes or feature sets. The throughput for each SCSI variety is also shown:

Signaling Speed	Narrow		Wide	
	Mode	Throughput (MB/s)	Mode	Throughput (MB/s)
SCSI-1	SCSI-1	5	Wide SCSI	10
Fast	Fast SCSI	10	Fast Wide SCSI	20
Fast-20	Ultra SCSI	20	Wide Ultra SCSI	40
Fast-40	Ultra2 SCSI	40	Wide Ultra2 SCSI	80
Fast-80(DT)	--		Ultra3 SCSI, Ultra160(/m) SCSI, Ultra160+ SCSI	160
Fast-160(DT)	--		Ultra320 SCSI	320

Next: Summary of SCSI Protocols and Transfer Modes

Summary of SCSI Protocols and Transfer Modes

For easier comparison, the chart below shows all of the different SCSI transfer modes and feature sets, along with their key characteristics. For your clicking pleasuer, I've also hyperlinked everything that wasn't nailed down ;^)

Transfer Mode	**Defining Standard**	**Bus Width (bits)**	**Bus Speed (MHz)**	**Through-put (MB/s)**	**Special Features**	**Cabling**	**Signaling Method**	**Maximum Devices Per Bus**	**Maximum Cable Length (m)**
"Regular" SCSI (SCSI-1)	SCSI-1	8	5	5		50-pin	SE	8	6
							HVD	8	25
Wide SCSI	SCSI-2	16	5	10		68-pin	SE	16	6
							HVD	16	25
Fast SCSI	SCSI-2	8	10	10		50-pin	SE	8	3
							HVD	8	25
Fast Wide SCSI	SCSI-2	16	10	20		68-pin	SE	16	3
							HVD	16	25

	SCSI-3 / SPI	8	20	20		50-pin	SE	8	1.5
								4	3
							HVD	8	25
ltra	SCSI-3 / SPI	16	20	40		68-pin	SE	8	1.5
								4	3
							HVD	16	25
l	SCSI-3 / SPI-2	8	40	40		50-pin	LVD	8	12
								2	25
							HVD	8	25
ra2	SCSI-3 / SPI-2	16	40	80		68-pin	LVD	16	12
								2	25
							HVD	16	25
l	SCSI-3 / SPI-3	16	40 (DT)	160	At least one of Fast-80, CRC, DV, QAS, Packet	68-pin	LVD	16	12
								2	25

Ultra160(/m) SCSI	SCSI-3 / SPI-3	16	40 (DT)	160	Fast-80, CRC, DV	68-pin	LVD	16	12
								2	25
Ultra160+ SCSI	SCSI-3 / SPI-3	16	40 (DT)	160	Fast-80, CRC, DV, QAS, Packet	68-pin	LVD	16	12
								2	25
Ultra320 SCSI	SCSI-3 / SPI-4	16	80 (DT)	320	Fast-160, ?	68-pin	LVD	16	12
								2	25

Some notes on this table:

- "(DT)" means that transfer mode uses double transition clocking.
- Throughput numbers are in decimal megabytes per second.
- To keep the table from getting cluttered, I've only mentioned current cabling, not obsolete cables.
- The number of devices includes the host adapter.
- For Ultra and faster speeds, the maximum length of cable for some signaling types depends on the number of devices on the chain; thus, the multiple rows.
- Ultra320 specifications are not final as of this writing.

Note: The number of devices (8 or 16) includes the host adapter.

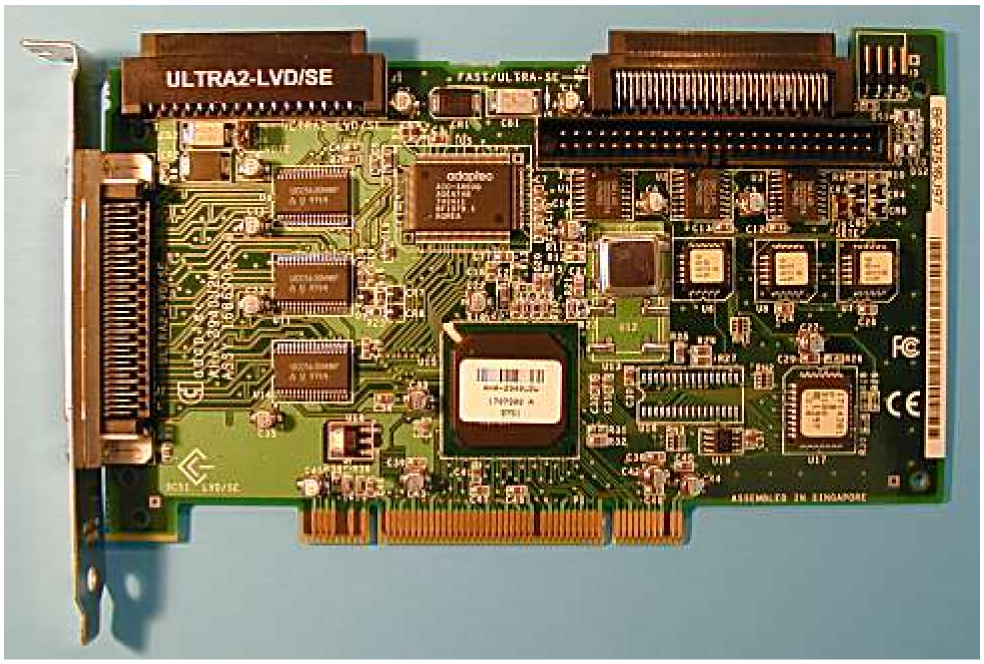
Next: SCSI Host Adapters

SCSI Host Adapters

Most IDE/ATA hard disks are controlled today by integrated IDE controllers that are built into the chipset on the motherboard. The SCSI interface is not, for the most part, controlled by built-in motherboard SCSI controllers, although some are and this is growing in popularity. Most systems require the addition of a special card which serves as the interface between the SCSI bus and the PC.

This device is called a *SCSI host adapter*, or alternately a *host bus adapter* (sometimes abbreviated *HBA*). It is sometimes called a *SCSI controller* or even just a *SCSI card*, though these are technically incorrect names. They are not accurate because SCSI is a systems-level interface, and every device on the bus has its own controller. Logically, the host adapter is just a SCSI device like any other. Its job is to act as the gateway between the SCSI bus and the internal PC's I/O bus. It sends and responds to commands and transfers data to and from devices on the bus and inside the computer itself. Since it is inside the PC, of course, the host adapter really *isn't* the same as the other devices on the bus--it's sort of a "first among equals", if you want to think about it that way. ;^)

Since SCSI is a very "intelligent" interface--meaning it has a lot of capabilities and the devices on it are able to interact in advanced ways--many SCSI host adapters have evolved rather exceptional capabilities, and can act in many ways to improve performance. In some ways, the host adapter is the key to good SCSI implementation in the PC, since no matter how advanced the peripherals are that you attach to the bus, everything goes through that host adapter.



A PCI-based Adapter Wide Ultra2 SCSI host adapter. Note the numerous connectors, which allow several chains of internal and external devices to be attached to the host adapter. You can see a 50-pin connector for internal devices on the right side near the top, pointing straight out towards you. There are two high-density Wide Ultra2 connectors on the top of the card. There is also one high-density external connector on the expansion slot insert on the left-hand side of the card. The PCI interface connector is on the bottom.

In this section I take a reasonably detailed look at host adapters. I focus primarily on the most important attributes and features that someone considering a SCSI setup might look for. This includes a discussion of device support, interfacing, connectors, resources and compatibility.

Tip: Motherboard support for SCSI is actually on the rise, especially in higher-end systems, as SCSI becomes more "mainstream". It is still not common to find it in most motherboards because it increases cost, and most people still are not using SCSI. If you are building a new PC and want to go with SCSI, consider a motherboard with an integrated SCSI host adapter. When selecting such a motherboard, however, it is *critical* to pay specific attention to what SCSI transfer modes and feature sets the motherboard will support. While most built-in SCSI controllers can be disabled, having to buy a SCSI host adapter six months after you buy a SCSI-capable motherboard--because the motherboard-based controller doesn't do what you need it to--is just a waste of time and money.

Next: Adapter Types and PC Bus Connections

Adapter Types and PC Bus Connections

In selecting a SCSI host adapter, one of the first decisions to be made is what type of host adapter you want. There are many different kinds of SCSI host adapters on the market, and they vary in cost and capabilities dramatically. Many lower-end adapters are designed specifically are designed to keep costs down to allow easy, inexpensive access to SCSI devices like scanners or CD-RW drives. Higher-end devices provide more capabilities and performance for users who require a full-featured implementation for hard disks and other performance drives.

A key distinguishing characteristic between various host adapter models is the type of system bus the card is designed for. SCSI host adapters have been made for all of the common PC I/O buses, including ISA, EISA, VLB, MCA and PCI. You obviously need to choose a host adapter that matches the system bus(es) in your machine. Until recently, motherboards that featured both PCI and ISA slots were common, giving you a choice. As is the case with most interfaces that have a significant impact on performance, an ISA-based card is generally a bad idea since this will greatly limit the performance of the bus. You cannot efficiently use even something like Fast SCSI through an ISA-bus-based host adapter, because ISA cannot handle more than about 8 MB/s of data throughput. (Or rather, you can use it, but you won't get all the performance possible.) Of course, by today's standards Fast SCSI isn't all that fast, and higher-performance transfer modes are even less suitable for ISA. However, ISA cards can be perfectly acceptable for slow SCSI devices such as scanners, Zip drives, optical drives and so on.

After 20 years, the ISA bus is finally going the way of the dodo; Intel and Microsoft are hard at work trying to kill off this old system bus interface, and many systems come only with PCI slots, making a PCI host adapter the only real option. Of course, not all PCI cards are alike; you certainly still have plenty of decisions to make even within PCI. Here too, low-end, inexpensive cards are more suitable for simple applications, and more expensive cards are appropriate for performance-demanding applications.

As I mentioned above, a very important performance issue concerning SCSI host adapters and the system bus they use is the throughput of the bus. If the throughput of the system bus is less than the maximum throughput of the SCSI channel, SCSI performance will be limited to whatever the bus's maximum rate is. Until recently, this wasn't much of a concern as long as a PCI host adapter was used, because PCI had more than enough "overhead" to handle any SCSI bus. In the last few years, however, with SCSI channels continuing to increase in speed, even the performance of regular PCI is now reaching a limiting point. The maximum practical bandwidth of regular (32-bit, 33 MHz) PCI is a little over 100 MB/s, and the newest SCSI devices use Ultra160 SCSI, capable of well over 100 MB/s of throughput. To get maximum performance from such an interface, regular PCI is not sufficient.

To this end, some higher-end Ultra160 and faster SCSI host adapters are designed to use new enhancements to traditional PCI. These include 64-bit PCI, capable of throughput of over 200 MB/s, and also the new PCI-X bus, which promises performance of up to 1 GB/s. Cards using 64-bit PCI are readily available and are backwards-compatible with regular 32-bit PCI, so they can be used on both newer systems with 64-bit PCI slots, or systems that have only 32-bit slots.

Note: As I discuss at the end of the page on SCSI bus speeds, one should always remember that SCSI throughput specifications are for the SCSI chain as a whole, not individual devices. 160 MB/s is the maximum for all the devices on an Ultra160 bus. The limits of regular PCI only become an issue if one is using enough devices simultaneously to push the limits of the bus--such as if RAID is being used. If you are just using one or two Ultra160 devices, regular PCI is probably sufficient for your needs.

Another reason to use PCI is that most newer host adapters that run on the PCI local bus support the use of bus mastering. This can be a very important feature, as it allows for more efficient transfer of data from the host adapter to the system memory. Full performance from high-end SCSI chains requires bus mastering support.

Next: Protocol Support

Protocol Support

Obviously, a key issue in selecting a host adapter is support for the particular transfer modes and feature sets that you want to use. The host adapter must be able to support all of the protocols that you want to use on the bus. If you want to use wide SCSI, the adapter must support 16-bit operation; if you want to use Ultra160 SCSI hard disks, your host adapter must be capable of supporting this speed. Nothing too surprising here.

Typically, host adapters are backwards compatible with older devices, so you can run a newer drive on an older, slower host adapter--you just give up some of the performance of the drive in doing so. Similarly, newer host adapters will support older, slower devices, providing that they are properly configured. See this page for more on compatibility between different SCSI flavors.

Of course, host adapters that support faster transfer modes are generally more expensive than ones that only support slower devices. Remember though, that if you purchase a cheaper, less capable host adapter, you will be limiting your expansion capabilities should you later decide that you *do* want to use the faster mode. Whether the more capable host adapter is worth the extra money depends on your situation and the chances that you will want or need to upgrade in the future.

Next: Signaling Type Support (SE, HVD, LVD)

Signaling Type Support (SE, HVD, LVD)

With the arrival of Ultra2 SCSI, the SCSI world now uses three different types of electrical signaling: conventional, single-ended SCSI, high-voltage differential (HVD) SCSI, and low voltage differential (LVD) SCSI. The host adapter used in a system must be electrically compatible with the drives that are to be used.

Let's deal with high voltage differential first. This is the "oddball"; it was never very popular in the market because of its expense and the fact that it is not compatible at the electrical level with the more popular single-ended devices. To use high voltage differential drives, you need to use a host adapter specifically designed to support HVD signaling. Adapters do exist to allow HVD drives to run on SE SCSI chains, but this is an expensive solution.

Warning: Do not attempt to connect HVD drives on a SCSI chain with non-HVD devices, or the latter may be damaged.

Since low voltage differential signaling is required for the newest, fastest devices, host adapters supporting LVD are taking over the market. Host adapters that support only SE signaling are still available, but these are severely restricted in terms of the performance and cable lengths they will allow, so they are generally used only for devices other than hard disks.

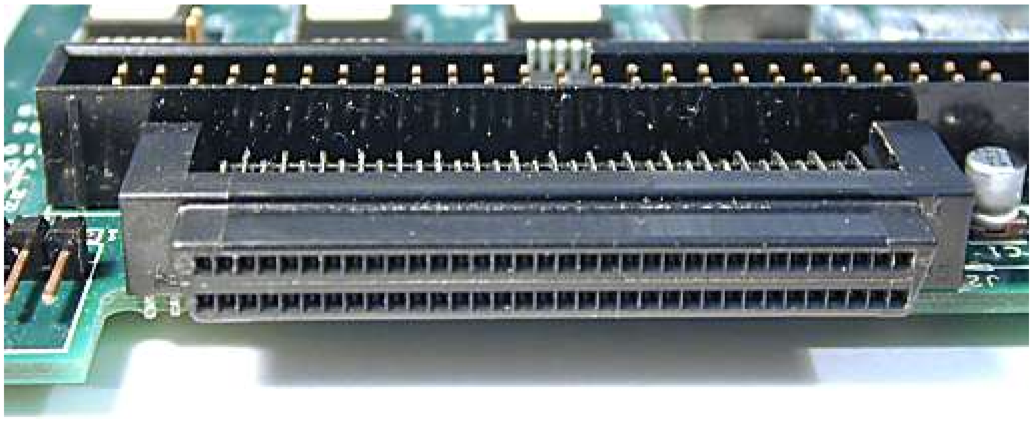
LVD also creates an issue for people who want to use both high-speed hard disks and also slower devices on the same host adapter. LVD and SE are electrically compatible--assuming that the LVD devices are multimode capable (LVD/SE or LVD/MSE)--so it isn't the same issue as with mixing SE and HVD. The problem is that LVD only works if *all* the devices on the SCSI bus are running in LVD mode. If you put a single-ended drive on a SCSI chain with an Ultra160 hard disk, the hard disk will run at no greater than Ultra speeds, knocking down performance. You will also see your maximum cable length reduced from 12m to either 3m or 1.5m!

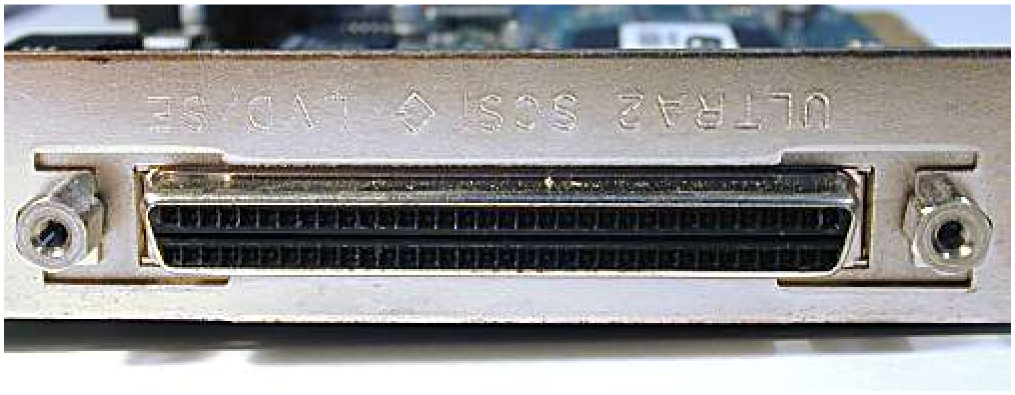
To get around this problem, many SCSI host adapters that support LVD modes (Ultra2 SCSI and higher) generally include separate support for running devices at Ultra or slower speeds in single-ended mode. This is implemented either through two distinct segments on the same SCSI channel--using translater chips that allow the SE and LVD devices to share data without interfering with each other electrically--or separate, independent SCSI channels within the same card. (See here for details on multiple segments and channels.) Be aware that if your SCSI host adapter does not support this feature, you will not be able to run Ultra speed or slower devices and Ultra2 or faster devices on the same machine. Of course, you can choose to use two SCSI host adapters, but this causes other complications...

Next: Connectors

Connectors

SCSI is a bus that supports both internal and external devices. To support these two types of devices, most SCSI host adapters come with both internal and external connectors. Internal connectors are usually mounted along the top edge of the SCSI host adapter, and are used for the ribbon cables employed for internal SCSI devices. External connectors are mounted along the outside edge of the host adapter (the part accessible from the back of the PC when the card is inserted into a system bus slot.)





Internal (above) and external (below) connectors on a Wide Ultra2 SCSI

host adapter. In the upper photo you can see two connectors; facing you is a 68-pin (wide) high-density connector, and facing up is a 50-pin (narrow) "regular
density" connector. In the lower photo is a 68-pin (wide) high-density connector.

The exact type of connectors provided on any given card depends on its design, and more specifically, the type of SCSI it is intended to support. A card that is designed to support narrow devices will have narrow (50-pin) connectors, while cards that are built to run wide devices will have 68-pin connectors. There are also different types of each of these two sizes of connector; for example, an older or lower-end host adapter may use the older high-density 68-pin connectors while high-end Ultra160 card may use the smaller very-high-density (VHDCI) connectors.

Obviously, when purchasing a host adapter check to make sure it has the connectors you need for your devices. Remember that adapters are available to interface devices that use different styles of connectors, but they can add significant cost to a SCSI implementation. For much more information on cable and connector issues, see this section of the site.

Next: Caching

Caching

Many higher-end SCSI controllers have built-in caches. The idea behind a cache is to use high-speed memory to hold recently retrieved results, to save time if the results are needed again in the near future. This improves performance because most SCSI devices are drives, much slower than memory in relative terms. Caching is a concept that is used extensively within the PC world; it is found in CPUs, hard disks, optical drives and a variety of other devices. To read more about caching in general terms, see this section.

Caching that is done by the SCSI controller adds an additional caching level that exists, logically, in front of the cache (buffer) that resides within SCSI hard disks or other components. When data is requested from a device on the SCSI bus, the host adapter sees if it is already in its internal cache and if so, returns the results much more quickly, saving transaction bandwidth on the SCSI bus at the same time. This of course improves performance over a system that does not have this type of cache.

The amount of cache memory on a host adapter depends on the model; some are user-upgradeable. Note that there are diminishing returns associated with cache memory; each megabyte you add helps performance less than the previous megabyte did...

Next: Multiple Segment and Channel Support

Multiple Segment and Channel Support

In its simplest form, a host adapter provides support for a single SCSI chain: that is, a single set of devices that are all connected together on the same SCSI bus. This is the way that many older and low-end SCSI host adapters work. They are fine for simple implementations, but are too limiting for complex SCSI setups. Especially with modern systems that need to use both LVD and single-ended devices, an adapter with support for just a single segment is insufficient for maximum performance. To expand capabilities, host adapter manufacturers make cards that support multiple *segments*, multiple *channels*, or both.

A *segment* is an electrically-isolated "piece" of a SCSI bus; a single bus can be made up of one or more segments. Cards that implement multiple segments allow for more flexibility because the segments are electrically separate. Each segment can have a cable as long as the normal maximum cable length allowed for that particular type of SCSI, for example. One segment can use an internal cable within the PC and another an external cable. It's important to remember though that two segments on a single channel are *logically* considered to be part of the same SCSI bus even if they are *electrically* separate. This means all devices on all segments must have unique IDs, and that maximum bandwidth is shared between all devices on all segments that make up the bus.

The most expensive host adapters go beyond multiple segment support and actually have multiple *channels*. These are similar in concept to the way an IDE/ATA controller typically has two channels. Each channel is completely independent of the other, both electrically *and* logically. This means the two run in parallel with each other: you get support for twice as many devices, and twice as much throughput. In essence, a card with two channels is two host adapters in the same package. For example, an Ultra160 host adapter with dual channels will support 30 drives (16 per channel less one each per channel for the host adapter) and theoretical throughput of up to 320 MB/s (160 MB/s per channel). Note that each channel can itself have more than one electrical segment.

Host adapters that support multiple channels are not really needed for most applications, especially if already using high-performance SCSI like Ultra160; they are more common in servers than desktop PCs. Multiple segments, however, are commonly found even in desktop SCSI cards. One common use for multiple segments is to allow independent use of LVD and SE devices on the same host adapter without causing the LVD devices to degrade to SE operation. See this page for more on this important feature of Ultra2 and faster host adapters. Some cards use multiple channels to isolate LVD and SE, which is probably even better (though it may be more expensive.)

Next: RAID Support

RAID Support

SCSI is the interface of choice for servers and high-end workstations, where both performance and reliability are critical. One of the most important ways that performance, data integrity and reliability are improved in modern PC systems is through the use of *redundant arrays of inexpensive disks*, or *RAID*. This term simply refers to the use of multiple hard disks in an array, with data spread across the disks. Accessing multiple disks simultaneously allows for faster performance; the optional use of redundancy allows for protection against hardware faults. You can read all the details about RAID here.

Most higher-end RAID solutions use SCSI, so support for RAID is commonly found in SCSI host adapters. In practice, these are not usually sold as "SCSI host adapters with RAID support" but rather are considered as a separate product line: "SCSI RAID controllers". For more information on RAID controllers, see this section. If purchasing a RAID controller, be sure to read all RAID controller specifications carefully, in addition to the information I just referenced--RAID cards vary widely in terms of features and implementation requirements.

Next: Drivers and Compatibility Issues

Drivers and Compatibility Issues

It's important to ensure that your operating system will support whatever host adapter you decide to use. Selecting a quality card from a well-known company is key to success in many cases. You want to make sure that the card comes with configuration software and drivers that will support whatever operating system and applications you are using. Most well-known cards will either be supported natively in Windows (and other operating systems) or will be provided with good drivers.

There are, in rare cases, problems with compatibility between older devices and cards, due to some loose interpretations of the SCSI standards. This has been pretty much cleaned up with the newer devices, however. Other compatibility difficulties may result from the mixing of devices using different types of SCSI transfer modes or feature sets; see here for more details.

Warning: One thing that is important to know is that different SCSI host adapters may use different addressing or translating methods to access data on hard disks to which they are connected. This means that switching host adapters can render the contents of the hard disk inaccessible. In some cases the disk must be reformatted after the new host adapter is installed. See here for more on this.

Next: Manual vs. Automatic Configuration

Manual vs. Automatic Configuration

Host adapters vary in terms of the methods that are used for configuring them. Older and cheaper cards, particularly ones that use the ISA bus to connect to the PC, typically require the use of hardware jumpers for configuration tasks such as setting the SCSI device ID for the host adapter, enabling or disabling termination, and so on. These are relatively inconvenient because making changes requires opening up the PC, and in some cases, pulling out the card to tinker with it.

Newer cards, especially those that use the PCI bus, are generally configured through software. This is done either using a separate configuration utility, or the built-in SCSI BIOS, a hardware program that resides on a chip within the host adapter (much like the system BIOS in concept, but dedicated to the SCSI card, not the system as a whole.) Some cards may use both. Some better cards may also automatically configure certain options, such as termination, by detecting which connectors are in use, for example.

These issues are discussed in more detail in the section on configuration. Note that when I talk about configuration here I am speaking of configuring the SCSI bus, not the host adapter itself; see this page for more on that subject.

Next: Resource Usage

Resource Usage

From the perspective of the PC as a whole, SCSI host adapters are *expansion devices*, since they plug into a system bus and represent a peripheral device on the system bus. (Some motherboards have integrated SCSI host adapter chips, but these are logically similar to separate host adapters even if no distinct physical card is used.) Host adapters typically require several different system resources, depending on the system bus that the host adapter is designed for, and the method it is using for transferring data over the system bus.

The following resource types are typically used on various adapters:

- **Interrupt Request Line (IRQ):** All SCSI host adapters use an interrupt request line or IRQ. The most commonly used ones are 9, 10, 11 or 12. IRQs 14 or 15 can usually be used as well if one or the other of the IDE/ATA channels in the system are not being used (see here for details on IDE/ATA resource usage). It should be noted that PCI-based host adapters will not require an explicit IRQ assignment. Rather, they will use one the system IRQ mapped to whatever PCI slot they are placed into. PCI also supports IRQ sharing. See here for more.
- **DMA Channel:** Many older host adapters based on the ISA or VLB buses use DMA channels to permit the transfer of data directly from SCSI devices to system memory. Usually DMA channels 1, 3 or 5 are used. PCI-based host adapters typically make use of PCI bus mastering to improve performance, which is a separate type of DMA that does not use regular ISA system DMA channels.
- **I/O Address:** The I/O address is used as the place through which data is transferred to the system. There are several ranges that are used by some SCSI cards.
- **BIOS ROM Memory:** The SCSI BIOS that contains the commands for controlling the host adapter typically takes up a 4000h address

location in the upper memory area. This is usually one of the five 4000h spaces in the address range of CC000h to DDFFFh.

Most newer SCSI host adapters, especially those using the PC bus, support the Plug and Play initiative. Plug and Play allows the system to configure resources for the host adapter automatically in many cases, reducing configuration difficulties. (Note that I am speaking here of Plug and Play at the system level, which deals with system resources. This is different than so-called Plug and Play SCSI, which is similar in concept but is applied to dynamically allocating SCSI device IDs on the SCSI bus, not PC system resources.)

Next: SCSI Cables and Connectors

SCSI Cables and Connectors

There are many different aspects about SCSI that can be confusing to someone new to the technology--and even someone *not* new to it. :^) Of all of the aspects of SCSI that sometimes cause a bit of difficulty, cables and connector issues are probably the worst. Unlike the IDE/ATA world, where there are a handful of different cable types, with SCSI there are literally *dozens* of different types of cables! It is difficult to even describe all of the options available. This is a result of the flexibility of the SCSI interface--more choice means more options, and hence, more decisions. :^)

In this section, I describe the cable and connector hardware most commonly seen on the SCSI interface. There are many different types of cables, and many types of connectors, and to some extent they can be "mixed and matched"--meaning that you may find different types of cables for each connector type and vice-versa. Therefore, I begin by first discussing general cabling issues, and then describing the most common technologies found for cables, and then the same for connectors. I then cover the specifics of the three most common general categories of cables: narrow, wide, and LVD cables, and provide pin-out listings for the signals they use as well. I then explain the single connector attachment method of connecting SCSI drives, talk about adapters, and discuss different ways that the SCSI bus is terminated.

Note: I have to state "up front" that I do not describe every type of SCSI cable and connector that exists in this section, just the most common ones. There are *so* many types of cables, including some that are created for very particular applications, that I couldn't even begin to describe them all. By reading this section you will know most of what you need to know about SCSI cables and connectors. However, if you are looking to buy SCSI cables, the most important thing you should look for is a *quality vendor* that will help you choose the correct cable for your application. SCSI cables can be expensive; a reputable vendor will be able to ask the correct questions to ensure that you get the hardware you need.

Next: General Cable and Connector Issues

General Cable and Connector Issues

The main reason why there are so many types of SCSI cables is simply that there are so many types of SCSI--and so many different ways of implementing them. This great flexibility is actually one of the key strengths of the interface. The design of any SCSI cable is based on a combination of different attributes chosen to implement a particular kind of SCSI bus.

Each SCSI cable must meet the specific electrical requirements associated with the SCSI signaling speeds and methods it supports. This refers not just to obvious matters--such as how many pins are on a particular connector type, or which signals are carried on which wires--but the more complex factors that are the domain of electrical engineering professionals. For example, the thickness of each wire in a cable, the characteristic impedance of the cable, materials used for the wires, connectors and covers, and so on.

The following are other factors that have an impact on the design of SCSI cables, as well as on the selection of cables to meet a particular application:

- **Cable Type:** SCSI is different from most PC interfaces in that it supports both internal and external devices. These use drastically different types of cabling, because the environment inside the PC is very different from that outside it. Both internal and external cables come in a variety of styles themselves. This is all discussed in more detail on this page.
- **Connector Type:** Different types of connectors are used for different kinds of SCSI. These are only partially dependent on the type of physical cable used; to some extent, connector types are "mixed and matched" with cable technologies to make particular cables. See here for more.
- **Cable Length:** The maximum length of a SCSI cable is dictated by the signaling type and signaling speed of the interface; this summary table shows the length limits for all the different types of SCSI. However, not all cables are built to the maximum length. Cables of all different lengths are made to suit different needs and budgets (most people don't need 12-meter-long cables for LVD devices, for example, even though they are legal.)
- **Number of Connectors:** Cables vary in terms of the number of connectors they include. Generally speaking, longer cables have more connectors, allowing more devices to be attached to the same SCSI bus segment. Specialized cables may have fewer connectors; for example, LVD cables can be 25m in length instead of the usual 12m if they are used "point to point"--just two devices on the cable.
- **Connector Spacing:** Some types of SCSI have limits regarding how closely two connectors can appear on the cable. If the cable has many connectors you may have to leave some of the connectors unused for maximum performance. In all cases, it is recommended that devices be evenly spaced across the cable.
- **Termination:** Some cables have a built-in terminator at the end of the cable while others require the addition of a separate terminator. See the page on termination for more.

- **General Quality:** The overall quality of a SCSI cable is very important, but is not something tangible that can be easily measured or quantified. Remember that not all cables are created equal. SCSI cables are often the culprits in problematic SCSI buses, so don't skimp on the quality of your cables.

Warning: Some companies sell cables with labels such as "SCSI-1 cable", "SCSI-2 cable" or "SCSI-3 cable". With the possible exception of "SCSI-1", these are extremely vague terms that do not tell you nearly enough about the cable to decide if it is the one you want. These terms refer to SCSI standards, which define SCSI *families*, not specific types.

Next: SCSI Cable Types

SCSI Cable Types

The term "SCSI cable" usually refers to a complete cable, including the wire, connectors and possibly a terminator as well. On this page I want to start by looking at the cable itself, the actual wires that make up the "overall cable". There are a number of different types of cables available; these are combined with various connector types to create specific cable implementations.

SCSI cables come in two distinct varieties: *external* and *internal*. External cables are used to connect SCSI devices that do not reside inside the PC, but rather have their own enclosures and power supplies; internal cables connect SCSI devices installed within the PC system box. These cables are totally different in construction, primarily because the external environment represents much more of a risk to data corruption. This means external cables must be designed to protect the data traveling on the cable. Internal cables don't have this problem because the metal case of the PC shields the components inside from most of the electromagnetic and radio frequency noise and interference from the "outside world". Thus, internal cables can be made more simply and cheaply than external ones.

Let's start by looking at external cables. These are commonly called *shielded cables* because they are made specifically to protect the data they carry from outside interference. They have a very specific design in order to ensure that data traveling on the cable is secured, including the following properties:

- **Twisted Pair Wiring:** All the wires in the cable are formed into pairs, consisting of a data signal paired with its complement. For single-ended signaling, each signal is paired with a "signal return" wire--a fancy name for a ground wire. For differential signaling, each "positive" signal is paired with its corresponding "negative" signal (see the description of differential signaling for an explanation of this). The two wires in each pair are then twisted together. This twisting improves signal integrity compared to running all the wires in parallel to each other. So an external narrow cable with 50 wires actually contains 25 pairs; a 68-wire cable 34 pairs. (This sort of wiring is also commonly used in other applications, such as network cabling, for the same reason.)

- **Shielding:** The entire cable is wrapped with a metallic shield, such as aluminum or copper foil or braid, to block out noise and interference.
- **Layered Structure:** The pairs of wires aren't all just tossed into the cable at random; instead, a structure of layers is used. The "core layer" of the cable contains the pairs carrying the most important control signals: REQ and ACK (request and acknowledge). Around that core, pairs of other control signals are arranged in a "middle layer". The outer layer of the cable contains the data and other signals. The purpose of this three-layer structure is to further insulate the most important signals to improve data integrity.

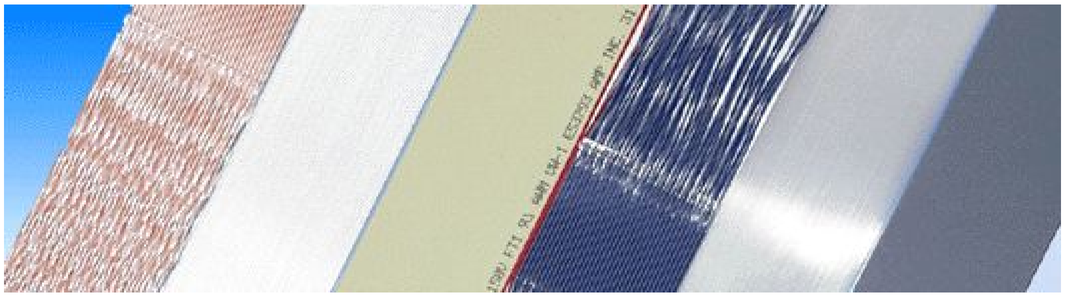
External cables have a round cross-section, reflecting the circular layers mentioned just above. Needless to say, these cables aren't simple to manufacture! All this precise engineering doesn't come without a cost: external SCSI cables are generally quite expensive. For internal cables all these special steps are not required to protect the data in the wires from external interference. Therefore, instead of special shielded, multiple-layer construction, internal devices use *unshielded cables*, which are flat ribbon cables similar to those used for floppy drives and IDE/ATA devices. These are much cheaper than external cables to make.

HCM 48213 - LVD COMPLIANT - (UL) TYPE CMG c(UL) 28 AWG 75C SHIELDED - 9/98 (B594U)

Close-up view of an external SCSI cable. Note the round shape of the cable's cross-section, and the labeling, which indicates that this is LVD-compliant, shielded cable, using AWG 28 conductors.

Original image © Computer Cable Makers, Inc. Image used with permission.

Even with internal cables, there are differences in construction (beyond the width issue, 50 wires for narrow SCSI or 68 wires for wide SCSI). One issue is the thickness of the wires used; another is the insulation that goes over the wires. Better cables generally use Teflon as a wire insulation material, while cheaper ones may use PVC (polyvinyl chloride; vinyl). Regular flat cables are typically used for single-ended SCSI applications up to Ultra speeds (20 MHz).



An assortment of different internal ribbon cables used for connecting SCSI hardware. Note that some are strictly flat cables, but the one on the far left and the one third from the right are partially flat and partially twisted pair cable.

Original image © CS Electronics
Image used with permission.

For Ultra2 or faster internal cables using LVD signaling, the poor electrical characteristics of cheap flat ribbon cables begin to become an issue in terms of signal integrity even within the PC. Therefore, a new type of internal ribbon cable was created for these cables, which actually combines some of the characteristics of regular internal and external cables. With these ribbon cables, pairs are twisted between the connectors on the cable--just like in external cables--but the ribbon remains flat near where the connectors go, for easier attachment. The return to pair twisting improves performance for high-speed SCSI applications, while increasing cost somewhat, though not as much as if external cables are used. This technology is sometimes called "Twist-N-Flat" cable, since it is partially flat and partially twisted-pair.

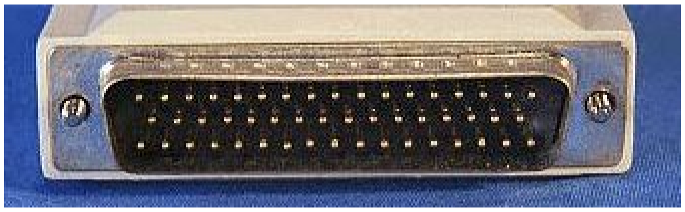
Next: SCSI Connector Types

SCSI Connector Types

Connectors are of course the physical devices that are used to attach a SCSI cable to a SCSI device. Several different types of SCSI connectors are used to construct SCSI cables. This is in itself unfortunate in a way; whenever there are multiple types of connectors for an interface, this means the potential exists for mismatched connectors between devices. Different connector types have evolved over the years as the SCSI interface has matured. In particular, the desire for *miniaturization* has been a driving force in the creation of new connector types--the oldest SCSI connectors were large, and creating smaller connectors improves the usability of SCSI cables and devices.

Below are the connector types most commonly seen used with SCSI cables in the PC world. Note that this list is not exhaustive, in part because there are several obscure variations used for some proprietary SCSI implementations. However, most of the cables you will find in the SCSI world use one of these connector types. The SCSI standards call different connector types "alternatives" (not really a good name since the "alternatives" describe different devices types and not really "choices" as that word implies). Since external and internal cables generally use different connectors, each has four different "alternatives". I'll begin with external connector types:

- **D-Shell (D-Sub, DD):** The earliest SCSI standard, SCSI-1, defined a 50-pin *D-shell* connector for narrow SCSI implementations. The name of this connector comes from the "D-shaped" metal shell that goes around the pins on the male half of the connector. The design is identical to the 25-pin and 9-pin D-shell connectors used for parallel and serial connections on PCs, but bigger. This connector type was very large and cumbersome, never really caught on. However, an alternative 25-pin version of the D-shell was widely used in the Apple hardware world. (Apple "stripped out" the 25 signal return and ground wires that normally would be paired with the true SCSI signals, to save cost). This also never became a standard in the PC world and is not generally seen unless you go looking for it.

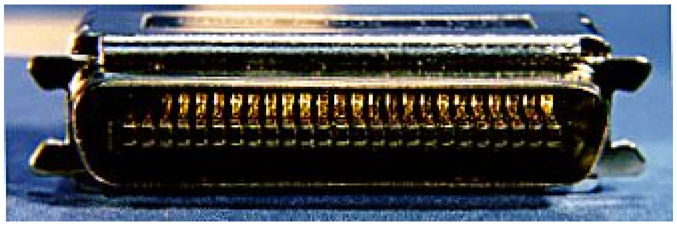


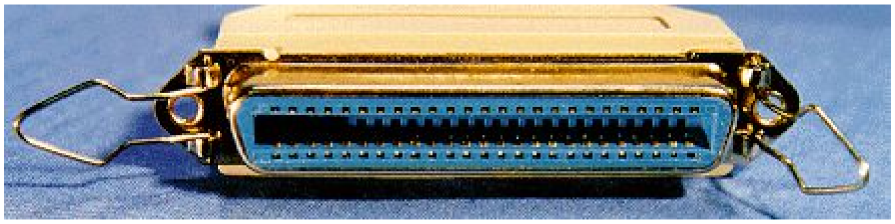
A male DD-50 SCSI connector.
Note the "D-shaped" metal shell around the pins.

Original image © Computer Cable Makers, Inc.
Image used with permission.

Warning: The Apple DB-25 SCSI connector is mechanically identical to a PC's parallel port connector. Be sure not to accidentally connect a DB-25 SCSI device to a PC's parallel port or something might be damaged.

- **Centronics:** The other external connector type defined by the SCSI-1 standard is a 50-pin connector that is commonly called a *Centronics* connector, after a formerly-popular printer that first used this type of connector. In Centronics connectors, instead of thin pins, two rows of flat contacts are used. Two latches on either side are used to hold the connector in place. Centronics connectors are still used for PC printer cables, on the end that attaches to the printer; SCSI Centronics connectors are the same, just with a different number of pins. These 50-pin connectors are still present in the current SCSI specification and are called "Alternative 2" external connectors.

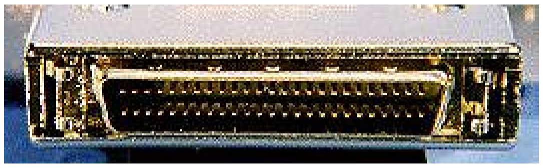


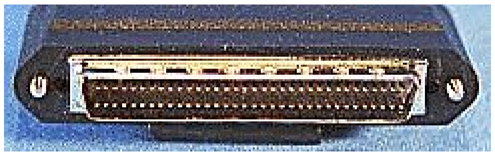


Male (above) and female 50-pin Centronics connectors. As you can see, there are no pins; the contacts are flat. Note the tabs on the sides of the male connector and the latches on the sides of the female connector, which snap into the tabs to secure the connector in place.

Original images © Computer Cable Makers, Inc. Images used with permission.

- **High-Density (HD):** The D-shell connectors defined in the SCSI-1 standard were replaced by newer, *high-density* shielded connectors in SCSI-2. These are really not all that different from the older D-shell connectors, but the space between pins was reduced, making the connectors smaller, cheaper to make and easier to use. The narrow, 50-pin version is called "Alternative 1", and the wide, 68-pin version "Alternative 3". These connectors use a "squeeze to release" latching mechanism instead of Centronics-style latches, and are still used by hardware devices today.

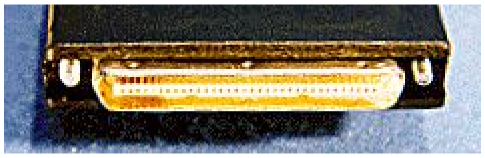




Male 50-pin (above) and 68-pin external high density connectors.

Original images © Computer Cable Makers, Inc. Images used with permission.

- **Very High Density Cable Interconnect (VHDCI):** To further improve the flexibility of SCSI hardware, a new type of external connector was defined as part of the SPI-2 standard. This connector is wide only (68 pins) and is sometimes called a "micro-Centronics" connector, because it uses the same design as the Centronics connectors, only with the contacts *much* smaller and closer together. This is "Alternative 4" for external connectors and is growing in popularity because of its small size. One way that VHDCI is useful; for example, is that two of these connectors can be squeezed side-by-side within the width of a single SCSI host adapter's back edge (expansion slot insert). This doubles the number of external connectors that can be crammed onto a high-end SCSI host adapter.

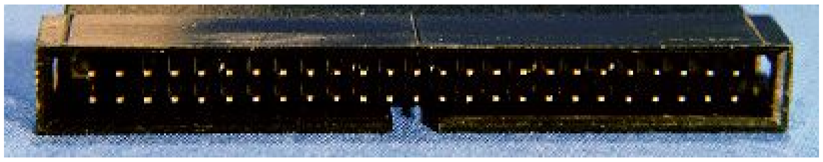


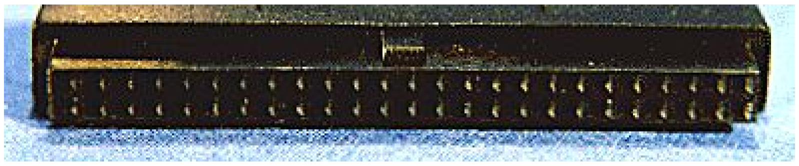
A male 68-pin VHDCI connector.

Original image © Computer Cable Makers, Inc.
Image used with permission.

OK, now let's look at internal (unshielded) connectors:

- **Regular Density:** The SCSI-1 standard defined a single connector type for internal narrow (8-bit) devices. This is a rectangular connector with two rows of 25 pins. This connector type is very similar to that used for IDE/ATA devices, except that there are five extra pins in each row. It is most often seen in older devices and also some newer, slower drives. It is called unshielded "Alternative 2" in the current SCSI standards.

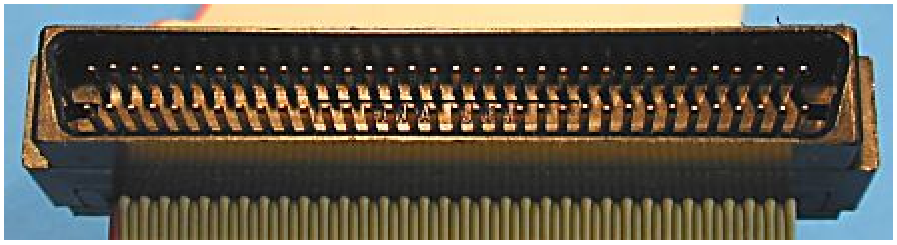




Male (above) and female 50-pin regular density internal connectors. Note the gap in the plastic shield around the male connector, and the tab on the female connector, for keying.

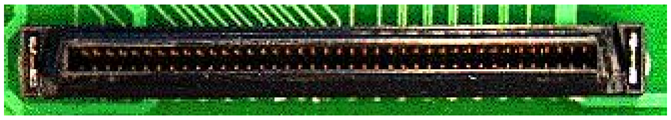
Original images © Computer Cable Makers, Inc.
Images used with permission.

- **High Density:** SCSI-2 defined two new connector types, which are both called *high density* because their pin spacing is half that of the older SCSI-1 connectors, making them much smaller. These are the most common SCSI connectors used today within the PC box. The narrow, 50-pin version is unshielded connector "Alternative 1" and the 68-pin version is "Alternative 3".



A male, internal, high-density 68-pin connector. The 50-pin connector is the same, just narrower. (It is much less common than the 68-pin version.)

- **Single Connector Attachment (SCA):** "Alternative 4" in the SCSI standards for unshielded connectors doesn't actually refer to cable connectors, but the connector used for the *single connector attachment* system for backplane-connection of SCSI drives. See the section on SCA for more details.



A female 80-pin SCA connector. This is the connector that would be found on a backplane designed for SCA SCSI drives.

Original image © Computer Cable Makers, Inc. Image used with permission.

Phew. :^) You may also find the cable and connector summary page helpful in understanding all of these "alternatives".

Next: Narrow (50-Conductor) Single-Ended Cables, Connectors and Signals

Narrow (50-Conductor) Single-Ended Cables, Connectors and Signals

Narrow cables are used for all narrow (8-bit) SCSI transfer modes. These are also sometimes called *50-conductor* or *50-pin* cables after the number of wires in the cable or pins in its connectors, respectively. They are also sometimes called *SCSI-1* cables since the SCSI-1 standard only included narrow SCSI; this is not really a preferred way of naming cables however. Officially, narrow cables in the SCSI standards are called "*A*" cables. There are many different types of narrow cables, depending on the type of cable and connectors used. They are all collectively called "A" cables, though obviously just knowing that a particular cable is an "A" cable isn't enough to tell you if it is one you can use for your application.

Narrow SCSI uses 50 signals, carried on the 50 conductors in the cable. These are organized into 25 *pairs* of two wires each. For single-ended SCSI, each pair generally consists of a signal and a *signal return*, which is the same as a ground line. The signals are the same for the wires in all "A" cables, but the numbering of pins depends on the type of connectors used. Here are the signals and numbering conventions for narrow SCSI:

Signal	**Pin #**		**Cable Conductor #**		**Pin #**		**Signal**
	Set 2	**Set 1**			**Set 1**	**Set 2**	
SIGNAL RETURN	1	1	1	2	2	26	**-DB(0)**
SIGNAL RETURN	2	3	3	4	4	27	**-DB(1)**
SIGNAL RETURN	3	5	5	6	6	28	**-DB(2)**
SIGNAL RETURN	4	7	7	8	8	29	**-DB(3)**
SIGNAL RETURN	5	9	9	10	10	30	**-DB(4)**
SIGNAL RETURN	6	11	11	12	12	31	**-DB(5)**
SIGNAL RETURN	7	13	13	14	14	32	**-DB(6)**
SIGNAL RETURN	8	15	15	16	16	33	**-DB(7)**
SIGNAL RETURN	9	17	17	18	18	34	**-P_CRCA**
GROUND	10	19	19	20	20	35	**GROUND**
GROUND	11	21	21	22	22	36	**GROUND**
(reserved)	12	23	23	24	24	37	**(reserved)**
(no connection)	13	25	25	26	26	38	**TERMPWR**
(reserved)	14	27	27	28	28	39	**(reserved)**
GROUND	15	29	29	30	30	40	**GROUND**
SIGNAL RETURN	16	31	31	32	32	41	**-ATN**
GROUND	17	33	33	34	34	42	**GROUND**
SIGNAL RETURN	18	35	35	36	36	43	**-BSY**

SIGNAL RETURN	19	37	37	38	38	44	**-ACK**
SIGNAL RETURN	20	39	39	40	40	45	**-RST**
SIGNAL RETURN	21	41	41	42	42	46	**-MSG**
SIGNAL RETURN	22	43	43	44	44	47	**-SEL**
SIGNAL RETURN	23	45	45	46	46	48	**-C / D**
SIGNAL RETURN	24	47	47	48	48	49	**-REQ**
SIGNAL RETURN	25	49	49	50	50	50	**-I / O**

Note: "-P_CRCA" was "-DB(P0)" (parity zero) before the introduction of CRC.

"Woah, woah, what are all those numbers, shouldn't it just be one number per signal?" Nothing's simple with SCSI, unfortunately, so let's see if I can explain this table. The table is double-width to prevent its length from getting excessive, and is "mirror-imaged" to make the data easier to understand and to show how the wires are "paired". On the outside are the signal names. Notice that all the "real signals" are on the right-hand side, and the "signal returns" and grounds are on the left. The middle two columns show the conductor numbers assigned to each signal; these are the numbers of the wires in the cable. The numbers between those two columns and the signal names represent two different sets of pin number assignments, which are used for different connector types in the SCSI standards. "Set 1" is the same as the conductor numbering; since the numbers alternate, this means that consecutive numbers are "pairs"; #1 and #2, #3 and #4 and so on, to #49 and #50. "Set 2" instead numbers going down the left column and then does the right column.

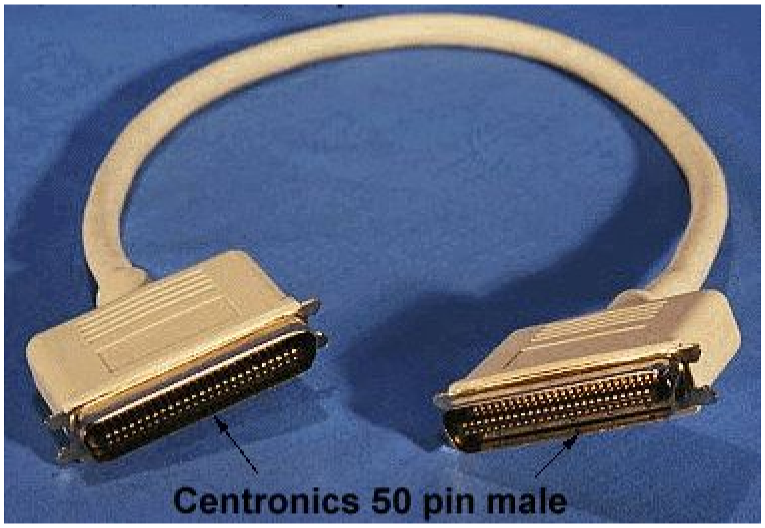
This table shows which connector types use which numbering schemes (see the page on connector types for help in making sense of the terms below):

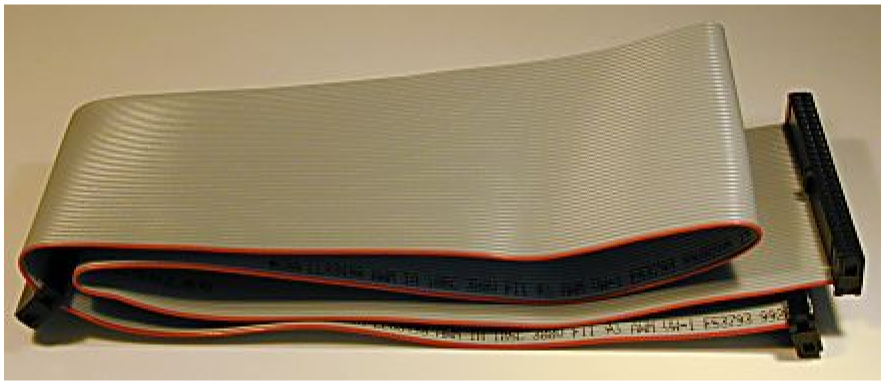
Connector Alternative	**External Cables**		**Internal Cables**	
	Connector Type	**Pin Set Number**	**Connector Type**	**Pin Set Number**
"Alternative 1"	High Density	2	High Density	2
"Alternative 2"	Centronics	1	Regular Density	1

This means that the older cable styles--Centronics for external and regular density for internal--have the pin numbers the same as the conductor numbers. Cables using the newer, high density connectors--for both internal and external cables--use the different numbering specified as "Set 2". I really do not know what the reason was for this change, though I would guess that it made attaching the connectors easier somehow.

OK, now I probably have you more confused than ever. Let's dig past all these formalities and talk about practical "A" cables. These are the most common types of "A" cables that are used in real SCSI systems in the PC for narrow, single-ended implementations:

- **External Centronics "A" Cables:** These are the oldest common type of external SCSI cables, using Centronics-style ("Alternative 2") connectors. They connect older external devices to narrow SCSI buses.
- **External High Density "A" Cables:** These cables are used for newer external devices that have high density ("Alternative 1") connectors instead of Centronics connectors.
- **Internal Regular Density "A "Cables:** Flat, 50-conductor ribbon cables using the older "regular density" ("Alternative 2") connectors. Widely used for connecting to older hard disks and slower devices such as CD-ROMs.
- **Internal High Density "A" Cables:** 50-conductor ribbon cables using the newer high density ("Alternative 1") connectors are used for newer or faster devices (though most of these are now wide devices, and so use wide cabling.)





An external, male Centronics "A" cable (above) and a female regular density internal ribbon "A" cable. These are two of the most common narrow cable types in the SCSI world.

Original images © Computer Cable Makers, Inc. Images used with permission.

To a great extent, the choice of cable depends on the connectors used on the host adapter and the devices being considered. There are some cables that have two different connector types on them, for special applications.There are also a myriad of adapters to allow different connector types to be attached to the same cable.

Next: Wide (68-Conductor) Single-Ended Cables, Connectors and Signals

Wide (68-Conductor) Single-Ended Cables, Connectors and Signals

Wide cables are used for wide (16-bit) SCSI transfer modes. These are also sometimes called *68-conductor* or *68-pin* cables after the number of wires in the cable or pins in its connectors, respectively. Wide cables are formally called "*P*" cables in the SCSI standards, though there are several different types of "P" cables.

Note: Wide SCSI was defined in the SCSI-2 standard, many years after narrow SCSI was already established. The first wide SCSI implementations were designed to use a 68-pin "*B*" cable in combination with the regular narrow "A" cable. This combination was cumbersome and expensive, and very unpopular in the hardware industry. The "P" cable replaced the "A+B" combination and is now the standard for wide SCSI implementations.

Wide SCSI uses 68 signals, carried on the 68 conductors in the cable. These are organized into 34 *pairs* of two wires each. For single-ended SCSI, each pair generally consists of a signal and a *signal return*, which is the same as a ground line. The numbering of the signals is different for the cable conductors and the connector pins, though fortunately there is only one set of pin numbers, unlike narrow SCSI (see the narrow SCSI cable description for more). Here are the signals and numbering conventions for wide SCSI:

Signal	Pin #	Cable Conductor #		Pin #	Signal
SIGNAL RETURN	1	1	2	35	-DB(12)
SIGNAL RETURN	2	3	4	36	-DB(13)
SIGNAL RETURN	3	5	6	37	-DB(14)
SIGNAL RETURN	4	7	8	38	-DB(15)
SIGNAL RETURN	5	9	10	39	-DB(P1)
SIGNAL RETURN	6	11	12	40	-DB(0)
SIGNAL RETURN	7	13	14	41	-DB(1)
SIGNAL RETURN	8	15	16	42	-DB(2)
SIGNAL RETURN	9	17	18	43	-DB(3)
SIGNAL RETURN	10	19	20	44	-DB(4)
SIGNAL RETURN	11	21	22	45	-DB(5)
SIGNAL RETURN	12	23	24	46	-DB(6)
SIGNAL RETURN	13	25	26	47	-DB(7)
SIGNAL RETURN	14	27	28	48	-P_CRCA
GROUND	15	29	30	49	GROUND
GROUND	16	31	32	50	GROUND
TERMPWR	17	33	34	51	TERMPWR
TERMPWR	18	35	36	52	TERMPWR
(reserved)	19	37	38	53	(reserved)
GROUND	20	39	40	54	GROUND
SIGNAL RETURN	21	41	42	55	-ATN
GROUND	22	43	44	56	GROUND
SIGNAL RETURN	23	45	46	57	-BSY
SIGNAL RETURN	24	47	48	58	-ACK
SIGNAL RETURN	25	49	50	59	-RST
SIGNAL RETURN	26	51	52	60	-MSG
SIGNAL RETURN	27	53	54	61	-SEL
SIGNAL RETURN	28	55	56	62	-C/D

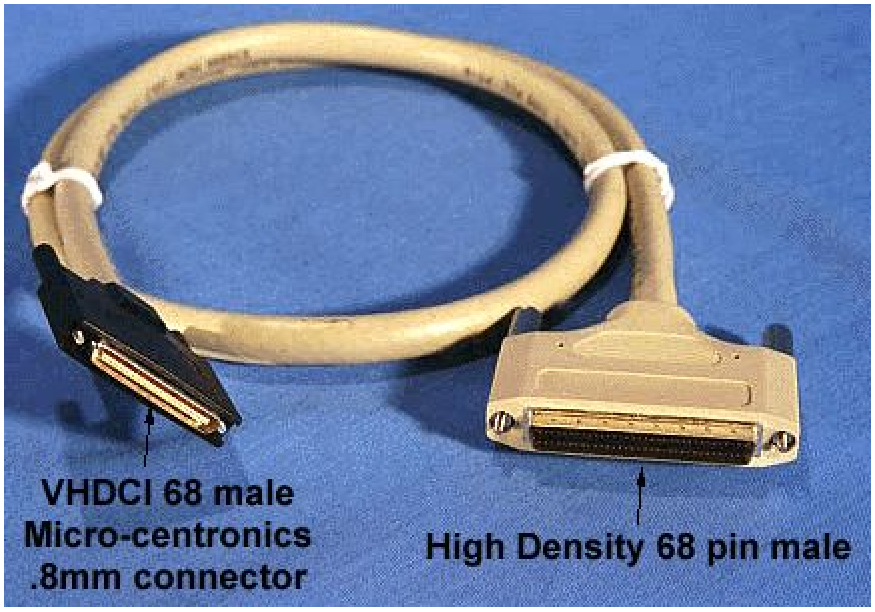
SIGNAL RETURN	29	57	58	63	**-REQ**
SIGNAL RETURN	30	59	60	64	**-I/O**
SIGNAL RETURN	31	61	62	65	**-DB(8)**
SIGNAL RETURN	32	63	64	66	**-DB(9)**
SIGNAL RETURN	33	65	66	67	**-DB(10)**
SIGNAL RETURN	34	67	68	68	**-DB(11)**

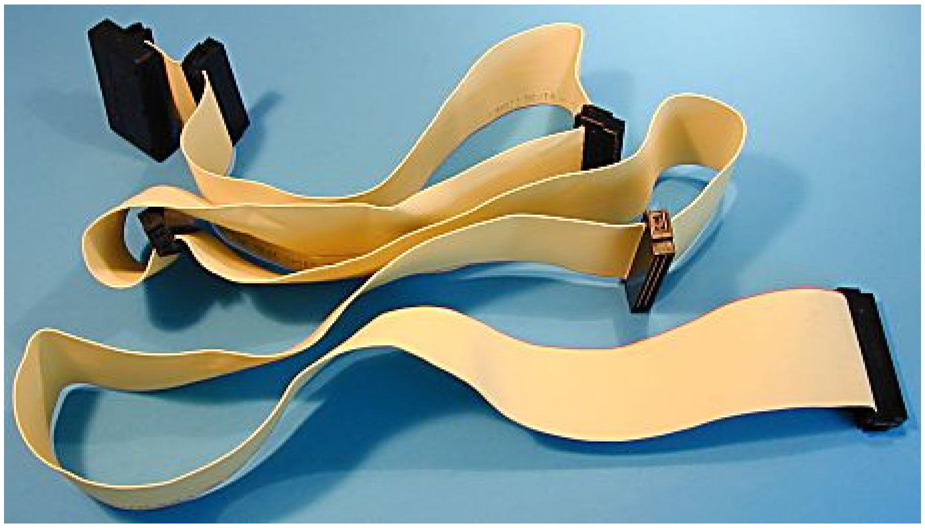
Note: "-P_CRCA" was "-DB(P0)" (parity zero) before the introduction of CRC.

The numbering of the conductors alternates between the left column of signal returns and the right column of signals, facilitating the creation of matched pairs within the cable, and also allowing for the creation of "partially twisted pair" LVD cables. The pins are numbered sequentially down the left column, and then the right column. This probably facilitates manufacturing in some way.

In practical terms, the following are the most common "P" cables used in the PC world for single-ended wide SCSI:

- **External High Density Cables:** The older style of external wide cables uses the larger high density connector.
- **External Very High Density Cables:** The newer style of external wide cables uses the very high density (VHDCI) connector. It is most common with the newest devices (though typically, VHDCI cables use LVD, not single-ended signaling.)
- **Internal High Density Cables:** Only one general type of internal wide cable is used for single-ended signaling, a 68-wire ribbon cable with high density connectors.





Above, an external wide cable that has one high density connector and one very high density connector. Below, an internal wide cable with five high density connectors. Note the integrated terminator on the internal connector (upper left).

Top image © Computer Cable Makers, Inc. Image used with permission.

As with narrow SCSI, the choice of external cable depends on the requirements of the hardware being used. There are cables available that have mixed connector types for special requirements, and also many kinds of adapters. Note that high-end SCSI now uses LVD, so LVD cables are required, which are similar in some ways to single-ended "P" cables, yet different. ;^) They are also, unfortunately, called "P" cables, so watch out for that! Note also that some wide SCSI implementations use single connector attachment (SCA) instead of discrete cables.

Next: Low Voltage Differential (LVD) Cables, Connectors and Signals

Low Voltage Differential (LVD) Cables, Connectors and Signals

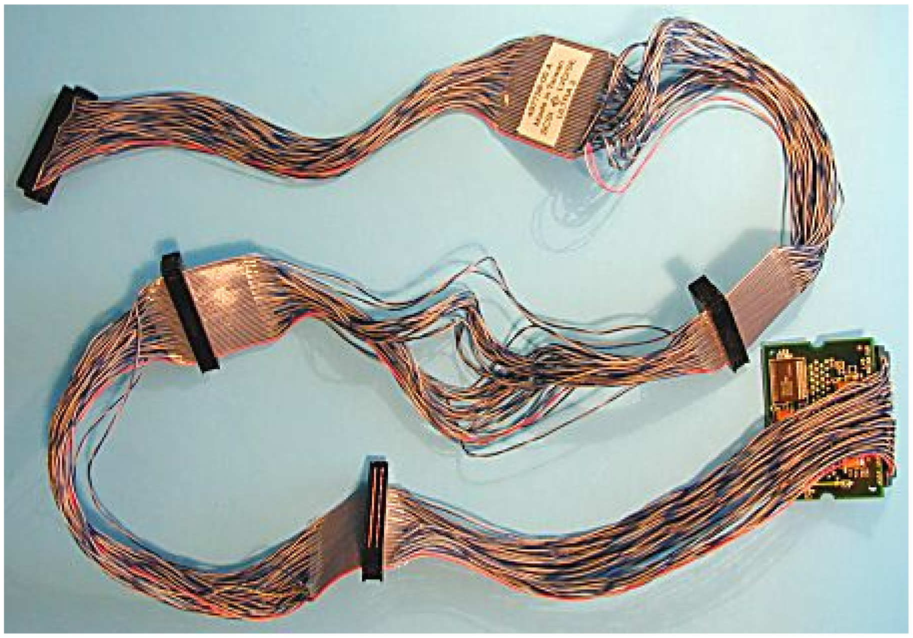
The newest SCSI cables are designed to implement the newest SCSI signaling method, low voltage differential (LVD). Since LVD is a different method of signaling than regular, single-ended (SE) SCSI, the signals used on these cables are quite different than those of SE cables. There are also some differences in terms of the physical cables as well.

First, let's talk about the signals. LVD cables have their signals organized into pairs just like SE cables. However, instead of each pair consisting of a signal and a signal return (ground), each pair consists of the positive and negative complements of the signal. There is also a change to the function of one cable conductor: a special signal called *DIFFSENS*, which is used to control differential signaling.

Other than these small changes, the signals for LVD narrow cables are the same as for SE narrow cables, and the signals for LVD wide cables are the same as for SE wide cables. Rather than repeat two large tables full of signals with nearly-identical versions, I would refer you to the tables for narrow and wide cables respectively. As you look at those tables, remember that for LVD, each "SIGNAL RETURN" is replaced with the "positive" complement of the active-low regular signal. So for example, cable conductor #15 for an LVD "A" cable carries the signal "+DB(7)", and cable conductor #57 for an LVD "P" cable carries the signal "+REQ". For narrow ("A") cables, DIFFSENS replaces one ground signal on cable conductor #21; for wide ("P") cables it is cable conductor #31.

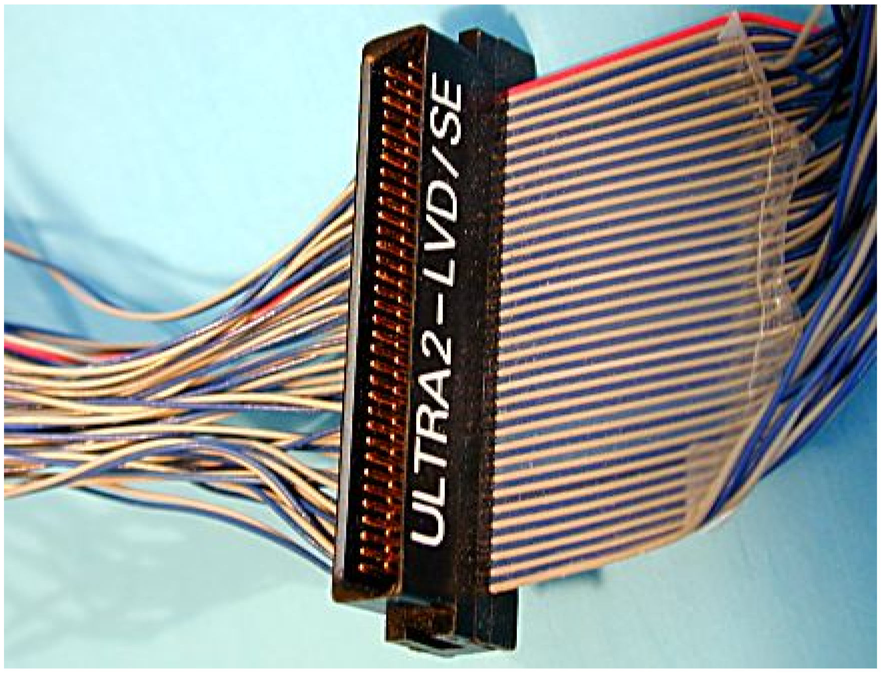
Note: I should point out that I mention narrow LVD cables primarily for "completeness"--they are defined in the SCSI standards but have never been widely used. LVD is generally only implemented for wide devices.

The physical cables used for LVD SCSI chains are also different from single-ended cables, despite the fact that the connectors used are the same for single-ended. This is really a different matter than the changes in signaling; since the wires don't know what signals they are carrying, in *theory* one should be able to use the same cables for LVD as for SE. In practice, however, the greater demands placed by the high speed used in LVD signaling means that problems can occur if you try to use SE cables for LVD implementations.



An internal, 68-wire, 5-connector LVD cable. Note the distinctive "loose" twisted pair wiring between the connectors. The circuit board at bottom right is an integrated LVD/SE terminator. (Incidentally, one of the connectors is hidden behind the terminator and hard to see.)

For external cables, you want to use cabling that is specifically rated for LVD use; these cables have been designed and tested for LVD applications, even if they look the same as their single-ended cousins. As with regular wide cables, they may be found with either high density or very high density connectors. Internal LVD cables are actually very different from SE cables. The reason is that to improve signal integrity, internal LVD cables typically don't use regular flat ribbon cabling. Instead, they use so-called "Twist-N-Flat" cabling, where adjacent pairs of wires are twisted between the connectors, and the wires "flatten out" where the connectors attach. See the discussion of cabling types for more on this kind of wiring.



Detail of the cable pictured above, showing one of the high density connectors, along with a flat section of the cable where the connector attaches, and the twisted pairs of the cable on either side.

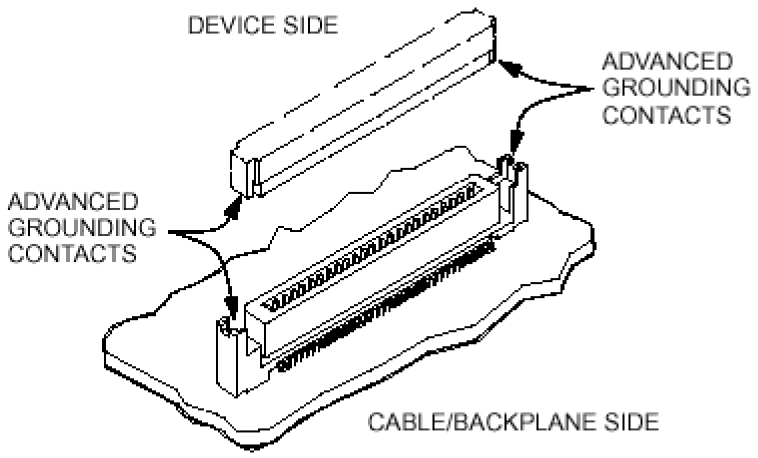
Next: Single Connector Attachment (SCA, SCA-2)

Single Connector Attachment (SCA, SCA-2)

The SCSI standards define four "alternative" connector types for both internal and external SCSI connections. Of those eight, seven involve the use of traditional SCSI cabling of one sort or another. Internal connection "Alternative 4", however, defines a totally different way of connecting SCSI drives to host systems. Instead of the use of discrete cables, the drives are plugged directly into the system. A single connector is used that includes all of the signaling and power inputs and outputs needed by the drive. This technique is called *single connector attachment* or *SCA*.

The reason why this attachment method was developed was to respond to the needs of higher-end systems. Better workstations and servers now employ advanced technology to allow multiple hard disks to be used together to increase performance and improve reliability. This is done through the use of Redundant Arrays of Inexpensive Disks (RAID); these high-end machines may have anywhere from two to dozens of SCSI drives in them. One important feature of RAID is hot swapping, which means that failed hard disks can be removed from the disk array and replaced with new ones without powering down the system. This allows busy machines to keep on running even if a drive fails. Conventional attachment protocols with separate power and data cables--and manual configuration--do not work well in this environment. Regular SCSI hard disk connection methods don't allow for hot swapping. In addition, all the cables involved in connecting power and data to a dozen drives are cumbersome

With the SCA system, the regular 68-pin data connector, 4-pin power connector, and several configuration jumpers on a hard disk are all replaced by a single, unified 80-pin connector. (Incidentally, SCA is a wide-only interface; there is no narrow SCA). This is a Centronics-style connector with special properties used to ensure safe hot plugging of devices into an active system. On the host side, mating connectors are mounted onto a *backplane* consisting of several removable drive bays (you can see a picture of a server case providing removable SCSI SCA drive bays here.)



Simplified schematic showing how SCA SCSI works.

The female connector is on the backplane within the PC, and the male attached to the device (typically a hard disk). The advanced grounding contacts allow hot swapping without creating electrical problems (see further down the page for more).

Image © Information Technology Industry Council (ITI) (See here for more on ITI.) *Image used with permission.*

The first SCA implementation had some issues, and the technology was revised in a new version now called *SCA-2*. Several attributes about the interface were changed with this new version of SCA, and SCA-2 is now the standard in the industry. In order to frame the discussion of the features of SCA, let's take a look at the single-ended signals and contact numbering scheme for the current standard 80-contact SCA connector:

Signal	Long Host Contact?	Connector Contact #		Long Host Contact?	Signal
12V CHARGE	YES	1	41	YES	**12V GROUND**
12V		2	42	YES	**12V GROUND**
12V		3	43	YES	**12V GROUND**
12V		4	44		**MATED 1**
3.3V		5	45	YES	**3.3V CHARGE**
3.3V		6	46	YES	**GROUND**
-DB(11)		7	47		**SIGNAL RETURN**
-DB(10)		8	48		**SIGNAL RETURN**
-DB(9)		9	49		**SIGNAL RETURN**
-DB(8)		10	50		**SIGNAL RETURN**
-I/O		11	51		**SIGNAL RETURN**
-REQ		12	52		**SIGNAL RETURN**
-C/D		13	53		**SIGNAL RETURN**
-SEL		14	54		**SIGNAL RETURN**

-MSG		15	55		**SIGNAL RETURN**
-RST		16	56		**SIGNAL RETURN**
-ACK		17	57		**SIGNAL RETURN**
-BSY		18	58		**SIGNAL RETURN**
-ATN		19	59		**SIGNAL RETURN**
-P_CRCA		20	60		**SIGNAL RETURN**
-DB(7)		21	61		**SIGNAL RETURN**
-DB(6)		22	62		**SIGNAL RETURN**
-DB(5)		23	63		**SIGNAL RETURN**
-DB(4)		24	64		**SIGNAL RETURN**
-DB(3)		25	65		**SIGNAL RETURN**
-DB(2)		26	66		**SIGNAL RETURN**
-DB(1)		27	67		**SIGNAL RETURN**
-DB(0)		28	68		**SIGNAL RETURN**
-DB(P1)		29	69		**SIGNAL RETURN**
-DB(15)		30	70		**SIGNAL RETURN**
-DB(14)		31	71		**SIGNAL RETURN**
-DB(13)		32	72		**SIGNAL RETURN**
-DB(12)		33	73		**SIGNAL RETURN**

5V		34	74		**MATED 2**
5V		35	75	YES	**5V GROUND**
5V CHARGE	YES	36	76	YES	**5V GROUND**
SPINDLE SYNC	YES	37	77	YES	**ACTIVE LED OUT**
RMT_START	YES	38	78	YES	**DLYD_START**
SCSI ID (0)	YES	39	79	YES	**SCSI ID (1)**
SCSI ID (2)	YES	40	80	YES	**SCSI ID (3)**

Looking at this table, you will notice several differences compared to the signal chart for regular wide SCSI attachment. These really define the special characteristics of the SCA interface:

- **Regular Signals:** The "core" SCSI signals are the same as for wide SCSI; they are found in the middle part of the connector. The only difference is that the signals are in the left column and the returns on the right, instead of the other way around.
- **Power Signals:** Three voltages of power are supplied to the hard disks: 3.3 V, 5 V and 12 V. Again, these are provided so that the drive can operate without needing a separate Molex-style four-pin power connector.
- **Long Contacts:** The "Long Host Contact?" column has the word "YES" in it for several contacts. The connector on the host (PC) side is specially designed so that these contacts are made a bit longer than the regular length associated with most of the contacts in the connector. This is an important feature: what it does is to ensure that these signals make contact before any others when a drive is inserted, and also that they *break* contact last when a drive is removed. Without this feature, subtle changes in the angle of insertion or pressure applied to the drive as it is pushed into the system could cause spurious results--a voltage contact might connect before the ground contacts, for example. (Remember that compared to the speed of electricity, human hands are extremely slow; even the split-second that passes while a connector is being seated represents an eternity to electronic circuits.)
- **Power Charge Signals:** Each of the three voltages mentioned above has associated with it a "precharge" signal. These are on long contacts while the regular voltage signals are on short contacts. When a drive is inserted, these precharge circuits make contact a bit before the regular voltage circuits do. A drive can be designed to use these charge signals to "precharge" its internal circuits during hot plugging, to soften the sudden "surge" of current when the regular power signals connect.
- **SCSI ID Signals:** Instead of using jumpers, SCA drives have their device IDs set through software control. This is obviously important to enable the use of many drives and hot swapping.

- **Other Control Signals:** Extra signals are provided to allow control over other drive features such as remote or delayed starting of the drive motor. A signal is also provided for the drive to indicate that it is active, so the system can light the appropriate drive activity LED.

The physical SCA connectors also incorporate physical features to make hot plugging of drives easier. SCA drives slide into drive bays and mate with the system connector in the back without the user being able to see what is going on; this is called *blind connector mating*. To ensure that the connectors line up properly, special guides are provided on the host connector.

There is also an LVD version of the SCA interface, which is now taking over from the single-ended type of SCA, as all new hard disks use LVD. The difference between LVD SCA and SE SCA is the same as the difference between LVD wide cable signals and SE wide cable signals: there are two changes. The first is that all of the signal returns are replaced with positive complement signals; for example, contact #58 on an LVD SCA connector is "+BSY", and so on. The second is that one ground is replaced by the DIFFSENS signal, in this case contact #46.

Warning: Though physically identical, LVD drives must use the LVD version of the SCA interface to function properly.

Next: SCSI Adapters

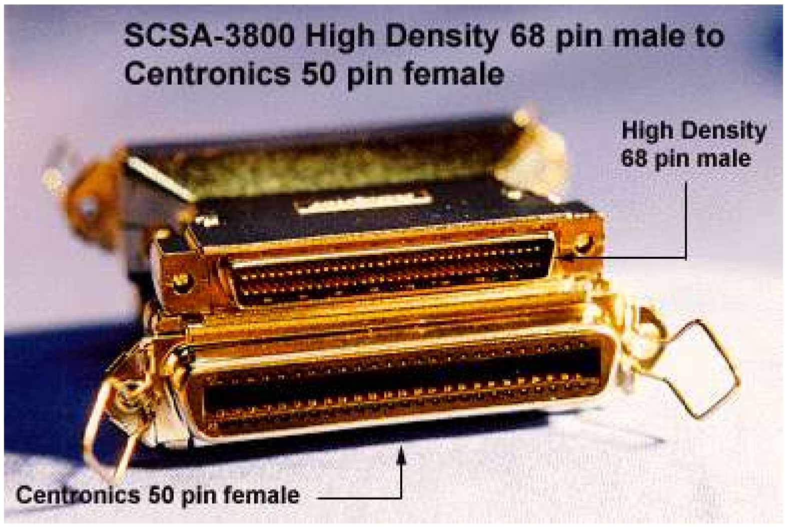
SCSI Adapters

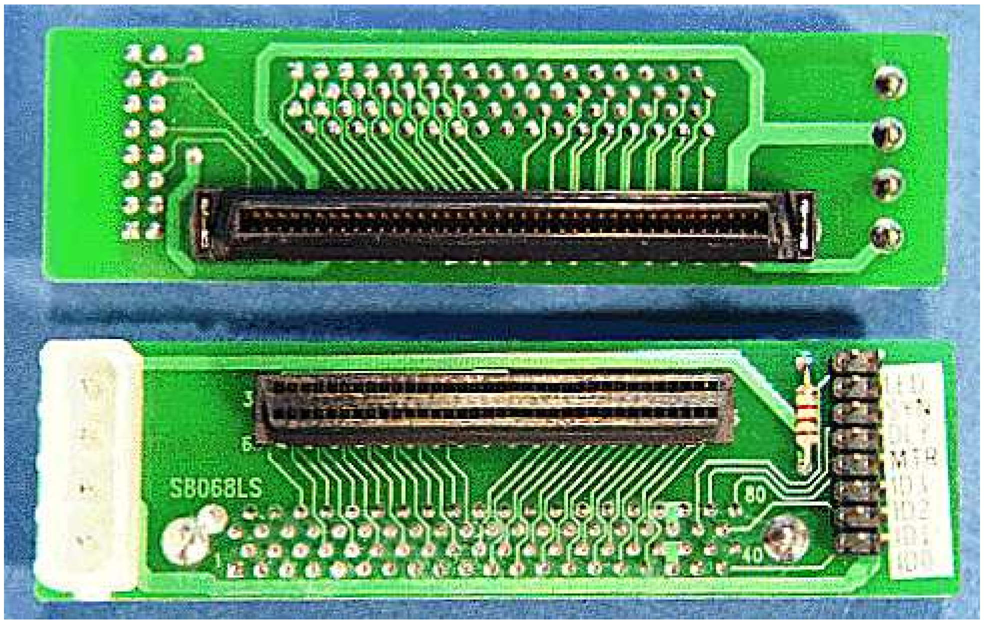
With so many different types of SCSI protocols, cables and connection methods, it's no surprise that there also exists on the market an enormous number of different *SCSI adapters*. These devices are generally manufactured and sold by the same people who make and sell SCSI cables, and are intended to solve some of the problems that crop up when SCSI users try to interconnect different kinds of SCSI hardware.

There are probably over a hundred different types of SCSI adapters available; some of the most popular ones fall into these general categories:

- **Mechanical Connector Adapters:** The most common types of adapters are used to allow devices with different types of connectors to be used on the same cable. These are simple, purely mechanical adapters that don't contain any logic or live circuitry, and are relatively inexpensive. For example, you might have an existing external SCSI cable using high density connectors, and want to add to the SCSI chain a device that has a Centronics connector. There are dozens of different connector adapters, reflecting the myriad of combinations of connector types.
- **SCA Adapters:** These are adapters that let you use SCA drives on regular systems that don't have SCA backplanes.

- **Wide/Narrow Adapters:** Adapters that let you put a narrow SCSI device on a wide cable or vice-versa. There are complications involved in doing this; see here for more.
- **Internal/External Adapters:** Adapters that let you use an internal cable outside the PC.
- **Signaling Method Adapters:** Adapters that let you use differential drives on a single-ended SCSI chain or vice-versa. These are electrically active adapters and are generally expensive.
- **Interface Adapters:** Adapters that let you use SCSI devices on other interfaces; the most common is an adapter to let you use certain types of SCSI drives on a PC's parallel port. Again, these can be a bit expensive. Also, you will be limited to (at best) the maximum speed of whichever interface is slowest.


SCSA-3800 High Density 68 pin male to Centronics 50 pin female
High Density 68 pin male
Centronics 50 pin female



Two adapters that are representative of two categories of adapter types. Above, a simple mechanical adapter, which converts from a high density 68-pin male connector to a Centronics-style 50-pin female. (There are actually two of them, stacked to show both connectors.) Below, an SCA to high-density 68-pin adapter.

Original images © Computer Cable Makers, Inc. Images used with permission.

It's important to remember that there can be reliability issues with using adapters. It might be possible to mate two devices to each other with the use of a mechanical adapter, but that doesn't mean that the interface will necessarily function reliably with that configuration. To some extent it depends on the nature of the SCSI bus being implemented, and the quality of the hardware. In some cases adapters work just fine with no problems, but in others getting everything to work together can be a bit tricky. It's best to consult with a qualified hardware vendor if you are unsure of how to make different devices work together.

Another aspect to keep in mind is that the cost of some SCSI adapters can be very high. It might be *possible* to adapt one type of device to use it with a very different type of host adapter, but in many cases it will not be cost-effective. Some adapters are so expensive that it would be cheaper to get a new cable, host adapter or other "incompatible" hardware rather than buy the adapter. This is particularly true of adapters that change between signaling methods.

Next: SCSI Bus Termination

SCSI Bus Termination

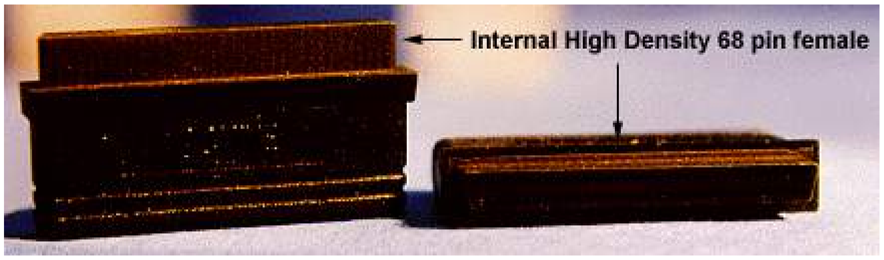
You can do an experiment (either physically or mentally) to illustrate why termination is required on a SCSI bus. Hold one end of a piece of rope about six feet long and have someone else hold the other end. Stretch the string so it is reasonably taut, but not tight, and then snap down on one end sharply. You will form a wave that travels down the string. When it reaches the end of the string it will "reflect" off the end and travel back again toward you, and then reflect again. It will go back and forth across the string, decreasing in amplitude each time until it eventually dies out.

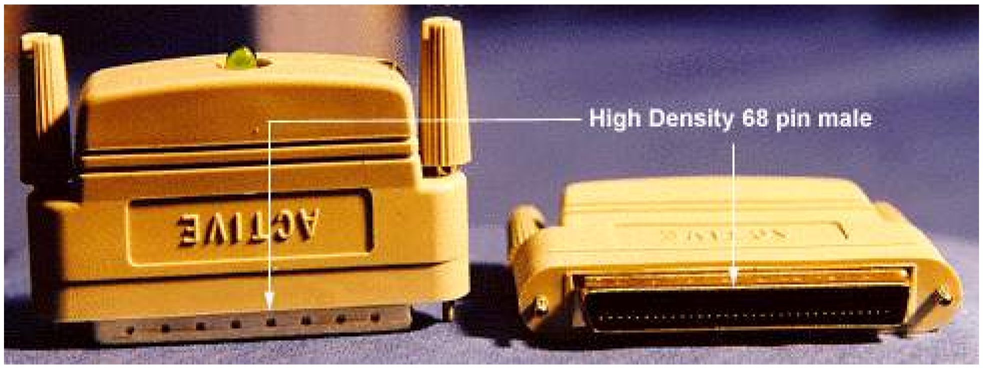
Electrical signals travel across wires in much the same way as physical waves travel across a string. When they reach the end of the wire, they will reflect and travel back across the wire. The problem is that if this is allowed to happen, the reflected signals will interfere with the "real" data on the bus and cause signal loss and data corruption. To ensure that this does not happen, each end of the SCSI bus is *terminated*. Special components are used that make the bus appear *electrically* as if it is infinite in length. Any signals sent along the bus appear to go to all devices and then disappear, with no reflections.

There are several different kinds of termination used on SCSI buses. They differ in the electrical circuitry that is used to terminate the bus. Better forms of termination make for more reliable SCSI chains; the better the termination, the fewer problems (all else being equal) with the bus, though cost is generally higher as well. In general terms, slower buses are less particular about the kind of termination used, while faster ones have more demanding requirements. In addition, buses using differential signaling (either HVD or LVD) require special termination.

Here are the different types of SCSI termination:

- **Passive Termination:** This is the oldest, simplest and least reliable type of termination. It uses simple resistors to terminate the bus, similar to the way terminators are used on coaxial Ethernet networks. Passive termination is fine for short, low-speed single-ended SCSI-1 buses but is not suitable for any modern SCSI speeds; it is rarely used today.
- **Active Termination:** Adding voltage regulators to the resistors used in passive termination allows for more reliable and consistent termination of the bus. Active termination is the minimum required for any of the faster-speed single-ended SCSI buses.
- **Forced Perfect Termination (FPT):** This is a more advanced form of active termination, where diode clamps are added to the circuitry to force the termination to the correct voltage. This virtually eliminates any signal reflections or other problems and provides for the best form of termination of a single-ended SCSI bus.
- **High Voltage Differential (HVD):** Buses using high voltage differential signaling require the use of special HVD terminators.
- **Low Voltage Differential (LVD):** Newer buses using low voltage differential signaling also require their own special type of terminators. In addition, there are special LVD/SE terminators designed for use with multimode LVD devices that can function in either LVD or SE modes; when the bus is running single-ended these behave like active terminators.





Internal (above) and external active terminators. The LED on the external shows that the terminator is connected. Some multimode LVD/SE terminators have LEDs that light up one color when the bus is running in LVD mode, and a different color when it is running in SE mode. This is useful for troubleshooting.

Original images © Computer Cable Makers, Inc. Images used with permission.

Terminators must be at the very ends of the bus, after all of the actual devices on the chain. This includes any devices that may be powered off or temporarily disconnected. Therefore, there are always exactly two terminators per bus or bus segment. Many devices contain internal terminators that can be used if the device is at one of the ends of the SCSI bus. However, differential drives typically do not include the ability to terminate the bus, so newer LVD applications require explicit terminator hardware. Sometimes terminators are built in to the end of the SCSI cable. In addition, systems using the Single Connector Attachment system have a different termination arrangement because the connection system is different. SCA drives do not have termination on them.

Note: Host adapters usually do include the ability to terminate the SCSI bus. In fact, many host adapters include multiple segments, and so have the ability to terminate each segment they support. Termination should only be enabled on a host adapter if the host adapter is the last device on any segment. If you are using both internal and external devices on a host adapter that has only one logical segment being shared by both internal and external drives, the host adapter is going to typically be in the middle of the chain between them, and its internal termination should be disabled.

Termination is a rather straight-forward affair when all of the devices on the SCSI bus are the same width: either narrow (regular, 8 bit) or wide (16 bit) SCSI. When you mix narrow and wide SCSI on the same bus, you must be more careful about termination. The issue that arises is that if part of the device is running in wide mode, but not all devices are wide, half of the data lines (the "high byte") may end somewhere on the bus; they need to be terminated, and that termination may occur in a different place than where the "low byte" data signals are terminated.

Normally these issues are handled using special adapters or cables that only extend the extra width to the devices that are using the wide portion of the bus. However, the extra signals on the wide part of the bus must *also* be terminated properly. Problems can result with wide devices when these extra signals are not terminated and are left "dangling". See this discussion of mixing wide and narrow devices for more details.

Next: Summary of SCSI Cables and Connectors

Summary of SCSI Cables and Connectors

The table below provides a quick reference summary of the different connector types used for both internal and external SCSI cables:

Cable Type	Connector "Alternative"	Connector Type	Contacts	Cable Name
External	**1**	High Density	50	External High Density "A" Cable
	2	Centronics	50	External Centronics "A" Cable
	3	High Density	68	External High Density "P" Cable
	4	VHDCI	68	External Very High Density "P" Cable
Internal	**1**	High Density	50	Internal High Density "A" Cable
	2	Regular Density	50	Internal Regular Density "A" Cable
	3	High Density	68	Internal High Density "P" Cable
	4	SCA	80	(n/a)

Note: I only included current connector types; for a description of the older D-shell connectors that were formerly internal connector "alternative 1", see the page on connector types.

Next: SCSI Configuration

SCSI Configuration

I've already devoted entire sections of this SCSI discussion to several important issues related to configuring and setting up SCSI systems. These include discussions of the various SCSI transfer modes, SCSI host adapters, and important cable and connector issues. You'll want to be sure you understand those key aspects of the technology before you try to equip a system with SCSI hardware.

However, there are several more considerations that must be taken into account when setting up a SCSI bus. In this section I will cover important issues that any person setting up or managing a SCSI PC will need to understand. I begin with a discussion of SCSI bus topology, which covers legal and illegal ways of connecting devices. I then talk about the number of devices allowed on a SCSI chain and how to set device IDs to prevent conflicts. I then cover several miscellaneous configuration issues, including software control, BIOS issues, and physical installation considerations. Finally, I talk about how to mix wide and narrow devices, and also how to use SCSI drives along with IDE/ATA drives in the same system.

Next: SCSI Bus Topology

SCSI Bus Topology

Topology is a term that refers to the shape or structure of things. In the computer world, it usually refers to the way that devices are connected or arranged. The SCSI interface uses a *bus topology*. This means that all devices are daisy-chained linearly in a long line. This is identical to how a thin Ethernet (10base2) network is set up (the cables and signals aren't the same of course, I am referring to how the devices are logically linked together in a line). This is sometimes called a *SCSI chain* as well.

When setting up a SCSI bus, it is imperative that each device be in a straight line. This means that each device (including the host adapter) is connected to either one or two other devices, and never more than that. The two devices at the end of the bus must be terminated, either internally or externally. The bus should never be connected in a loop, star or other formation.

For a two-device bus, the topology typically looks like this:

Terminator -- Device A -- Device B -- Terminator

For a four-device bus, it is as follows:

Terminator -- Device A -- Device B -- Device C -- Device D -- Terminator

It does not matter which device is located where on the chain, and any of the devices can be either internal or external, but the terminators must be at the ends. The terminators can be either explicit devices or part of the final devices on the cable. So in the second example, if Device D had internal termination, it could be enabled instead of using a separate terminator on that end of the cable. Obviously, having the internal devices at one end of the bus, and the external devices at the other end, with the host adapter in the middle, is going to be the setup that makes the most sense for those that are using both internal and external drives. If the host adapter is placed in the middle of the chain, it must have its termination disabled, otherwise the devices past it on the chain will not function. (The exception is if the host adapter implements multiple channels or segments. If so, each channel or segment must be separately terminated; see here more.)

If using a cable with more connectors than you have devices, it is acceptable to leave some of the connectors unused, but they should be left in the middle of the cable, with the terminators still at the end of the cable. Terminating a SCSI chain while leaving connectors "dangling" is not a good idea, as this can create signaling problems. So the following is legal:

Terminator -- Device A -- (unconnected) -- Device B -- Device C -- (unconnected) -- Terminator

But the following is *not* legal:

Terminator -- Device A -- Device B -- Device C -- Terminator -- (unconnected) -- (unconnected)

Some SCSI chains, particularly those used for faster transfer modes or LVD signaling, work better if the last device on any SCSI cable is connected to the last connector on the cable, the one closest to the terminator. Also, evenly spacing devices out over the bus is preferable from a reliability standpoint.Thus, the following would probably be a better configuration than prior example, even though that one is technically legal:

Terminator -- Device A -- (unconnected) -- Device B -- (unconnected) -- Device C -- Terminator

Next: Number of Devices

Number of Devices

One of SCSI's strengths as an interface is its support for large numbers of devices on the bus, and the fact that different "conversations" between devices can be taking place simultaneously. SCSI buses can support as many as 16 different devices; compared to the IDE/ATA interface and its limit of two, this offers significant flexibility.

There are four different issues that influence the number of devices that can be used on a single SCSI channel:

- **Bus Width:** SCSI was originally defined as a narrow, 8-bit bus, with support for 8 different devices. When wide 16-bit SCSI was created, this was expanded to support for 16 devices. Since the host adapter is itself a SCSI device, this means a theoretical maximum of 7 other devices for narrow SCSI, or 15 for wide.

- **Transfer Mode and Signaling Method:** Certain transfer modes and signaling methods limit the theoretical maximum number of devices, due to electrical signal integrity considerations. This is particularly true of Wide Ultra SCSI, where single-ended implementations are limited to 8 devices despite being 16 bits wide, because single-ended signaling can't handle 16 devices on a cable at that speed.
- **Cable Length:** Many SCSI transfer modes involve a "tradeoff" between cable length and the number of devices that can be supported. For Ultra SCSI in particular, reducing the number of devices on the chain allows the use of longer cables and vice-versa. Newer implementations that use LVD signaling are normally limited to 12m in length, but this can be extended to 25m if only two devices are used on the cable.
- **Practical Considerations:** The limitations above are all based on theoretical considerations. There are also practical issues involved in running many devices on a SCSI bus. For example, you may be able to run 16 hard disks on your SCSI bus *in theory*, but to do this would require a very large system case and a very power power supply. For external devices there are similar issues to be addressed. Long cables with many connectors are also very expensive and may require devices to be placed close to each other so the cable will reach all the devices. Having many devices can make high reliability more difficult. And so on...

This table shows a summary of all the different transfer modes and signaling methods, and the number of devices supported for each.

Note: It's pretty unusual in most systems to need more than 8 devices, much less 16. If this is an issue, however, you can connect more devices either by using two host adapters in the PC, or a single host adapter that supports multiple independent channels.

Next: SCSI Device IDs

SCSI Device IDs

Each SCSI device is addressed on the bus via a specific number. For narrow SCSI (which allows up to 8 total devices), these are numbered 0 through 7; for wide SCSI (16 devices) the numbering is 0 through 15. The priority that a device has on the SCSI bus is based on its ID number. For the first 8 IDs, higher numbers have higher priority, so 7 is the highest and 0 the lowest. For Wide SCSI, the additional IDs from 8 to 15 again have the highest number as the highest priority, but the entire sequence is lower priority than the numbers from 0 to 7. So the overall priority sequence for wide SCSI is 7, 6, 5, 4, 3, 2, 1 , 0, 15, 14, 13, 12, 11, 10, 9, 8.

The priority levels are used to guide the *arbitration* process. I describe arbitration in some detail here, but in a nutshell, it is the process by which different devices decide which one can have control of the bus. If more than one device wants control at the same time, the higher-priority device will "win", while the lower-priority device will have to wait for its turn. Since devices are addressed specifically by other devices, the ID setting on the bus doesn't matter a great deal on low-traffic buses--all that is essential is that no two devices be set to the same ID, or obviously great confusion will ensue. In high-traffic settings, you will typically want to set the *slower* devices (scanners, tape drives) to the higher-priority IDs, to ensure that they are not crowded off the bus by the faster devices like hard disks.

Another consideration is that if you have any devices that absolutely cannot tolerate delays in receiving their stream of data--such as a CD recording drive or a video encoder--they should be given top priority on the bus. Many people also like to make the host adapter the highest-priority device on the bus, which is why host adapters will often default to a SCSI ID of 7. It should be noted that some older host adapters can be finicky about device IDs. Some will only boot a hard disk if it is set to device ID 0. (This is inflexible and has been basically done away with in newer hardware.)

Warning: Some host adapters support multiple bus segments on the same SCSI bus. Device IDs must be unique across all segments that are on the same bus. You can't put two devices both with ID #7 for example on two different segments of the same SCSI chain.

The method of actually configuring the ID depends on the specific device. Many devices use hardware jumpers or switches, or even a rotary dial to set the device ID, on the back of the device enclosure. More sophisticated devices use software utilities--this is most common with more modern SCSI host adapters. If the drive uses jumpers, be sure to check the configuration settings carefully; different drives use different jumper combinations to set particular numbers.

Finally, the use of Plug and Play SCSI allows for automatic assignment of device IDs on the bus, to eliminate devices trying to use the same IDs simultaneously, for systems that support the feature (and when it works). Single Connector Attachment drives also have their IDs set by the host system, to allow for automatic configuration and hot swapping.

Next: BIOS Issues

BIOS Issues

SCSI hard disks do not suffer from most of the BIOS "barrier" issues that plague IDE/ATA hard disks, because they do not have to deal with the severe restrictions that result from the incompatible IDE and BIOS geometry specification limits. SCSI by its nature uses logical block addressing (LBA) to address devices, and does not rely on the system BIOS for geometry information as IDE/ATA does.

Since SCSI hard disks are controlled by the internal SCSI bios on the host adapter, this gets around the issues inherent with the motherboard's BIOS. However, it means that any limits of the host adapter BIOS can affect access to devices it controls. Some older host adapters did have problems that made them unable to access hard disks over 1 GB in size, or some other arbitrary limits. Newer host adapters should not have these limitations. Of course, if you are using the FAT file system, you are still restricted to 2 GB per partition, since this is a file system issue, not a hardware or BIOS one. See here for a full discussion.of this barrier.

Next: SCSI Software Interface Protocols (ASPI and CAM)

SCSI Software Interface Protocols (ASPI and CAM)

In order for a device to be used on the SCSI bus, it is necessary for it to have a driver that interfaces it to the bus. (The exception is hard disks, which are normally usable directly by the host adapter, since it is designed for that purpose). The problem with this is making sure the drivers are available for all the devices you want to use and that they work properly. A further problem is making sure that operating systems and applications can work with all of the drivers that are written for various devices.

To help make the situation with drivers more standardized, Adaptec--the biggest name in SCSI host adapters--developed the *Advanced* (originally *Adaptec*) *SCSI Programming Interface*, or *ASPI*. This driver acts as an abstraction layer that hides the details of the host adapter from the operating system or application and makes device support more universal.

Most of the popular operating systems can make use of ASPI. DOS requires an ASPI driver to be loaded, while the protected mode operating systems such as Windows 95 and Windows NT have native support for ASPI. One problem with using SCSI under DOS or Windows 3.x is that the lack of protected-mode drivers means yet another real-mode drive that uses up precious conventional memory.

An alternative interface protocol is called *Common Access Method* or *CAM*. This is a more sophisticated and complex software interface protocol, which is defined as one of the SCSI-3 standards. CAM and ASPI both do basically the same thing; which is used depends on the particular system.

Next: Plug and Play SCSI (SCAM)

Plug and Play SCSI (SCAM)

Each SCSI device is required to have a unique device ID in order to be addressed over the shared SCSI bus. This has usually involved setting jumpers or switches on devices like hard disks, and either physical or software setup for host adapters. To eliminate this need for manual configuration, a specification was developed to automate the assignment of device IDs. This protocol allows the SCSI bus to dynamically shift and reallocate IDs when a new device is added to the bus, and is called *Plug and Play SCSI* or alternatively, *SCSI Configured AutoMatically* (*SCAM*). (Yuck, what a contrived acronym.) Plug and Play SCSI also supports automatic termination of the SCSI bus.

Note: This idea is similar in concept to "regular" Plug and Play in the overall PC system, but remember that Plug and Play SCSI is used to assign SCSI device IDs, not system resources. In fact, many modern SCSI host adapters also support system plug and play over the PCI bus to allow dynamic setting of system resources. This is a different matter altogether, though these may be sold as "Plug and Play SCSI controllers"...

While a great idea in theory, SCAM never lived up to its potential. The main issue with the feature is that it doesn't always work reliably--it's not a simple matter to automatically assign device IDs or figure out which devices should enable termination. Many people would find that they needed to disable the feature and manually configure drives to eliminate problems. There were also difficulties if devices supporting SCAM were mixed with other devices that did not support the feature. Eventually, hardware makers started recommending that people avoid the feature altogether and just configure devices manually, reasoning that this would reduce the likelihood of problems. And let's be honest--it doesn't take that long to set the IDs of a few devices, and you typically only have to do it once anyway.

SCAM was eventually removed from the SCSI parallel interface standard in the SPI-3 revision.

Next: Physical Installation Issues

Physical Installation Issues

SCSI hard disks are not inherently any different in terms of physical construction than their cousins that use other interfaces such as IDE/ATA. However, there are often more issues related to proper installation of SCSI drives, not due to the interface per se, but due to the fact that SCSI drives tend to be faster and run hotter than IDE/ATA drives do. The fastest SCSI devices may require active cooling; even if it isn't needed, attention must definitely be paid to where the drives are installed in the system case, and the cooling capacity of the case as a whole. See here for more on hard disk cooling and installation issues.

The issues with installing a SCSI system will vary from one system to another. Simple SCSI installs with only a few devices are actually fairly straight-forward, so don't let this page scare you off. :^) However, more involved configurations require more planning. For example, a special ("server") case will generally be needed for installations with many drives; special enclosures are often used for RAID arrays. Providing sufficient power to a number of SCSI drives means that the power supply of the system must often be "beefed up" as well.

Cabling concerns can also affect the physical installation of SCSI systems. Since it is best to use shorter cables if at all possible--both for improved signal quality and to reduce cost--it may be preferable to place SCSI drives relatively close together, as long as they are not so close to each other that cooling or ventilation suffers. This would also apply to external devices. Fortunately, it does not matter which devices are connected to which cable connectors.

Next: Mixing Narrow and Wide Devices

Mixing Narrow and Wide Devices

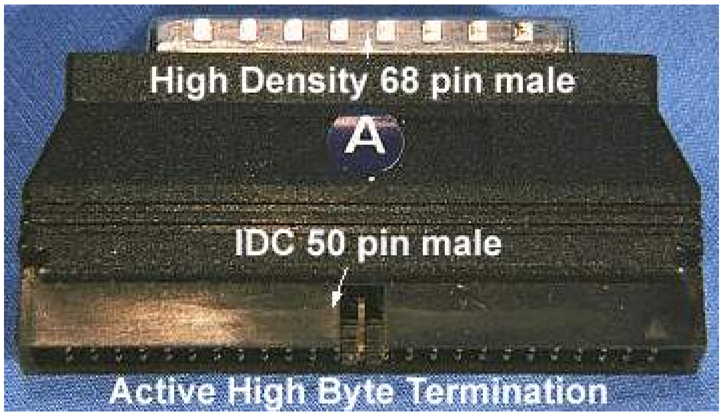
The simplest SCSI configurations use only narrow devices, or only wide devices. When this is done, the correct cable is chosen, all the devices are connected in a bus topology, and the appropriate type of termination is placed at both ends of the bus. Termination is relatively straight-forward because all devices have the same width. However, there are often situations where narrow and wide devices may need to be mixed on a single SCSI channel. This has become even more of an issue in recent years, because the newest hard disks are wide only, but many other kinds of devices are still produced for the narrow interface.

The best way of mixing wide and narrow devices is to use a host adapter that has built in support for separate segments or channels for wide and narrow devices. If you use this sort of host adapter, you can effectively set up the wide and narrow devices separately, using a narrow cable and terminator for the narrow segment (or channel), and wide hardware for the wide segment (or channel). The host adapter takes care of the issues involved in mixing the devices (though you may need to help through the setting of configuration parameters). Another reason why this sort of adapter is a good idea is that narrow devices generally use single-ended signaling, while modern wide devices require LVD for maximum performance. The two cannot be mixed on the same bus segment without the hard disks dropping down to single-ended mode.

If you do not have a host adapter with separate support for narrow and wide devices, you will have to mix them on the same SCSI chain, which introduces several complications. Here are some of the issues you will need to take into account when doing this:

- **Adapters:** You will have to use either a wide or a narrow cable, depending on whether the host adapter you are using is wide or narrow. Then, whichever drives are the opposite width will need an adapter so they fit onto the cable. For example, if you have a wide cable connected to a wide host adapter and several wide devices, to add a narrow device to this chain you will need an adapter for the narrow device to let it plug into the wide cable.
- **Performance:** If you put a wide device on a narrow SCSI channel, you will cut its potential maximum performance since it will only be able to send data 8 bits at a time. This is generally only a big issue for hard disk drives. The converse is not true, of course; putting narrow devices on a wide channel will not double their performance. Nice try though. ;^)

- **Device Addressing:** Narrow devices cannot see or access device IDs over 7. If you are going to use narrow devices on a wide host adapter, set the host adapter's device ID to something between 0 and 7 or the device won't work. (It's a good idea to just leave host adapters at device 7 period.)
- **Termination:** This is really the biggest issue with mixing devices. The problem is this: what happens to the extra 8 data bits present on the wide SCSI bus when they are connected to narrow devices? If you just connect the narrow data signals and leave the wide signals "hanging", then that part of the bus will be unterminated! Similarly, many wide devices, when connected to narrow buses, won't work properly if the high bits are not properly dealt with. Therefore, the wide data signals must be terminated when bus widths are mixed. This is often done by the adapters used for mixing devices, as long as you purchase the right hardware. Connectors that automatically terminate the extra "wide" signals are said to include *high byte termination*.



A wide/narrow adapter that includes high byte termination.

Original image © Computer Cable Makers, Inc. Image used with permission.

As you can see, mixing narrow and wide devices on the same cable can be a bit complicated. If you are having trouble getting a mixed configuration to work, try asking for help from the vendor that sold you your hardware. Whatever the problem, chances are that someone else has experienced it before you. :^)

Next: Using SCSI With IDE/ATA

Using SCSI With IDE/ATA

There are now many PC systems in common use that combine SCSI drives or devices with IDE/ATA ones. This is easier to do now that it has been in the past, but still takes more work than using only one or the other by itself. For the most part you end up having to do double the work since you have to configure two different interfaces. Still, there are situations where it makes sense. For example, you may want to add a SCSI optical disk or other device but continue using IDE/ATA hard disks, since they are significantly less expensive than SCSI ones. Many people also add IDE drives to existing SCSI systems to store infrequently-used large files, or for backup purposes.

In most cases, IDE and SCSI devices can be combined without too much difficulty. In particular, adding non-hard-disk SCSI devices to an existing IDE/ATA system rarely poses much of a problem. The difficulties seem to be largely confined to mixing SCSI and IDE/ATA hard disks. In particular, many people who mix SCSI and IDE hard disks want to boot from the SCSI drive, because it is probably the fastest one in the system. However, by default, PCs will look for and boot the first IDE/ATA drive they see in the system, since the system BIOS natively supports IDE/ATA and not SCSI. This causes fits for people who have been using an all-SCSI system and add an IDE drive, and find the PC now wants to boot the new drive.

There are a couple of ways to get around this. The easiest one is to use a system whose BIOS supports booting from SCSI instead of IDE in a system that has both. This is typically implemented via the boot sequence BIOS setting. Today, most newer PCs will support this feature, but some retail PCs don't provide the boot sequence feature. Many older PCs don't have this setting either.

Another option is to set up your drives so that only the SCSI drive has a bootable primary partition. Configure the IDE/ATA drive to only contain logical, non-bootable volumes in the extended partition. If you do this, the system may boot from the SCSI drive because it is the only one that is bootable. Of course, if the IDE/ATA drive already has a primary partition, you will have to use a third-party repartitioning tool to change it. This also may not work on all systems or with all operating systems; the boot sequence solution is preferable.

Finally, there can be complications if you try to use a SCSI host adapter and some types of add-in IDE/ATA controllers in the same PC. IDE cards like the Promise "Ultra" series appear to the system as if they were a SCSI card. If there is also a real SCSI host adapter, which drives are recognized first comes down to which card is seen first by the operating system at boot time. If you have this configuration and the drives are being seen in the "wrong order", you may be able to fix the problem by manually changing the various cards' resource settings. Swapping the PCI slots used by the two cards may also correct the problem.

Next: IDE/ATA vs. SCSI: Interface Comparison

Cut To The Chase

I still think it is best to read all of the other sections in this area and look at how IDE/ATA and SCSI compare in various respects. However, I realize that (as usual) I have made this large by blabbing too much and perhaps some people are looking for a quick answer. Just recognize that a short answer is rarely as good as a long one.

Overall, the SCSI interface is technologically superior to the IDE/ATA interface. It supports more devices, allows for better multitasking, more expansion, use of more high-end devices, more types of devices, and more performance-enhancing features. IDE/ATA is intended primarily for PCs that are not using a lot of peripherals, or for those that are cost conscious.

For the average person who is using their PC for light office work, games, internet use, etc., I still recommend IDE/ATA. The reasons are simple: cost and simplicity. Virtually all PCs today ship with IDE/ATA hard disks standard, and if you are using a small number of devices--say one hard disk and one CD-ROM drive, IDE/ATA offers more than adequate performance for the typical office or home user. As long as the machine is reasonably modern and the PC is set up properly, IDE/ATA will work without much difficulty, and there is no need to special-order or customize the machine, since it is really the "standard". The cost of a simple IDE/ATA setup is far less than an equivalent SCSI bus.

For the user who is performance-conscious, who will be doing real multitasking, using many devices at once, doing heavy development work, supporting multiple users at once on the machine, or who otherwise wants the best and is not afraid if it costs a few hundred dollars more, SCSI is the obvious choice. SCSI offers the most flexibility, the most choice of peripherals, and the best performance in a multitasking or multi-user environment.

So if you are looking to set up a low-end or middling system, I would say SCSI is out of the question. The simple reason is that for the additional cost of going to SCSI, you could probably improve overall performance more by buying a faster processor, more system RAM, or a faster hard disk. For a high-end system, SCSI has definite advantages and is preferred. It is those who are "in the middle" who might have a tough decision to make.

Also bear in mind that there are some peripherals that generally are only available on SCSI, or that have significant enough advantages on SCSI that if you want to use this type of device, SCSI is pretty much a must. Once you make the decision to go with SCSI, the cost of going to SCSI for your hard disks is reduced because you've already made the commitment for the host adapter, which is the large up-front cost of SCSI.

The final, real bottom line is: if you want it cheap and simple with good performance, use IDE/ATA. If you want maximum performance and flexibility and have the money to pay for it, use SCSI.

Next: Cost

Cost

In terms of cost, the IDE/ATA interface is superior to the SCSI interface in virtually every way. The only exception would be if you needed to use, say, 10 devices, in which case SCSI might be cheaper because this would require a special solution to do in IDE/ATA. I am assuming that nobody would use IDE/ATA for that many drives anyway.

Here are four specific reasons why SCSI is more expensive than IDE/ATA (there are probably others as well):

- **Additional Hardware:** SCSI setups require a host adapter, which means either an add-in card or a more expensive motherboard. Cables, terminators and adapters also add to the cost of most SCSI implementations.
- **Lower Volume:** Far fewer SCSI devices are sold than IDE/ATA devices. The price of an item manufactured in high volume is usually less than one manufactured in low volume.
- **Niche Market:** Since SCSI has a reputation for being higher-performance and is generally used by those who are less cost sensitive, sellers can afford to run higher margins and still make the sale, and will usually do so. People are willing to pay more for SCSI, and SCSI costs more because of this.
- **More Advanced Technology:** This is really a matter of appearances: since the performance-conscious use SCSI, it is the interface where the most advanced new drives will typically show up first. Newer and faster technology is more expensive than older and slower technology. The price gap between the cost of SCSI and IDE/ATA drives has actually increased over the last few years. Just remember that comparing a "36 GB IDE drive" to a "36 GB SCSI drive" is making an apples-and-oranges comparison, because the SCSI drive is almost certain to offer much more performance, and for reasons that have nothing to do with the interface.

For those for whom cost is an important consideration, IDE/ATA will win over SCSI virtually every time. For low-end systems, the extra funds required to go to SCSI will usually be better spent upgrading core parts of the system such as the processor or system RAM. This is important to remember: those building performance systems should make sure they have addressed proper component balance and adequate system memory and other key components before making the jump to SCSI.

Next: Performance

Performance

Comparing the performance of the SCSI and IDE/ATA interfaces is not an easy task. While those who favor SCSI are quick to say that it is "higher performance" than IDE/ATA, this is not true all of the time. There are many different considerations and performance factors that interact when considering the performance equation, because performance is so dependent on system setup and on what is being done with the PC. I will try to look at some of these factors and how they influence system performance for both interfaces:

- **Device Performance:** When looking at particular devices, there is theoretically no difference between SCSI and IDE/ATA. The device itself should be the same in terms of its internal performance factors. In practice, this is rarely the case. Many manufacturers only make a particular drive as SCSI or IDE/ATA, not both, so direct comparisons aren't easy. Since SCSI is known to be the choice for those seeking performance, higher-performance drives tend to show up on the SCSI interface well before they do on IDE/ATA (you pay for this performance, of course, but that's a separate issue). Another issue is the implementation of the integrated device controller logic and the interface chip. Some companies that produce the same device (the physical hard disk assembly, for instance) for both SCSI and IDE/ATA may do a much better job of writing the control logic for one interface than for another. In general, SCSI drives offer higher performance than IDE/ATA ones.
- **Maximum Interface Data Transfer Rate:** The interface or external data transfer rate describes the amount of data that can be sent over the interface. As described here, it's important not to place too much emphasis on the interface transfer rate if you are only using a small number of devices. Comparing SCSI and IDE/ATA, both interfaces presently offer very high maximum interface rates, so this is not an issue for most PC users. However, if you are using many hard disks at once, for example in a RAID array, SCSI will offer better overall performance.
- **Single vs. Multiple Devices and Single vs. Multitasking:** For single devices, or single accesses (as in DOS), in many cases IDE/ATA is faster than SCSI, because the more intelligent SCSI interface has more overhead for sending commands and managing the channel. If you are just using a single hard disk, or doing simple work in DOS or Windows 3.x where everything happens sequentially, most of the benefits of SCSI are lost. For multitasking operating systems, especially where transfers are occurring between multiple devices, SCSI allows multitasking and command queuing and reordering, which enables devices to set up multiple transactions and have them take place basically simultaneously. In contrast, IDE/ATA transactions to one device "block" the channel and the other device cannot be accessed. Putting two devices on two different channels allows simultaneous access, but severely restricts expandability. IDE/ATA still does not have the advanced features that SCSI has for handling multiple devices.
- **Device-Mixing Issues:** IDE/ATA channels that mix hard disks and CD-ROMs are subject to significant performance hits in some situations, due to the fact that these are really different protocols on the same channel. SCSI does not have this problem.
- **Technological Currency:** IDE/ATA has one big advantage over SCSI in terms of performance, if cost is a consideration. Both interfaces are constantly being updated to offer faster performance, both in terms of the interfaces themselves and the drives produced for them. However, to take advantage of these improvements requires additional hardware purchases. For SCSI, the extra investments are much more costly than IDE/ATA. If one is on a limited budget, it could well be argued that staying current with IDE/ATA technology will offer better long-term

performance than going with SCSI but only being able to upgrade every three or four years due to cost considerations.

Overall, SCSI *is* a higher-performance interface. For very simple applications, like a single hard disk and a single CD-ROM drive on different channels, IDE/ATA has a marginal advantage. For complex applications, SCSI has a significant advantage.

Next: Configuration and Ease of Use

Configuration and Ease of Use

Much like the performance issue, the winner here depends on how many devices you want to use. Both IDE/ATA and SCSI have had a rather spotty history in terms of their ease of setup and configuration, and both are much better today than they have been in the past. Overall, I would say that IDE/ATA is easier to set up, especially if you are using a reasonably new machine and only a few devices. IDE/ATA support is built into the BIOS, and there are fewer issues to deal with: far fewer different cable types, no bus to terminate, only one type of signaling, fewer issues with software drivers, and in general fewer ways that you can get yourself into trouble.

The difference between the interfaces is, if anything, increasing. Over the past few years IDE/ATA has in many ways become simpler to deal with, as manufacturers have agreed on standards and fixed problems with drivers and support hardware. SCSI has gotten more complex, especially now that new hard disks use LVD signaling, which is more complex to set up.

The configuration simplicity advantage for IDE/ATA drops off quickly if you want to get maximum performance while using more than a few devices. You then have to worry about where they are being placed on the channel, finding IRQs and other resources for multiple channels, etc. This can be done without too much difficulty, but there are many different things to take into consideration. In contrast, once SCSI is set up, you can put 7 devices on the bus (or 15 for wide SCSI) with very little effort, although you do have to watch the termination as you expand the bus.

SCSI has a significant advantage over IDE/ATA in terms of hard disk addressing issues. While IDE/ATA hard disks are subject to a host of capacity barriers due to conflicts between the IDE/ATA geometry specifications and the BIOS Int 13h routines, SCSI is not.

Next: Expandability and Number of Devices

Expandability and Number of Devices

On this particular score there isn't much of a competition: SCSI beats IDE/ATA hands down. While SCSI is more involved and expensive to set up, once you make the appropriate investments of money and time you get a bus that can be expanded relatively easily to either 7 devices (with narrow SCSI) or 15 (with wide SCSI, which is what most new systems use). On the other hand, IDE/ATA normally supports only four devices; you can expand this to eight if you add an after-market IDE/ATA controller, but that can introduce its own issues.

Of course, this advantage of SCSI only matters if you actually need this much expansion capability. For most users, four device expandability is certainly sufficient, and eight is definitely more than enough.

Next: Device Type Support

Device Type Support

At one point, SCSI held a significant advantage over IDE/ATA in terms of the types of devices each interface supported. Since SCSI is a high-level system interface used by performance machines, there has always been a wide variety of different kinds of hardware produced for the SCSI interface. In contrast, IDE/ATA began as a hard disk interface, and support for other types of hardware was only added later on. Even as late as 1997, there were many more hardware choices if you had a SCSI system than if you went with IDE.

This has changed in recent years. As the number of IDE/ATA systems on the market has grown, many manufacturers have migrated their devices to the IDE interface. A good example is that of CD-RW drives; a few years ago you needed a SCSI system if you wanted to use a CD-RW drive, but they are now commonly available for both interfaces. There are still more different device choices for SCSI than IDE, but the difference is less important than it once was.

One place where SCSI beats IDE/ATA easily is in support for external devices: IDE/ATA has none. SCSI drives can even be located in a different room from the machine that is using them, if that's an issue for some reason.

Next: Device Availability and Selection

Device Availability and Selection

SCSI leads IDE/ATA in terms of the number of different types of devices available, but often trails behind it in terms of the number of different models available for each device type. Since the IDE/ATA market is so much larger than the SCSI market, there are many more brands and types of various devices available for the IDE/ATA market than for SCSI.

One area where this can be an issue is with hard disks. Hard disk options for the IDE/ATA interface range from small value models to large performance units. For SCSI, there are fewer choices, particularly on the low end of the scale. This means that you will have more difficulty finding economical drives for SCSI setups unless you go with older units (which is certainly an option). Of course, if you want the fastest technology, SCSI gives you more choices.

Of course, this isn't to say that your choices with SCSI are necessarily few; there are many different companies producing SCSI hardware. You will have more difficulty trying to set up SCSI *inexpensively*, but that's sort of par for the course. :^)

Next: Software / Operating System Compatibility

Software / Operating System Compatibility

For many years, there were significant support and compatibility issues with SCSI that did not occur with IDE/ATA. Since IDE was the "native" hard disk interface for most PCs, and was by far the most common interface used, all software worked with it. SCSI on the other hand was much less common in PCs, especially in their earlier years, and so there were occasional support issues. Extra drivers or special software would be needed to get older operating systems like "straight DOS" and Windows 3.x to function with SCSI at all.

Today, this is pretty much a non-issue. Modern operating systems all provide support for SCSI host adapters and devices. You may need to install a driver or some utility software, but if you stick with well-known brand names even this may not be necessary.

Next: System Resource Usage

System Resource Usage

Generally speaking, SCSI is superior to IDE/ATA in terms of how many system resources are used. If you are only using a single IDE/ATA channel, the two are basically a wash in terms of resource usage. However, once you go to a dual IDE channel situation you will generally consume more resources than SCSI uses--and most PCs have dual channels by default, unless you disable one. If you were ever to set up a four-channel IDE implementation you would be using significantly more resources than if you had just set up a SCSI bus.

There are in fact some people who set up SCSI specifically to get around the system resource constraints for which the PC is "famous", and which using multiple IDE/ATA channels exacerbates. Doing this also allows you to worry less about needing to take more resources in the future if you expand to many different devices.

There is one system resource issue involved in using SCSI under DOS or Windows 3.x, however, that doesn't apply to newer Windows operating systems. Both of the older operating systems generally require a driver in order to use SCSI, which can take up a decent-sized chunk of conventional or upper memory. The IDE interface does not normally have this requirement. This is only an issue for some systems, of course, and the importance of conventional memory has diminished somewhat today, as those older operating systems are less and less used.

Next: Support for Non-PC Platforms

Support for Non-PC Platforms

While not an issue for most people, there are some for whom support for devices by machines outside the PC platform can be an issue. For these applications, SCSI is almost always a better choice than IDE/ATA. SCSI is more universally found outside the PC world. It is used by many different manufacturers of computer hardware, including the Apple platform, many types of UNIX workstations, minicomputers and more. IDE/ATA is mostly a PC interface, though Apple has started using the interface as well in recent years (likely to reduce cost).

For applications where devices or data must be shared between machines, external SCSI devices give you the flexibility to dynamically share or move information from one platform to another. It's even possible to have more than one computer on the same SCSI bus, something not possible with IDE/ATA at all. Of course, compatibility in terms of software and file systems is still an issue to be dealt with.

Next: Summary of IDE/ATA and SCSI Comparisons

Summary of IDE/ATA and SCSI Comparisons

The following table summarizes the comparison of SCSI and IDE/ATA. See the individual sections for a more thorough explanation of the summary conclusions below:

Interface Factor	IDE/ATA	SCSI
Cost	Low	Moderate to high
Performance	High for single devices or single tasking, moderate to low for multiple devices or multitasking	High in most situations
Ease of Configuration and Use	High for small number of devices, moderate for large number of devices	Low to moderate for both small and large numbers of devices
Expansion and Number of Devices	Moderate	High
Device Type Support	Moderate	High
Device Availability and Selection	High	Moderate

Software / Operating System Compatibility	High	Moderate to high
System Resource Usage	Moderate to poor	Good
Support for non-PC Platforms	Moderate	Good

Next: Hard Disk Logical Structures and File Systems

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 access to the storage on a hard disk system. 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 same hard disk can be arranged in many different ways, and even multiple ways in different areas of the same disk. 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. I begin with a discussion of different PC operating systems, and an overview of different file system types. I then go into significant detail describing the major structures and key operating details of the most common PC file system, FAT (FAT12/FAT16/VFAT/FAT32). I talk about utilities used for partitioning and formatting hard disks, and also talk a bit about disk compression (even though it is no longer nearly as important as it once was.) I place special emphasis on how to organize the disk for maximum performance--while not getting bogged down in the minutiae of optimization where it will buy you little.

Most of the focus in this section is on the FAT family of file systems, because these are by far the most commonly used, and also the ones with which I am most familiar. I do mention alternative file systems, but do not go into extensive detail on them, with one exception. Recognizing the growing role of Windows NT and Windows 2000 systems, a separate, comprehensive section has been added that describes the NTFS family of file systems. If you are mostly interested in reading about NTFS, you may want to skip some of the earlier subsections that describe FAT, and skip directly to the NTFS material. Bear in mind, however, that some of the NTFS discussions build upon the descriptions of FAT, since in some ways the file systems are related. So I recommend reading the section in order, if possible.

Note: If you are running a less-common operating system such as OS/2, Linux or BeOS, you likely know more about those operating systems than I do. :^) While I have done my best to research file system support for the alternative operating systems, my personal experience with them is limited. I decided it was better to mention these operating systems and cover their file system support to the best of my knowledge, rather than just leave them out. However, I probably missed something somewhere, so if you find any mistakes, please let me know.

Next: Operating Systems and File Systems

Operating Systems and File Systems

The *operating system* is the large, relatively complex, low-level piece of software that interfaces your hardware to the software applications you want to run. The operating system you use is closely related to the file system that manages your hard disk data. The reason is a simple one: 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 a brief look at the most common operating systems in use on the PC, provides a bit of their history as it relates to file systems, and mentions the file systems that they use. The operating systems are listed approximately in the chronological order of when they were first used for PC hardware (although not exactly, since I chose to keep the discussions of the consumer Windows 9x/ME operating systems near each other, as this seems a more clear way to describe them). Note that different "sub-versions" of a particular operating system (such as different variants of Windows 95) may provide support for different kinds of file systems (thanks to the strange way that Microsoft sometimes chooses to operate... :^))

Next: DOS (MS-DOS, PC-DOS, etc.)

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

The very first operating system used on the earliest IBM PCs was called simply the *Disk Operating System*, abbreviated *DOS*. There are few PC users who have not heard of DOS; at one time it was used on pretty much every PC, and it is still around today in many different forms. Microsoft's version of DOS is the most common one, and is called MS-DOS--with the "MS" abbreviation being rather self-explanatory. :^) For a while IBM Corporation was producing a competing product called PC-DOS, and there are other companies' alternatives around as well.

As the name "DOS" implies, the use of disks is an inherent part of the DOS operating system. The FAT file system has been an essential component of DOS since the beginning, and generally speaking, DOS uses only various versions of FAT for managing files. Different flavors of FAT (sorry :^)) are supported by different versions of DOS, as follows:

- **DOS 1.x and 2.x:** These ancient DOS versions support only the FAT12 file system, used today primarily for floppy disks. If you're still using DOS 1.x or 2.x, man do *you* need a new computer! :^)
- **DOS 3.x through 6.x:** These are more common versions of DOS for older PCs running either straight DOS or Windows 3.x. DOS 6.x was especially popular; millions of copies of these operating systems were sold. DOS 3.x through 6.x support the older FAT12 and the newer FAT16, which was the file system standard for many years in the PC world.
- **DOS 7.0:** MS-DOS 6.22 was the last "standalone" version of DOS sold by Microsoft. After DOS6.22, Microsoft sold DOS only as the

underpinnings of other operating systems, such as Windows 95/98/ME. The first of these was MS-DOS 7.0, which runs "underneath" the first Windows 95 version (Windows 95A). It supports FAT12, FAT16 and VFAT, the enhanced version of FAT that includes support for long file names.

- **DOS 7.1 and later:** These versions of DOS underlie Windows versions from Windows 95 OEM Service Release 2 (Windows 95B) and later. They support FAT12, FAT16, VFAT and FAT32.

The most confusion is caused by the "appearance" of FAT32 support in the second release of Windows 95--corresponding to DOS 7.1--which was complicated by the fact that later versions of Windows that support FAT32 were not "officially" sold to the public. See the discussions of Windows 95A and Windows 95B and 95C for more information.

Tip: You can check the version of DOS your system is running by using the "VER" command from any DOS command line. Note that this may not work for DOS running under a Windows version, as Microsoft desperately wants people to think that Windows 95/98/ME do not run on the ancient DOS platform. ;^)

Next: Windows 3.x

Windows 3.x

Microsoft's first foray into the world of the graphical operating system was Microsoft Windows 1.0. I never used this product, but it is universally considered to have been rather scary. :^) Bill Gates and company did not give up, and subsequent versions of Windows followed. Microsoft finally started to pick up a head of steam with the release of Windows 3.0. The versions that followed, including Windows 3.1, Windows 3.11 and Windows for Workgroups 3.11, were the most common graphical operating systems used in the early 1990s, prior to the creation of Windows 95. These are often collectively called *Windows 3.x*.

To the technical purist, Windows 3.x isn't a true "operating system". The reason is that it runs strictly on top of DOS, and uses DOS (and BIOS) facilities and routines for most of its hardware management, including disk access. For this reason, some consider it just a "graphical shell". Another famous "Bill" made the point that names don't matter all that much, but for our purposes, the matter of what Windows 3.x really is does matter. Since it uses DOS for disk access, this means that Windows has the same file system support as whatever version of DOS underlies it. In most cases that is MS-DOS 6.x, most commonly MS-DOS 6.22. See this discussion of DOS file system support for more information.

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, partial 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 still "plain" FAT, so special features like long filenames are not included.

Next: Windows 95A (Initial Release)

Windows 95A (Initial Release)

The early versions of Windows that Microsoft created in the early 1990s represented a significant improvement to those who had been used to the mostly text-based and single-tasking environment that DOS represented. (Of course, Microsoft didn't exactly invent the graphical operating system! They just used their marketing muscle to make Windows the standard on the PC desktop. But that's a different matter altogether. :^)) Still, early versions of Windows were very rudimentary in a number of respects. They ran on top of DOS and were limited to 16-bit applications. Multitasking capability was limited and problems were frequent.

In 1995, Microsoft introduced Windows 95, which represented the "next step" towards a comprehensive consumer-oriented graphical operating system for PCs (Windows NT had already been created at the time, but was geared towards businesses and servers.) 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. As mentioned in the discussion of DOS, Windows 95 in fact includes a version of DOS, that is designed to work with it and its file structures.

When Windows 95 was released, it came with a new and updated version of the traditional FAT file system: VFAT. At the same time, Windows 95 was compatible with older FAT12 and FAT16 partitions and disks The initial version of Windows 95 is now sometimes called "Windows 95A" to distinguish it from later editions, or "Windows 95 Retail" in recognition of the fact that it was the only revision of Windows 95 officially sold to the public. It's important to remember that this version of Windows 95 does *not* support the now ubiquitous FAT32 file system. Since modern hard disks basically require FAT32 (or NTFS) for reasonably convenient management, those still running the first version of Windows 95 may need to consider an upgrade (Windows 98 or Windows ME being the successors to Windows 95, but certainly not the only choices.)

Tip: If you are not sure of which version of Windows 95 is on your PC, use this procedure to find out.

Next: Windows 95B and 95C (OEM SR 2.x)

Windows 95B and 95C (OEM SR 2.x)

Windows 95 was a big success for Microsoft, and I would say this was for good reason: while certainly sharing many of the flaws associated with other Microsoft operating systems, Windows 95 was a significant improvement over Windows 3.x in virtually every way. However, Microsoft developed a problem with the operating system as time went on, specifically related to the file systems it supported. As hard disks grew in size, they began to approach the maximum size allowed by a FAT16 partition. This meant that PC makers were forced to divide the drives in their new systems into multiple partitions, which was extra work and which some customers didn't like. People buying new hard disks also had a similar problem. Since hard disks were only going to keep getting bigger, Microsoft had to do "something".

That "something" came in the form of the FAT32 file system, which allows for much larger single partitions than the older 16-bit version of FAT. FAT32 support was included in an updated version of Windows 95 that Microsoft released in 1996, along with some other new features. In a move that was controversial at the time, Microsoft decided to make these new operating systems available only to OEMs (original equipment manufacturers, in this case PC and PC component makers)--no retail version was ever created. The new version of Windows 95 was called "Windows 95 OEM Service Release 2", which is usually abbreviated to "Windows 95 OEM SR2" or "Windows 95 OSR2". It is also sometimes called "Windows 95B".

OSR2 was never sold to the public at retail, so if you wanted to use the newer version with FAT32 support you had to either buy a new PC, motherboard or hard disk, or buy a "gray market" copy from a retailer willing to break Microsoft's rules (and doing this was quite popular in 1996 and 1997). Microsoft never really explained *why* they refused to make FAT32 available to the unwashed masses, but speculation is that they didn't want to expend the energy involved in doing a full quality assurance cycle to ensure that it would work with all the hardware in use at the time. By restricting it to OEMs and new systems, they didn't have to worry as much about whether OSR2 would work with all older hardware and software, and could even push some of the validation effort onto the OEMs themselves.

Between 1996 and 1997, Microsoft actually released three slightly different variants of Windows 95 OSR2:

- **OEM SR2.0:** This is the first OSR2 version, also called "Windows 95B".
- **OEM SR2.1:** This is OSR2.0 with the addition of a patch to allow (rudimentary) USB support within Windows 95. Also called "Windows 95B".
- **OEM SR2.5:** This was produced at around the time that Microsoft became obsessed with tying browsers into their operating system. :^) It includes everything from OSR2.1 and also Internet Explorer 4. This variant is often called "Windows 95C".

Note that all of these are identical, except as noted, and they all support the FAT12, FAT16, VFAT and FAT32 file systems.

Well, I included all this verbiage so that you would understand all of the various issues surrounding FAT32 support under Windows 95. The matter of FAT32 not being available at retail became a non-issue when Microsoft introduced Windows 98, which all but removed Windows 95 OSR 2.x from the market. Of course, millions of PCs are still running this operating system. If you have the older Windows 95 (now called "Windows 95A") and want FAT32 you should upgrade to Windows 98 or Windows ME, not Windows 95 OSR 2.x (unless there is a specific reason why you prefer Windows 95 of course!)

Next: Windows 98, Windows 98 Second Edition and Windows ME

Windows 98, Windows 98 Second Edition and Windows ME

Microsoft took a lot of heat when it created Windows 95 OEM SR 2, because they decided not to sell that version of the operating system at retail. As hard disks got bigger and bigger, demand for FAT32 support increased. Unfortunately, as a "regular customer" of Microsoft's, you would have no way to get FAT32 support, unless you chose to buy new hardware or buy a copy that had been sold contrary to Microsoft's licensing rules. As a result, 1996 and 1997 were two years of confusion in the area of file system support on Microsoft operating systems. :^) This situation persisted until 1998, when Microsoft released the first version of Windows 98.

Windows 98, among other improvements to Windows 95 (and some subtractions, depending on your perspective) includes full support for FAT32, and was definitely sold at retail. :^) The operating system also provides the capability of converting an existing FAT16 partition to FAT32. This was the first Microsoft operating system to provide complete "official" support for FAT32. It also of course supports the older FAT variants.

In 1999, Microsoft released Windows 98 Second Edition ("Windows 98 SE"), which in this author's opinion was little more than a glorified bug patch that Microsoft was able to sell as a new product. In 2000 came Microsoft Millennium Edition ("Windows ME") which is another evolutionary tweak on the Windows 98 OSes. Neither of these new operating systems added any additional file system support (and as far as I am concerned, neither added anything of much substance in *any* way, though others would probably disagree with that...)

Next: Windows NT

Windows NT

All of the other operating systems that I have mentioned in the earlier pages in this section are, in one form or another, evolutionary enhancements that hearken back to the very first version of MS-DOS 1.0, written in 1981. This includes Windows 3.x, Windows 95, Windows 98 and Windows ME. While Microsoft would like very much for people to forget this fact, I refuse to. :^) These operating systems are generally capable for most users, but are not designed for the rigors of heavy use. In the early 1990s, Microsoft recognized that if it wanted to tackle the growing (and lucrative) business and corporate market, a more advanced, capable, secure and scalable operating system would be required.

The result was a new version of Windows, designed and built from the ground up. This new operating system was released in 1993 as *Windows NT*, with the "NT" supposedly standing for "New Technology" (though since its arrival was delayed for some time, pundits began to joke that it really stood for "Not There". ;^)) Windows NT was specifically designed for the corporate environment, and intended for use on high-powered servers and workstations. Unlike the "consumer" Windows operating systems, it is *not* based on MS-DOS, though it can run some MS-DOS programs (through emulation of a virtual MS-DOS machine.)

For compatibility, Windows NT supports the older FAT family of file systems, but not FAT32. However, NT's intended file system of choice is the highly-capable NTFS, which is discussed in this section of the site. Many of the more advanced features of Windows NT are in fact tied to the use of the NTFS file system. However, FAT support provides flexibility for certain applications, especially ones that involved multiple operating system installs on the same PC. Note that even though Windows NT will read both FAT and NTFS partitions, the two file systems are not compatible with each other.

Several versions of Windows NT were created. The first was version 3.1, which was named to coincide with Windows 3.1, the then-current consumer operating system. Windows NT 3.5 followed shortly thereafter, and then Windows NT 3.51. All of these early NT versions had some limitations, and used the older Windows 3.1-style interface that many people found somewhat lacking. They support FAT and NTFS partitions, as mentioned, and will also support HPFS, the native file system of IBM's OS/2.

NT came into its own with Windows NT 4.0, which was released in 1996 and became very popular in the late 1990s. It supports FAT and NTFS, like the earlier NT versions, but *not* HPFS, support for which was removed in version 4.0. From a file system support standpoint, Windows NT 4.0 was perceived as having one major weakness, and that was lack of support for FAT32 (support for FAT32 under Windows NT was possible through the use of third-party drivers, but not natively). As the 1990s closed, Windows NT became somewhat dated, and was "replaced" by Windows 2000, which included FAT32 support (amongst other things).

Next: Windows 2000

Windows 2000

Windows NT was very successful for Microsoft through the 1990s, but the software giant didn't rest on its laurels. As Windows NT 4.0 began to age, certain flaws began to show, including a lack of support for the latest hardware and other limitations. From a file systems perspective, the most important was the lack of support for FAT32. Microsoft addressed some of these through the use of service packs, but mostly concentrated on the next version of the operating system. It had been unofficially called "Windows NT 5.0" for some time, but Microsoft instead called the new operating system *Windows 2000*.

Windows 2000 builds upon Windows NT 4.0 in most respects, and differs from the older operating system in two ways when it comes to file systems. The first is the addition of support for FAT32, as I hinted at in the preceding paragraph. :^) This was a much-desired change, especially with FAT32 all but replacing FAT16 in newer Windows 9x/ME systems. The other was that NTFS under Windows 2000 was enhanced, through the creation of the NTFS 5.0 version of that file system. Windows 2000 will still read older NTFS partitions, but it must be installed on an NTFS 5.0 partition; NTFS 5.0 is Windows 2000's "preferred" file system.

Note: For a full discussion of NTFS, see this section of the site. For more on NTFS versions, see here.

Next: OS/2

OS/2

In the early 1990s, two of the biggest names in the PC world, IBM and Microsoft, joined forces to create OS/2, with the goal of making it the "next big thing" in graphical operating systems. Well, it didn't quite work out that way. :^) The story behind OS/2 includes some of the most fascinating bits of PC industry history, but it's a long story and not one that really makes sense to get into here. The short version goes something like this:

1. Microsoft and IBM create OS/2 with high hopes that it will revolutionize the PC desktop.
2. OS/2 has some significant technical strengths but also some problems.
3. Microsoft and IBM fight over how to fix the problems, and also over what direction to take for the future of the operating system.
4. Microsoft decides, based on some combination of frustration over problems and desire for absolute control, to drop OS/2 and focus on Windows instead.
5. IBM and Microsoft feud.
6. IBM supports OS/2 (somewhat half-heartedly) on its own, while Microsoft dominates the industry with various versions of Windows.

Now, OS/2 afficionados will probably take issue with at least some of that summarization, but that is what happened in a nutshell, or at least I think so. :^) At any rate, OS/2 continues to be supported today, but really has been relegated to a niche role. I don't know how long IBM will continue to support it.

OS/2's file system support is similar, in a way to that of Windows NT's. OS/2 supports FAT12 and FAT16 for compatibility, but is really designed to use its own special file system, called HPFS. HPFS is similar to NTFS (NT's native file system) though it is certainly not the same. OS/2 does not have support for

FAT32 built in, but that there are third-party tools available that will let OS/2 access FAT32 partitions. This may be required if you are running a machine with both OS/2 and Windows partitions. I believe that OS/2 does not include support for NTFS partitions.

Note: As I mentioned in the introduction to this section, I have little personal experience with OS/2, so if anything has changed with respect to file system support in OS/2, please feel free to let me know.

Next: UNIX / Linux

UNIX is one of the very oldest operating systems in the computer world, and is still widely used today. However, it is not a very *conspicuous* operating system. Somewhat arcane in its operation and interface, it is ideally suited for the needs of large enterprise computing systems. It is also the most common operating system run by servers and other computers that form the bulk of the Internet. While you may never use UNIX on your local PC, you are using it indirectly, in one form or another, every time you log on to the 'net.

While few people run UNIX on their own systems, there are in fact a number of different versions of UNIX available for the PC, and millions of PC users have chosen to install "UNIXy" operating systems on their own desktop machines. There are dozens of variants of the basic UNIX interface; the most popular one for the PC platform is *Linux*, which is itself available in many flavors. While UNIX operating systems can be difficult to set up and require some knowledge to operate, they are very stable and robust, are efficient with system resources--and are generally free or very inexpensive to obtain.

UNIX operating systems are designed to use the "UNIX file system". I put that phrase in quotes, because there is no single UNIX file system, any more than there is a single UNIX *operating* system. However, the file systems used by most of the UNIX operating system types out there are fairly similar, and rather distinct from the file systems used by other operating systems, such as DOS or Windows.

As an operating system geared specifically for use on the PC, Linux is the UNIX variant that gets the most attention in PC circles. To improve its appeal, the programmers who are continually working to update and improve Linux have put into the operating system compatibility support for most of the other operating systems out there. Linux will read and write to FAT partitions, and with newer versions this includes FAT32. I believe Linux can also read and write HPFS volumes, and can read NTFS volumes as well, but not write to them. Even BeFS is now supported. Some of these may require special drivers or utilities to be added to a basic Linux install.

While I use UNIX every day (The PC Guide runs on a UNIX server) my experience with Linux specifically is limited. Fortunately, one of the best things about Linux is that there is a myriad of helpful information about it all over the Internet: a popular one is *slashdot.org*. Since the Linux operating system is constantly being updated, your best bet is to hook up with one or more of the better Linux web sites to keep yourself informed on the happenings in this alternative operating system for PCs.

Next: BeOS

BeOS

In 1996, a new operating system was introduced to the world by a company called simply "*Be*". Its product, the *BeOS*, was intended to be an alternative to the "big" operating systems used by most people. Support for PowerPC (Apple hardware platform) came first, and then in 1998, BeOS for x86 (PC) computers was released. BeOS was developed from the "ground up" and was intended to be especially suited for dealing with multimedia applications, such as video and audio, on consumer-level hardware. To the PC user dissatisfied with Microsoft operating systems, BeOS represents another important option. While still a very small percentage of the overall operating system market (and for that reason mostly ignored in traditional hardware and software circles) it is the choice of a number of PC hobbyists and power users.

BeOS's primary file system is its own, the *BeOS File System*, alternately abbreviated as *BFS* or *befs*. This file system is unique to BeOS, but from my understanding most resembles a UNIX-type file system. One of BeOS's strengths, however, is that to encourage its use by those already running other operating systems, Be has built into BeOS the capability of accessing a multitude of other file systems. BeOS can read or write to FAT12, FAT16, VFAT and HPFS partitions. Support for FAT32 has either been added to the operating system recently, or is available through a third-party add-on. BeOS can even read NTFS partitions with the appropriate tool added, though it cannot write to them.

My experience with BeOS would be best categorized as follows: "Between Slim and Nil, with Slim having just left town". (Hey, honesty is supposed to be a virtue. :^)) Any of you BeOS aficionados who find anything incorrect in what I have written above, please let me know. While I don't use BeOS myself I think it is important that alternative operating systems be at least given a fair mention when discussing PC hardware.

Next: PC File Systems

PC File Systems

As I described in the introduction to this part of The PC Guide's hard disk coverage, the file system refers to structures and software used to organize and manage your data and programs on your hard disk. Most people are familiar with only the most common file system family (FAT and its variants) which are used on most PC-platform machines. These common file systems are the ones that get most of the attention, while others are largely ignored; I have been guilty of this myself on this site. However, there are in fact many different types of file systems in use by different operating systems for PC hardware. Many PC users actually employ more than one type of file system, to handle different tasks.

In this section I provide an overview of most of the various file systems that are commonly (or even less commonly) used on PC hardware. This includes a look a look at the various types of FAT file systems, as well the NTFS file systems used by Windows NT and Windows 2000. I also discuss several less common file systems: HPFS, UNIX file systems and the BeOS File System.

Note: In this section I only provide a brief overview of the major characteristics of each file system. The remainder of the subsections in the Hard Disk Logical Structures and File Systems section expand on the details of the FAT family of file systems, and there is also a separate section with details on NTFS. For the less common file systems, coverage is mostly limited to the appropriate page in this subsection of the site.

Next: File Allocation Table File System (FAT, FAT12, FAT16)

File Allocation Table File System (FAT, FAT12, FAT16)

The most common file system in the PC world is actually a family of file systems. The basic name for this file system is *FAT*; the name comes from one of the main logical structures that the file system uses: the file allocation table. This file system is the one that was used by DOS on the first IBM PCs, and it became the standard for the PCs that followed. Today, most PCs still use a variant of the basic FAT file system created for those early machines over two decades ago.

The most common name of the file system, "FAT", is problematic, even though it is still often used. The first FAT file system used 12-bit file allocation tables; this was later expanded to 16 bits, and became the most common file system implementation for hard disks from the late 1980s to the late 1990s. To distinguish these versions of FAT from the 32-bit successor called FAT32, the older FAT variants are now sometimes called FAT12 or FAT16. However, you will still hear just "FAT" used a lot; if so, you need to find out what specifically is being referred to, if it matters in that particular context. For more elaboration on the differences between FAT12, FAT16 and FAT32, see this page.

While designed specifically for MS-DOS and used by all of the Microsoft operating systems that followed (in one way or another) FAT's ubiquitousness has caused support for it to become an important characteristic for other operating systems as well. Virtually all PC operating systems can support the basic FAT file system, even ones that bear little resemblance to Microsoft DOS or Windows. FAT12 and FAT16 partitions are the "least common denominator" of file systems--while they are limited in many ways, they are the easiest file systems to use when compatibility is a concern. Even non-PC platforms can in many cases read disks formatted in the FAT file system.

Note: Most of the other subsections of the Hard Disk Logical Structures and File Systems section describe the structures and operation of the FAT file system family in more detail, so I will not get into a larger discussion of FAT here. Please refer to the other sections for details on how FAT works.

Next: Virtual FAT (VFAT)

Virtual FAT (VFAT)

When Microsoft introduced Windows 95 in, well, 1995, they made several improvements to the operating system when compared to its predecessors. One of the changes made was an enhancement to the classical FAT16 file system that had been in use up until that point. The new variation of FAT was called *Virtual FAT* or *VFAT* for short. (Note that many people use the terms "FAT" and "VFAT" interchangeably, even though they are technically not the same. All "FAT" partitions created under Windows 95 or later operating systems are really VFAT partitions.)

VFAT has several key features and improvements compared to FAT12 and FAT16:

- **Long File Name Support:** One of the most annoying limitations of operating systems prior to Windows 95 was the eleven-character file name restriction. For most people, VFAT's most important accomplishment was enabling the use of long file names by the Windows 95 operating system and applications written for it, while maintaining compatibility with older software that had been written before VFAT was implemented.
- **Improved Performance:** The disk access and file system management routines for VFAT were rewritten using 32-bit protected-mode code to improve performance. At the same time, 16-bit code was maintained, for use when required for compatibility.
- **Better Management Capabilities:** Special support was added for techniques like disk locking, to allow utilities to access a 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. Most of the new capabilities relate to *how* the file system is used, and not the actual structures on the disk 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. See the discussion of long file names for more information on how VFAT implements longer file names under Windows 9x/ME.

Note that as with FAT, much of the rest of the discussion of file systems in other subsections in this area of the site deals with VFAT.

Next: 32-Bit FAT (FAT32)

32-Bit FAT (FAT32)

As hard disks continued to increase in size through the 1990s, the limitations of the FAT16 and VFAT file systems began to become obvious. The use of large cluster sizes led to a significant amount of wasted hard disk space (slack). Eventually, hard disk manufacturers started to create drives so large that FAT16 could not be used to format a whole drive in a single partition. PC makers complained that FAT16 was unwieldy for modern machines, and users were confused with PCs that came with what seemed like two or three different "hard disks" (which were of course really just different partitions created to get around the FAT16 volume size limit.)

To correct this situation, Microsoft created *FAT32*. This newest FAT variant is an enhancement of the FAT/VFAT file system (even though the "V" was dropped from the name, FAT32 is based more on VFAT than FAT). It is named FAT32 because it uses 32-bit numbers to represent clusters, instead of the 16-bit numbers used by FAT16. FAT32 was created primarily to solve the two problems mentioned above. It allows single partitions of very large size to be created, where FAT16 was limited to partitions of about 2 GiB. It also saves wasted space due to slack when compared to FAT16 partitions, because it uses much smaller cluster sizes than FAT16 does.

FAT32 was first introduced in Windows 95's OEM Service Release 2, and was originally available only in later versions of Windows 95 when purchased from a hardware manufacturer. FAT32 support was later included in Windows 98, Windows ME and Windows 2000 as well. Many non-Microsoft operating systems can also now either read from or read/write FAT32 partitions. With hard disk sizes headed into the stratosphere, FAT32 has all but replaced FAT16 for those using "consumer grade" Microsoft operating systems.

Note: Operating systems older than Windows 95 OSR2, including the first version of Windows 95 ("Windows 95A"), Windows 3.x, and all versions of DOS before version 7.1, cannot read FAT32 partitions.

Aside from the difference in the way clusters are assigned and numbered, FAT32 is at its essence the same as regular VFAT, and the descriptions of FAT file structures apply to FAT32 as well in most cases. There are, however, some minor structural differences between FAT32 and its predecessors, in areas such as the use of file allocation tables, and the location and size of the root directory. The matter of using regular FAT16 vs. FAT32, and choosing partition and cluster sizes, is discussed in detail here.

Next: New Technology File System (NTFS) Version 1.1 / 4.0

New Technology File System (NTFS) Version 1.1 / 4.0

When Microsoft created Windows NT, it built the operating system pretty much from scratch--it was based on certain existing concepts, of course, but was totally different from older Microsoft operating systems. One of the key elements of NT's architecture was the file system created especially for the operating system, called the *New Technology File System* or *NTFS*. NTFS enables many of the goals of Windows NT to be realized.

NTFS is a much more complex and capable file system than any of the FAT family of file systems. It was designed with the corporate and business environment in mind; it is built for networking and with the goals of security, reliability and efficiency. It includes many features, including file-by-file compression, full permissions control and attribute settings, support for very large files, and transaction-based operation. 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. Its most significant drawbacks are increased complexity, and less compatibility with other operating systems compared to FAT.

The NTFS file system actually has more than one version. The one used by Windows NT is commonly called either version 1.1 or version 4.0, and has a few less features than the newer NTFS 5.0 used by Windows 2000. You can find a full discussion of NTFS in this subsection of the site, and a discussion of different NTFS versions and their differences here.

While originally created for Windows NT and now used primarily by Windows NT and its successor, Windows 2000, limited NTFS support has also been added to other operating systems. For example, operating systems like Linux (as well as some other UNIX variants) and BeOS can read NTFS partitions, which helps with the interoperability of different operating systems on the same machine. However, non-Microsoft operating systems usually cannot *write* to NTFS partitions, generally because of the security features built into NTFS by Microsoft.

Next: New Technology File System (NTFS) Version 5.0

New Technology File System (NTFS) Version 5.0

With the creation of Windows 2000, Microsoft added several new features to the operating system originally known as Windows NT 5.0. Several of these features are tied closely to the characteristics of the file system, which Microsoft updated at the same time. The new variation of NTFS is called *NTFS 5.0*. In fact, Windows 2000 relies on the changes that NTFS 5.0 includes so much that NTFS 5.0 is required; the operating system will convert older NTFS volumes to NTFS 5.0, and in some cases NTFS is required to enable key Windows 2000 capabilities.

NTFS 5.0 includes several new and useful features compared to the older NTFS versions used in Windows NT 4.0 and earlier. You can find a full discussion of NTFS in this area of the site, and a discussion of different NTFS versions and their differences here.

NTFS 5.0 was designed for Windows 2000 and that is the only operating system that provides full support for the file system. Windows NT 4.0 can also access NTFS 5.0 partitions if the operating system is updated with service pack #4 (SP4), however, it cannot make use of the new features that NTFS 5.0 facilitates. Much in the way that alternative operating systems have had third-party drivers created for them to allow them to read the older NTFS file systems, I would imagine that similar drivers will be created to allow Linux, BeOS and other operating systems to read NTFS 5.0 partitions, if this has not been done already.

Next: High Performance File System (HPFS)

High Performance File System (HPFS)

When Microsoft and Intel created OS/2, they sought to create an operating system that was more capable than the DOS and Windows versions then available. At the time, the only file system widely used on the PC was the FAT file system, which had a number of significant limitations. The FAT file system was restricted in terms of the size of partitions it could support, only allowed 11-character names, and it had none of the organization, security and reliability features that are so important to corporate, business and individual "power" users. To address these concerns, a new file system was created specifically for OS/2: the *High Performance File System* or *HPFS*.

HPFS offers many significant improvements over the FAT file system. These include the following:

- Support for long file names (up to 254 characters) and mixed-case file names.
- More efficient use of disk space, since files are stored on a per-sector basis instead of using multiple-sector clusters.
- Better performance due to overall design, including an internal architecture designed to keep related items closer to each other on the disk volume.
- Less fragmentation of data over time, and less need to defragment the file system.

Considering how much more advanced it was than FAT, one might have expected that HPFS would have become quite popular. Unfortunately, HPFS's wagon was hitched to OS/2, and for a number of reasons (many of them related to politics between IBM and Microsoft) OS/2 never really caught on. As interest in OS/2 waned, so did support for HPFS. While early versions of Windows NT (3.51 and earlier) supported HPFS volumes natively, this support was removed in Windows NT 4.0. Support for HPFS volumes can be added to some non-OS/2 operating systems by using third-party support software, but overall this file system seems to be firmly in a "niche status". Many of its features were incorporated into NTFS by Microsoft, one should note--or at least, NTFS has some definite similarities to HPFS.

You can find more information about HPFS by reading *this HPFS FAQ*, which isn't "official" but seems to be well-written and fairly detailed. This page includes a discussion of the internal architectural characteristics of HPFS volumes. Note that you'll find them easier to understand if you comprehend the internal structures of FAT file systems first. Comprehending NTFS also helps. :^)

Next: UNIX / Linux File Systems

UNIX / Linux File Systems

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).

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. For more information, you may wish to consult *this page*, which provides a nice overview of the file system. *This page* provides more detail and also describes common UNIX file-manipulation commands. There are literally thousands of other sites and pages about UNIX on the Internet!

Next: BeOS File System (BFS)

BeOS File System (BFS)

The BeOS operating system is designed to use its own file system, called (unsurprisingly) the *BeOS File System*, abbreviated *BFS* or sometimes *befs*. BFS is based strongly on the fundamental design concepts of the UNIX file systems, so those experienced with UNIX will find much that is familiar in BFS. However, BFS has also been tailored to meet some of the goals of the Be operating system, so BFS also has some special features that reflect Be's objective of positioning BeOS as a multimedia-friendly operating system.

Some of the special characteristics of BFS include:

- The ability to represent multiple media devices as a single partition or volume.
- Support for advanced caching methods.
- Performance optimizations for multimedia applications.
- Portability; BFS partitions can be moved between different hardware platforms easily.

These are in addition to most of the benefits associated with UNIX file systems, though there are of course tradeoffs; it wouldn't be accurate to say that BFS was "better than" UNIX file systems, for example.

BFS is not used on many PCs, for the simple reason that BeOS is not widely implemented. To my knowledge, Windows operating systems cannot access BFS partitions, but someone may have written a third-party driver that will allow DOS or Windows to access BFS. There is also an interface that will allow BFS partitions to be used by Linux, which to some extent reflects the fact that both Linux and BeOS tend to attract the same user base--technically knowledgeable PC users looking for something outside the realm of Microsoft operating systems.

For more information on BFS, see *this page*.

Next: PC Operating System and File System Cross-Reference

PC Operating System and File System Cross-Reference

In separate sections, I discussed the various operating systems used by most PCs, and also the file systems used for most PC storage devices. In each case, I described which file systems went with which operating systems. There are a lot of operating systems and file systems, however, so I have included this summary cross-reference table, which shows at a glance what supports what:

Operating System	**FAT**				**NTFS**		**HPFS**
	FAT12	**FAT16**	**VFAT**	**FAT32**	**1.1 / 4.0**	**5.0**	
MS-DOS (Straight)	Yes	Yes, DOS 3.0 or later	--	--	--	--	--
Windows 3.x	Yes	Yes	--	--	--	--	--
Windows 95A	Yes	Yes	Yes	--	--	--	--
Windows 95B / 95C	Yes	Yes	Yes	Yes (OEM Only)	--	--	--
Windows 98 / 98SE / ME	Yes	Yes	Yes	Yes	--	--	--
Windows NT	Yes	Yes	Yes	--	Yes	Partial, with NT 4.0 SP4	Yes, patch or hack needed for NT 4.0
Windows 2000	Yes	Yes	Yes	Yes	Yes	Yes	Third Party?
OS/2	Yes	Yes	Yes	Yes	--	--	Yes
UNIX / Linux	Yes	Yes	Yes	Yes	Read Only	Read Only	Yes
BeOS	Yes	Yes	Yes	Yes	Read Only	Read Only	Yes

Note: Support for non-native file systems in the alternative operating systems such as OS/2, Linux or BeOS depends entirely on the version of the operating system being used. Later versions generally support more file system types. In some cases special modules must be added or modifications made to the source code.

Next: Major Disk Structures and the Boot Process

Major Disk Structures and the Boot Process

File systems organize hard disks and other storage devices so that they can be used. When you are dealing with large storage units in the gigabytes, such as modern hard disk drives, it is necessary to have a structured way of arranging the data. At the highest level, hard disks and other large media must sometimes be broken into multiple pieces for easier use. In any event, it is necessary to have some fundamental structures that make it possible for a PC to determine the characteristics of the file system on the hard disk, to permit the system to be started up and files accessed.

In this section I describe the major disk structures that organize data in the FAT family of file systems (FAT12/FAT16/VFAT/FAT32). I begin by describing the master boot record (MBR) and explaining the way that hard disks are organized into partitions. I also explain what the different partition types are, and what the limitations are in mixing them. I then discuss volume boot sectors and the role of the active partition. Finally, I describe the DOS boot process (which is also used for consumer versions of Windows) and conclude with a discussion of boot sector viruses.

Note: The descriptions in this section apply to the FAT file systems primarily. If you are using an operating system like Linux exclusively, then only some of the material in this section will be of relevance to you. However, all operating systems running on a PC use the high-level partition structure to some extent, because all file systems on a PC must fit into the overall partitioning scheme used for PC hardware (though this is a bit of an over-simplification.)

Next: Master Boot Record (MBR)

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*. (Though the master boot sector should not be confused with volume boot sectors, which are different.)

The master boot record is always located at cylinder 0, head 0, and sector 1, the first sector on the disk (see here for more on these disk geometry terms). 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. One of the partitions is marked as active, indicating that it is the one that the computer should use for booting up.
- **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.

Next: Primary, Extended and Logical Partitions

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 chunks, 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.

The rules that determine how partitions are used were set down very early in the original design of the PC, and have remained virtually unchanged since that time. Recall that when the PC was first invented there was only (really) one type of file system. Still, it was envisioned that in the future, multiple operating systems and/or file systems might be wanted on a single machine. Therefore, provision was made in the partitioning scheme to allow for several different partitions. The rules that govern partition setup are as follows:

- A maximum of four partitions can be placed on any hard disk. These are sometimes 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.

- Only one partition may be designated, at any given time, as *active*. That partition will be used for booting the system. See here for more on active partitions and switching active status between partitions.
- DOS (and the operating systems that depend on it for booting, which includes all consumer Windows operating systems) will only recognize the active primary partition. Any other primary partitions will be ignored.
- One of the four partitions may be designated as an *extended DOS partition*. This partition may then be subdivided into multiple *logical partitions*. This is the way that two or more logical DOS volumes can be placed on a single hard disk.

Alright, I realize that this is somewhat confusing, so let's take a look at some examples of systems, to show you how this scheme is used:

- **Single Partition Windows PC:** Many PCs have all of their disk space made into a single partition, and use one of the FAT file systems. Such a machine would have just a single FAT primary partition on it, and nothing else. The other three "slots" for partitions on the disk would be empty.
- **Multiple Partition Windows PC:** To use more than one partition at a time on a DOS/Windows system, two partitions are used. One is a regular DOS primary partition (which becomes the "C:" drive). The other is the extended DOS partition. Within the extended DOS partition, all the other logical drives are created. So a drive with four logical drive letters would have the first (C:) be the active primary partition, and the other three (D:, E: and F:) would be logicals within the extended DOS partition.
- **Multiple Operating System PC:** A system with multiple operating systems could use one primary partition for each of up to four different file systems.

If you want, you can also combine multiple partitions with multiple operating systems. For example, you could have a primary DOS partition, an extended DOS partition, and a Linux partition. Such setups are far less common than the simpler examples above.

The extended DOS partition causes most of the confusion on the part of those setting up new DOS/Windows PCs. It really functions as a "container" that holds all DOS partitions except for the first (primary) volume. The reason that this structure was used is that the original design, with its limit of four partitions, was too restrictive. The extended DOS partition system allows you to have up to 24 disk partitions in a single system, without violating compatibility with older hardware and software designed based on the original four-partition limit. Of course, nobody would use that many partitions on a system in most cases, because it would be a data organization nightmare! :^)

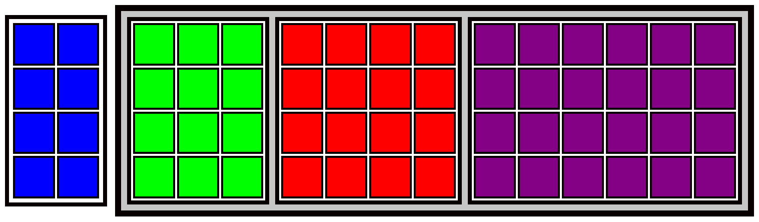
Within the extended DOS partition, the logical drives are stored in a linked structure. The extended partition's information is contained in the master partition table (since the extended partition is one of the four partitions stored in the master boot record). It contains a link to an *extended partition 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 linked 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 more clear. Let's suppose you are setting up a new system and starting with an empty 60 GB hard disk. You could create a single 60 GB partition, which would be a primary DOS partition. However, in many cases dividing up such a large disk will make it easier to manage the space, will reduce lost disk space due to slack, and will reduce defragmentation time on partitions containing more actively-used data. (See here for much more on the issues involved in partitioning.) So instead, let's say you want to split up this drive as follows:

- One 8 GB primary partition for Windows and other operating system and program files.
- One 12 GB partition for data.
- One 16 GB partition for games.
- The rest of the disk in a 24 GB partition for large multimedia files, short-term backups and "future expansion".

I'm assuming no complicating factors here, just a simple example. To do this, you will first set up a primary DOS partition 8 GB in size. This is the first of your four partitions. You will then create an extended DOS partition that is 52 GB in size. This is the second partition on the hard disk. Within the extended DOS partition you will create three logical volumes: one 12 GB, one 16 GB and one 24 GB. These are your second, third and fourth volumes (logical partitions). 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 logical partitions. Your hard disk will have one primary DOS partition, and one extended DOS partition containing three logical DOS volumes.



Graphical depiction of the partitioning of a 60 GB hard disk, as described in the example above. Each square represents 1 GB of space on the hard disk. The blue squares are the 8 GB primary partition; the green, red and purple squares are the

12 GB, 16 GB and 24 GB partitions respectively. The latter three partitions are logical partitions contained within an extended DOS partition, indicated by the larger, gray-shaded rectangle.

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.

Next: Volume Boot Sectors

Volume Boot Sectors

Each partition (also sometimes called a "logical DOS volume" in the DOS/Windows world) 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. It is also sometimes called the *volume boot record* or *partition boot sector*. 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, and number of sectors per cluster used on the partition.
- **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. The information contained in the disk parameter block is used by the operating system to determine where other internal structures of the partition are located, such as the file allocation tables.

Next: Active Partitions and Boot Managers

Active Partitions and Boot Managers

Only primary partitions can be used to boot the operating system, and of these, only the specific primary partition that is set to be bootable. DOS calls the bootable partition the *active* partition. Only one partition can be set active 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.

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; older AMI BIOSes are (in)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 early 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, but at least you understand its history now. :^)

Most people are only going to have one primary partition on their PC, because most people only use one operating system. Remember that even if you want to split your disk up into multiple FAT file system partitions, only the first will be a primary partition--the rest will be logical drives within an extended partition. However, if you are using more than one operating system--meaning ones that use different file formats, like Linux and Windows ME, not DOS and Windows ME, 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* or *boot loaders*. What a boot manager does is insert itself into the very beginning of the boot process, sometimes 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. Other methods may also be used to accomplish the same general objectives.

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. There are now a variety of different boot manager products on the market, including some that come included with utilities like Partition Magic.

Next: The DOS Boot Process

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 has 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 finds a boot sector on a device, the process of loading the operating system begins. If the operating system is DOS, or any variant of Windows that starts out by booting the equivalent of DOS--which is all of them other than Windows NT or Windows 2000--then a specific operating system load sequence commences, which is normally called the *DOS Boot Process*. In the case of Windows, additional steps are added to the end of the process after the underlying DOS operating system has loaded.

The process below outlines how booting from the hard disk functions. 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 are searched are skipped.

Here are the steps in the DOS boot process:

1. 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.
2. 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.
3. 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.
4. 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".

5. 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.
6. 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.
7. 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.
8. 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".
9. 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. First, IO.SYS is loaded and its code executed. IO.SYS will then executed MSDOS.SYS (in pure DOS systems--MSDOS.SYS is just a text file in Windows 95 and later.) Then the more complete operating system code loads and initializes the rest of the operating system structures. For MS-DOS, this means loading the command interpreter (COMMAND.COM) 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 32-bit Windows versions like Windows 95/98/ME, the steps above are only the beginning of the process. The initial DOS operating system files control the loading and execution of many more routines as the boot progresses, which perform tasks such as reading the system registry, initializing hardware devices and starting the graphical operating system shell. In fact, it is surprising in some ways just how many different pieces of code have a hand in starting up the PC.

Next: Boot Sector Viruses

Boot Sector Viruses

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

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 major classes of viruses, called *boot sector infectors*, targets the vulnerable boot areas of the hard disk. (The other major groups of viruses attack individual files, or spread using email and other Internet protocols.) 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.

Next: FAT File System Disk Volume Structures

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. I explored these high-level structures in this section. Moving down to the next level of detail, we now will explore some of the key characteristics of each disk volume's internal structure. It is at this level that we examine such issues as the organization of files and directories on the disk volume.

This section takes a look at the disk volume structures used in the FAT file system. I start with a look at the file allocation tables (FATs) used for each volume. Then, I describe the way directories are structured within each volume, and describe the differences between root and regular directories. Finally, a full exploration of file names is provided, including a discussion of conventional DOS file naming conventions, long file name implementation under VFAT and FAT32, and a description of file attributes.

Incidentally, I should mention that the volume boot sector is also part of each disk volume, but it is discussed on this page in the major disk structures section. (Originally, I had a page on it here too, but it was just duplication, so I removed it.)

Note: The descriptions in this section are geared specifically towards the characteristics of the FAT family of file systems: FAT12, FAT16, VFAT and FAT32. Other file system share some of the same characteristics, but also differ in important structural ways. For more information on NTFS, see here. For more information on other file systems, check the appropriate pages in this section, which contain links to external pages with more detail.

Next: File Allocation Tables

File Allocation Tables

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 operating systems that use FAT family file systems (including DOS and most versions of Windows.)

Data is stored in individual 512-byte sectors on the hard disk. In theory, it is possible for each file to be allocated to a number of individual sectors, and this is in fact done for some file systems (such as HPFS.) However, for performance reasons, individual sectors are not allocated to files in the FAT system. 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: a 10 GB disk partition has 20,000,000 sectors! 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. Clusters and how they work are described in full detail in this section.

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. This chaining process is described here.

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 idea behind the backup copy is that it could be used in the event that the primary becomes damaged.

In the conventional FAT system, however, the backup FAT system doesn't work too well. The problem is that the two copies are kept right next to each other on the disk, so that if, 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. Another problem is that disk utilities frequently duplicate the primary FAT to the backup FAT location. This means that any corruption that arises in the primary FAT may be duplicated to the backup copy before it is noticed.

Under FAT32, some improvements were made to the FAT backup scheme. First, either copy of the FAT can be designated the "primary" and either the "backup". Second, the method by which the FAT is copied from the primary to the backup location can be disabled. The combination of these features allows the second FAT to be protected and used in the event of problems with the first.

Next: Files, Directories, Paths and the Directory Tree

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 (of course, UNIX predates the PC by over a decade!) 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 with it. Even the file extension doesn't distinguish a file; it is just a naming convention.

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 simple files or more directories (often called subdirectories). Each directory is a container that holds files or more subdirectories (or both). The directories can be "nested" to many levels of depth. Taken as a whole, the structure of directories and subdirectories forms a logical tree. (Well, it actually looks like the root structure of a tree. The "root" and "branch" analogies are used interchangeably in the computer world, much to the confusion of some arborists. :^))

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" on the "D:" logical drive. 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 "D:\Customers\Texas\customer-list.txt". The filename is "customer-list.txt", and the path is "D:\Customers\Texas".

Note: Windows sometimes calls directories "folders". They are exactly the same thing.

Next: Internal Directory Structures

Internal Directory Structures

Every file on the hard disk 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 to the operating system. 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. (In some ways, a better analogy would be this web site itself; each index frame points to either individual pages, or another sub-index. The PC Guide is hierarchical in the same way as a PC's directory structure, and the home page would be the equivalent of the root directory.)

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 any 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 is wanted, so this date/time shouldn't be taken too seriously. 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. Under normal circumstances the date/time stamp can be useful for telling when you last modified a file, for example, but you should not count on it to "prove" anything about when someone was last using their PC, for example.
- **File Size:** The size of the file in bytes.
- **Link to Starting 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. These are named "." (single dot), which refers to the current directory, and ".." (double dot), which refers to the parent directory. 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. Note that the root directory has no parent directory, for obvious reasons. :^)

Next: Root Directory and Regular Directories

Root Directory and Regular Directories

A hierarchical directory structure is used to organize the files that exist on any hard disk volume. This "logical tree" is used on almost every file system, because of the intuitive way it arranges files, and the power it gives the user to create meaningful organization schemes for files. Much as a real tree has all its branches and roots come together in one spot, so too does a file system directory structure. The directory at the "base" of the logical tree is called, appropriately enough, the *root directory*. The root directory is special because it follows 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 confusion, and there isn't any need to have more than one anyway. In the conventional FAT file system, the root directory is fixed in place at the start of the DOS volume; it "anchors" the directory tree. The root directory located on the disk volume directly after the two copies of the FAT, which are themselves directly below the other key disk structures. This contrasts with regular directories, which can be located anywhere on the disk.

In addition to being fixed in location, the root directory is also fixed in size (under FAT12/FAT16/VFAT). 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**
360 kB 5.25" Floppy Disk	112
720 kB 3.5" Floppy Disk	112
1.2 MB 5.25" Floppy Disk	224
1.44 MB 3.5" Floppy Disk	224
2.88 MB 3.5" Floppy Disk	448
Hard Disk	512

One of the improvements introduced in the newer FAT32 version of the FAT file system was to remove these restrictions on the root directory. Under

FAT32, the root directory is treated much more like a regular directory, and can be relocated and expanded in size like any other. (It's still a good idea not to load up the root directory with too many files.)

There are a couple of other special things you should know 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).

Next: File Names and Extensions

File Names and Extensions

As virtually every PC user knows, standard (DOS-based) 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.

Since the file name is limited to eight characters and the extension to three, the conventional DOS naming scheme is sometimes called *8.3 naming*.

The file extension is occasionally the source of some confusion. Modern operating systems use them as a "file type" of sorts. They tell you--and your operating system--what kind of file you are looking at, at a glance. 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.

However, it's important to remember that there is nothing special at all about these extension names. There is nothing inherently different between an "EXE" file, an "HTM" file, and a "BAT" file--they are all "just files" to the file system, and any special meaning arising from the file extension is a function of the operating system and software, and how they interpret them. For example, I said that an EXE file is *normally* an executable file, because 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 text 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. The text editor may default to a different extension, but it won't cry foul at your choosing "TEST.EXE" as a file name.

So why are file extensions used? They are essentially a shorthand way of organizing files by type. They are used by various pieces of software, including DOS and Windows themselves, 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 the operating system 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? Probably just an error message, but you can cause errors or confuse applications by changing file extensions (which is one reason why Windows will warn you if you try to change a file extension in Windows Explorer.)

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". Graphics programs will look for "JPG" or "GIF" files, amongst others. This standard functionality is why using consistent file extensions is important.

This use of file extensions by software programs is now universal, and there are in fact hundreds of different kinds in use. Windows maintains a list of *file associations* that tell Windows Explorer which programs go with which file types (extensions). When you double-click a file in Explorer to open it, Explorer 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. :^))

Tip: You can change the program associated with any file extension by editing the file associations. Within Windows Explorer, select "Options..." from the "View" menu. Then click the "File Types" tab, select the item to be modified, and select "Edit...". Find the file type you want to modify, highlight the "open" Action, and select "Edit..." again. Change the program and close all the windows and you should be all set. Note that some programs will automatically "re-establish" associations to themselves whenever you run them! (They are supposed to ask first but software writers can be a tad, uh, possessive. :^))

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 9x/ME specifically avoid using spaces in file names when they create 8.3 aliases for long file names.

Next: Long File Names

Long File Names

Until the initial release of Windows 95, all file names using DOS or Windows 3.x (which uses the DOS file system) 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". The second name makes sense when you create it, because you know what you're saving in the file. A few months later, however, that name will probably be nearly impossible to decipher. Users didn't enjoy this limitation.

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--NTFS. 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 if they wanted compatibility to be continued.

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.

Long file name support was provided as part of the *Virtual FAT* (*VFAT*) system created for Windows 95. The VFAT file system accomplishes these goals, for the most part, 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 too 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: *+ , ; = [ ]*.

Note: The VFAT system for file names was created for Windows 95, and kept for subsequent versions of Microsoft operating systems. This includes Windows 98 and Windows ME. The FAT32 file system builds on VFAT and also uses the same scheme.

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 sometimes called a *short file name* to distinguish it from a long file name. Aliasing is performed 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 of a VFAT partition, where the total number of directory entries is limited. (FAT32 removes this restriction, but it's still good form to avoid putting lots of files in the root directory of a partition.)

In order to make sure that older versions of software don't get confused by this non-standard usage of directories, 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, because prior to the creation of long file names, such a combination of file attributes had no valid meaning. (This was a major league kludge on the part of Microsoft, but one that works!)

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 an older 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 9x/ME must be aware of long file names to work properly. Of course today, with long file names having been in common use for many years, this is not nearly the issue that it was when Windows 95 was first released.
- **"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. Another concern is that if you load a file into an older program using the alias, you can only save back to the same file name, or to another 8.3 file name. (If you save back to the same name in the same directory, the original long file name will be retained.)
- **Problems with Conflicting Alias File Names:** There are two significant problems with the long file name aliasing scheme:
    - **Alias Changes:** 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.
    - **Overwriting:** 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 Realiasing:** 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. For example, many registry entries refer to the short file name alias, not the long file name. For a full discussion of this problem, please read the PC Guide article Xcopy Xposed. 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.

Next: File Attributes

File Attributes

Each file is stored in a directory, and uses a directory entry that describes its characteristics such as its name and size. The directory entry 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 the operating system 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 programmer that knows what it is doing can override the attributes of a file, and certainly, the writers of viruses do this as a matter of course!

That said, most 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

The attribute bits are summed to form the attribute byte. So, the attribute byte for a hidden, read-only directory would be 00010011, which is 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 apply 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. Of course in practice, trying to do this would result in a mess--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. Still, you should

not rely on this mechanism absolutely to ensure that your critical files are backed up.

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.

Next: Clusters and File Allocation

Clusters and File Allocation

As I have been explaining in various sections in this discussion of hard disk structures and file systems, the purpose of the file system is to organize data. The most popular file system in the PC world is the FAT family of file systems: FAT12, FAT16, VFAT and FAT32. All of these file systems use a specific technique for dividing the storage on a disk volume into discrete chunks, to balance the needs of efficient disk use and performance. These chunks are called *clusters*. The process by which files are assigned to clusters is called *allocation*, so clusters are also sometimes called *allocation units*.

In this section I describe how FAT file system clusters and file allocation work. I start by describing what clusters are themselves. I then talk about the mechanism by which clusters are assigned to files. I explain how the FAT file system handles deleting files--and undeleting them as well. I then discuss how the cluster system can cause the file system to become fragmented. I conclude with a discussion of file system errors that can occur under the FAT file system.

Next: Clusters (Allocation Units)

Clusters (Allocation Units)

As described here, the smallest unit of space on the hard disk that any software can access is the *sector*, which normally 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. The file system HPFS uses this type of arrangement.

Under the FAT file systems (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 20 GB disk volume using 512 byte sectors managed individually would contain over 40 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). In some situations 128-sector clusters may be used (65,536 bytes per cluster). 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 third-party partitioning utilities can alter the cluster size of an existing partition (within limits) but aside from using these, 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 an important consideration in making sure you maximize the efficient use of the disk--larger cluster sizes result in more wasted space because files are less likely to fill up an integer number of clusters. This issue is given a full treatment here.

Next: File Chaining and FAT Cluster Allocation

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 the 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; for the FAT16 file system this may be 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.HTM" 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 launch 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 know 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.

Next: File Deletion and Undeletion

File Deletion and Undeletion

As you use any PC, you will routinely create and delete files. Now, deleting a file means to erase it from the disk, which you would think means the file is destroyed. Naturally, you would only do this to files you no longer need.

However, in many circumstances, it is useful to be able to "undo" the results of deleting a file. Accidental deletions happen far more often than you might imagine, believe me. :^)

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. Instead, the system 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 "freed" clusters will eventually be reused by other files, as they request more space for storage. However, if you accidentally delete a file you can very often recover it if you act quickly. In DOS you can use the *UNDELETE* command. There are also third-party tools that will undelete files, such as Norton Utilities' UNERASE. If you run one of these tools immediately, it can identify and recover the deleted files in a directory. You will have to provide the software with the missing first character of the file name (which was overwritten by the E5h code in that file's directory entry when the file was deleted).

The less work 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. If you delete a file on a system that is fairly full, and then start making many new files, some of the clusters formerly used by the deleted file may be reused, and their former contents lost. Obviously, if you defragment your disk or do some other large-scale disk work, you will most likely lose the contents of deleted files forever.

Many operating systems have made deletion and undeletion less of an issue by integrating protection for erased files into the operating system itself. Newer Windows versions send all deleted files initially to a "Recycle Bin", from which they can be restored if needed. These deleted files stay around for a while, in case you want to undelete them, and if they are in the Recycle Bin they can be restored to their former locations with no data loss. However, the size of the Recycle Bin is limited and eventually files will be permanently removed from it.

Warning: If you value your files, you will not rely too much on the capabilities of utilities that restore deleted files. They are no substitute at all for proper backup procedures.

Next: Fragmentation and Defragmentation

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 disk 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 most versions of Windows 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 most 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:

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

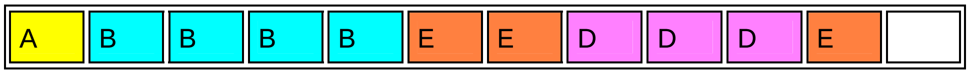
A	B	B	B	B	C	C	D	D	D		

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

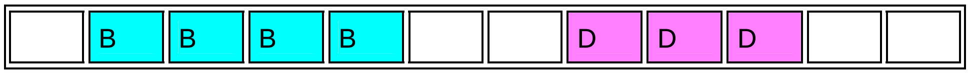
A	B	B	B	B			D	D	D		

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

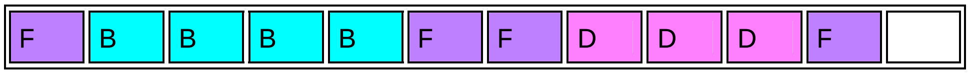
E into two fragments, using part of the space formerly occupied by C. Here's what the "disk" looks like now:



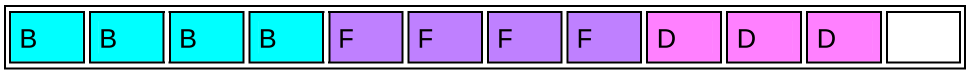
Next, we delete files A and E, leaving the following:



Finally, we create file F, which takes up 4 clusters. The disk now looks like this:



As you can see, file F ends up being broken into three fragments. This is a highly simplified example of course, but it gives you the general idea of what happens. Since real disks have thousands of files and thousands of clusters, the fragmentation problem with real disks is much greater than what I have illustrated here. 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:



Note that the exact order of the files after defragmentation is not guaranteed; in fact, a classical defragmenter won't pay much attention to the order in which files are placed on the disk after defragmenting. Some newer, more capable defragmenters include special technology that will put more frequently-used files near the front of the disk, where performance is slightly faster. They can also move the operating system swap file to the front of the disk for a slight speed increase.

Next: FAT File System Errors

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, and so on, which are discussed in some detail here. 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 in the System Care Guide that every hard disk volume be scanned for routine file system problems on a daily basis, if possible.

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 newer versions of Windows automatically scan the disk for errors when they starts, if they detect 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 user has come across this problem from time to time. Lost clusters are simply ones 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 scan an entire disk volume for lost clusters using the following procedure (or something similar to it):
    1. Create a copy in memory of the FAT, noting all of the clusters marked as in use.
    2. 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.
    3. 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*.

  Lost clusters are usually the result of interrupted file activity of some sort--a program will allocate some clusters to a file it is building, and if the file is not properly finished and closed, the clusters never get correctly linked to a file name. The program that detects lost clusters 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 an artificial file name and links the lost clusters to that name, so that a real file is formed. Usually this file will then be damaged in some way, but you can often at least see what this orphaned data was and in some cases, recover at least part of 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. Alternately, one of the clusters in the middle of two or more cluster chains may point to the same place. Obviously this is a problem: the same cluster cannot be part of more than one file, at least not in *this* universe! :^) Each time you use either of the cross-linked files, you will overwrite all or part of 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 at least one of them.) Often, both files will be lost and you will need to restore them from a backup.

- **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.

Next: Partitioning, Partition Sizes and Drive Lettering

Partitioning, Partition Sizes and Drive Lettering

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 a hard disk, even if only "partitioning" it into a single volume, 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 the matter of "efficiency" has several different facets, depending on one's priorities. Even though you can fit an entire disk into one partition (with proper operating system support), in many cases you will not want to do this, for a variety of reasons that we will explore in this section. The pages that follow examine all of the issues involved in selecting partition types and deciding how to organize a hard disk into volumes under the FAT family of file systems. This includes a look at the relative merits of the FAT16/VFAT file systems and the newer FAT32 file system, and a discussion of related issues such as partition conversion. I also describe the somewhat tricky mechanism by which partitions are assigned drive letters.

Note: I should point out that my emphasis in the discussions in this rather large section has changed since I first wrote it back in 1997. At that time hard disks were of a decent size, but not tremendously large. Most people were still using the FAT16 file system and relatively slow CPUs, so there were significant issues with performance and disk storage efficiency--many users had problems with big chunks of their disk space being used up by slack. As I rewrite this material in 2001, hard disks have increased in size by 10 times or more, and both hard disks and other components are much faster as well.

Thus, I no longer believe it is worthwhile, for most people, to invest many hours on tweaking the file system for small performance or space gains. I continue to note, where appropriate, the most optimal ways of doing things, but I also point out where spending a great deal of effort may not yield acceptable returns. As with other areas of the site, I continue to bear in mind that many people aren't using the latest hardware. For this reason, optimization tricks that may be of significant value for older machines have been maintained, and I do not just assume that everyone is running the latest hardware.

Next: FAT Sizes: FAT12, FAT16 and FAT32

FAT Sizes: FAT12, FAT16 and FAT32

Throughout my discussion of file systems, I have referred to the FAT *family* of file systems. This includes several different FAT-related file systems, as described here. 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 the maximize size of the table. 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 very small volumes, and is used on floppy disks and hard disk partitions smaller than about 16 MB (the latter being rare today.)
- **FAT16:** The FAT used for most older systems, and for small partitions on modern systems, 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, even with FAT32 now more popular than FAT16. 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. VFAT is a variant of FAT16.
- **FAT32:** The newest FAT type, FAT32 is supported by newer versions of Windows, including Windows 95's OEM SR2 release, as well as Windows 98, Windows ME and Windows 2000. 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 very small hard disk volumes	Small to moderate-sized hard disk volumes	Medium-sized 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

Tip: If you are not sure of whether a given disk volume on your PC is formatted using FAT16 or FAT32, you can use this procedure to find out.

Next: FAT Partition Efficiency: Slack

FAT Partition Efficiency: Slack

One issue related to the FAT file system that has gained a lot more attention over the years is the concept of *slack*, which is the colloquial term used to refer to wasted space due to the use of clusters for storing files. This began in the mid-1990s when larger and larger hard disks began shipping with most systems. Typically, retail systems were not being divided into multiple partitions, and users began noticing that large quantities of their hard disk seem to "disappear". In many cases this amounted to hundreds of megabytes on a disk of only 1 to 2 GB in size. When the use of FAT32 became more common this problem was less of an issue for a while. Today, with hard disks sized at 40 GB or more commonplace, even FAT32 has problems with slack.

Of course the space doesn't really "disappear", assuming we are not talking about lost clusters, which 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 during that expansion. 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. (When collecting rain water, 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 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 partitions to waste up to 40% of their space due to slack, although 25-30% is more common.

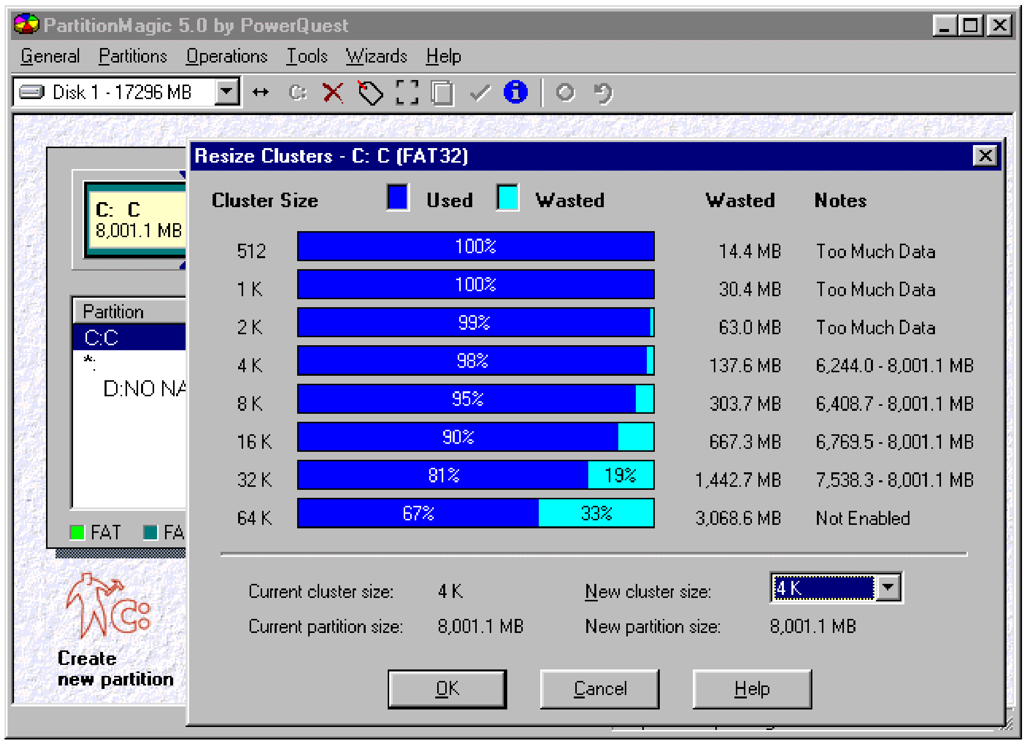
Let's take an example to illustrate the situation. Let's consider a hard disk volume that is using 32 kiB clusters. There are 17,000 files in the partition. If we assume that each file has half a cluster of slack, then this means that we are wasting 16 kiB of space per file. Multiply that by 17,000 files, 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 kiB 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,	Sample Slack Space,

	50% Cluster Slack Per File	67% Cluster Slack Per File
2 kiB	17 MB	22 MB
4 kiB	33 MB	44 MB
8 kiB	66 MB	89 MB
16 kiB	133 MB	177 MB
32 kiB	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, sometimes 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. One way that cluster sizes can be reduced is to use FAT32 instead of FAT16, as described in other pages in this section. However, on very large modern hard disks, big partitions even in FAT32 use rather hefty cluster sizes!


PartitionMagic 5.0 by PowerQuest
General Partitions Operations Tools Wizards Help
Disk 1 - 17296 MB
C: C
8,001.1 MB
Partition
C:C
*:
D:NO NA
FAT FA
Create new partition

Resize Clusters - C: C (FAT32)

Cluster Size	■ Used	■ Wasted	Wasted	Notes
512	100%		14.4 MB	Too Much Data
1 K	100%		30.4 MB	Too Much Data
2 K	99%		63.0 MB	Too Much Data
4 K	98%		137.6 MB	6,244.0 - 8,001.1 MB
8 K	95%		303.7 MB	6,408.7 - 8,001.1 MB
16 K	90%		667.3 MB	6,769.5 - 8,001.1 MB
32 K	81%	19%	1,442.7 MB	7,538.3 - 8,001.1 MB
64 K	67%	33%	3,068.6 MB	Not Enabled

Current cluster size: 4 K   New cluster size: 4 K
Current partition size: 8,001.1 MB   New partition size: 8,001.1 MB

Slack analysis of the C: drive from one of my PCs, as displayed by Partition Magic 5.0. Partition Magic is probably the most popular third-party partitioning tool. It allows you to change the cluster size of the partitions on your disk. This analysis tool shows you how much slack would be wasted for various cluster sizes on a given disk volume, to let you decide what cluster size you would like to use. It also shows the minimum and maximum partition sizes that are allowed for the cluster size and file system under consideration.

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 kiB or 8 kiB partitions to be acceptable; most consider the slack of 32 kiB cluster size partitions excessive; and the 16 kiB partitions seem to go both ways. It depends entirely on your needs, and how critical your disk space is. With today's large disks, many people don't care as much about slack as they used to; a typical PC user bringing home a new machine with a 30 GB hard disk isn't going to get it half full even after quite a while. For others, slack is still very important. I personally only avoid the 32 kiB partitions when possible, 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 1 cent, this means that the "cost" of this problem is less than $5. 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 $50 on utility software to avoid it may not be too smart either, considering that for not much more than that you can get a second hard disk with dozens of gigabytes! 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.

Next: Relationship of Partition Size and Cluster Size

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 very 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. The place where partition size and cluster size begin to interact is in the use of FAT16, the standard FAT type used for hard disk partitions between 16 MB and 512 MB in size, and used for drives up to 2,048 when FAT32 is not supported. 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 (a similar table for FAT32 can be found here). 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 64 kiB (65,536), the maximum partition sizes fall short of the "round" MB numbers that everyone always uses. Thus the maximum partition size for 8 kiB clusters is 511.92 MiB, not 512 MiB:

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

Note: Windows NT supports a 64 kiB 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 kiB cluster partitions aren't supported by Windows 9x/ME or other FAT-using operating systems, this isn't a popular option. I would recommend using NTFS under NT for a partition this large, or FAT32 under Windows 2000.

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 kiB 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 kiB for the cluster size and not 16 kiB or 32 kiB. But that doesn't mean you can't have a 300 MB partition that uses 32 kiB 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).

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 kiB clusters. Of course, if your operating system supports FAT32, you should just use that.

Next: FAT32 Performance Tradeoff: FAT32 Cluster Sizes and FAT Sizes

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. Now that huge hard disks with dozens of gigabytes have made FAT32 essential for newer systems, you often don't have a practical choice between FAT16 and FAT32 any more. However, the principles in this page are still relevant, depending on how you are setting up your hard disk. They can also help influence your decisions regarding how to partition very large drives under FAT32.

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 kiB 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 kiB to 4 kiB. 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. It is usually the right thing to do in this situation.

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 kiB	4 kiB
Number of FAT Entries	65,526	524,208
Size of FAT	~ 128 kiB	~ 2 MiB

Still worse, if we increase the size of the FAT32 volume from 2 GiB in size to 8 GiB, the size of the FAT increases from around 2 MiB to a rather hefty 8 MiB. 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 kiB 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 MiB 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 kiB cluster size is technically able to support a disk volume 1 TiB (1,024 GiB) in size. The *little* problem here is that the FAT size would be astronomical--over 1 GB! (268 million times 4 bytes per entry).

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

Cluster Size	**"Minimum" Partition Size (GiB)**	**"Maximum" Partition Size (GiB)**
4 kiB	0.5	8
8 kiB	8	16
16 kiB	16	32
32 kiB	32	!?

Note: I am not really sure what the maximum partition size is for a FAT32 partition. :^) I have heard various different numbers, but nobody seems to be able to produce an authoritative answer. "Officially" it is 2,048 GiB (2 TiB), but there are likely to be other limiting factors...

As you can see, despite the claims that FAT32 would solve large cluster problems for a long time, it really hasn't. As soon as you hit 32 GiB partitions, you are back to the dreaded 32 kiB clusters we all knew and hated in FAT16. Obviously 32 GiB is a lot better than having this happen at 1 GiB, of course, but still, FAT32 is more of a temporary work-around than a permanent solution. When FAT32 was first introduced, many people said "Yeah, but 32 GiB is *huge*. I'll probably never have a disk that large and if I do, I won't care about a little wasted disk space!" In fact, it took less than *five years* for hard disk makers to move from selling 1 to 2 GB hard disks, to selling ones 32 GB in size or more! And those same people are finding they *do* care about the wasted disk space, though perhaps less than they did when they only had 1 GB. :^)

The table below is a little exercise I did for fun, to show the size of the FAT (in MiB) as the partition size increases, for various cluster sizes. You can see why FAT32 doesn't stick with 4 kiB clusters for very long--if it did, you'd need enormous amounts of memory just to hold the table. (A 60 GB partition, if formatted with 4 kiB clusters, would result in a FAT that is 64 MiB in size, which is about as much as the entire system memory in many newer PCs.) 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 MiB through partitions of size 64 GiB:

Partition Size	**4 kiB Clusters**	**8 kiB Clusters**	**16 kiB Clusters**	**32 kiB Clusters**
8 GiB	**8 MiB**	4 MiB	2 MiB	1 MiB
16 GiB	16 MiB	**8 MiB**	4 MiB	2 MiB
32 GiB	32 MiB	16 MiB	**8 MiB**	4 MiB
64 GiB	64 MiB	32 MiB	16 MiB	**8 MiB**
2 TiB (2,048 GiB)	--	*1,024 MiB*	*512 MiB*	*256 MiB*

The last entry, 2 terabytes, is for fun, to show how laughable I find it when people go on about 2 TiB hard disks "being supported by FAT32". Technically they are, I guess, if you want to deal with a FAT that is 256 MiB 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. We also better hope that no other little "surprises" show up as disks get larger: several did pop up when the 64 GiB barrier was reached, such as difficulties with the FDISK program.

Really, what this page shows is that 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. I should make clear that I am not recommending against the use of FAT32 on Windows 9x/ME systems. With modern drives of 50 to 100 GB, FAT16 is just too impractical, with its 2 GiB partition limit. At the same time, I want to make sure that people realize that FAT32 has its own limitations. It is really more of a kludge than a permanent solution to the problems of large partitions and cluster-based file allocation.

Next: Using Partitioning with Hard Disks Over 2 GB

Using Partitioning with Hard Disks Over 2 GB

While partitioning can be somewhat complicated, there is one aspect to it 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.

This was a big issue for the users of the first Windows 95 version, which did not have FAT32. In some cases a system would ship with say, a 6 GB hard disk, and it would be necessary to split it into at least three partitions due to FAT32's 2 GB limit. Even in that case, you would end up with three partitions with 32 kiB clusters, creating a lot of waste due to slack. To avoid this, many people would segment a 6 GB disk into six or even more partitions, but this introduces other issues, as described here.

With the introduction and widespread adoption of FAT32 in newer operating systems, the problems with hard disks over 2 GB have been rendered largely moot. If you are running Windows 95 OSR2, Windows 98 or Windows ME, you should use FAT32, which will let you use the full size of a large hard disk in a single partition. Of course, even with FAT32 you may want to use partitioning to reduce lost space due to slack, as described on the next page. However, the need is much reduced compared to using FAT16.

Next: Using Partitioning to Reduce Slack

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 storage efficiency 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 both to FAT16 and FAT32 partitions--on older systems that use FAT16 partitions for volumes over 512 MiB, cluster sizes will be 16 kiB or 32 kiB, and slack will be significant, and the same goes for FAT32 partitions that are 16 GiB or more.

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:

FAT16 Cluster Size	Size of Each Partition	Number of Partitions	Typical Total Slack (All Partitions)	Typical Total Slack (% of Disk Space)
2 kiB	128 MiB	16	28 MiB	1.4%
4 kiB	256 MiB	8	56 MiB	2.8%
8 kiB	512 MiB	4	112 MiB	5.6%
16 kiB	1 GiB	2	225 MiB	11.2%

32 kiB	2 GiB	1	450 MiB	22.5%

As you can see, putting a 2 GB disk in a single partition is inefficient; 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 dividing the disk further. In fact, the best solution might seem to be just going to 128 MiB 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. :^))

With a larger disk and FAT32, the example is not that much different, but the slack depends entirely on how many more files you put on that larger disk. If you put the same number of files on the larger disk using FAT32, slack (as a percentage of disk space) decreases dramatically; if you put many more files on the larger partitions then you "give back" some of the slack savings, though of course you are still ahead of where you would have been using FAT16. Which of these is most appropriate depends on how you are using your system. In many cases, the large hard disks available today are used to hold big files, such as video and audio, that were rarely seen on older PCs. In other applications, a bigger disk just means many more small files.

Let's look at an example where have, say, a mythical 64 GiB (68.7 GB) hard disk and 96,000 files on it. Here, I am looking at a disk 32 times the size of the previous example, but have only increased the number of files by a factor of four. This means slack, as a percentage, will be lower even for partitions of the same cluster size. Here's how this looks under FAT32, assuming the same 60% end cluster waste:

FAT32 Cluster Size	Size of Each Partition	Number of Partitions	Typical Total Slack (All Partitions)	Typical Total Slack (% of Disk Space)
4 kiB	8 GiB	8	225 MiB	0.35%
8 kiB	16 GiB	4	450 MB	0.7%
16 kiB	32 GiB	2	900 MB	1.4%
32 kiB	64 GiB	1	1,800 MB	2.8%

As you can see, the total amount of slack, in bytes, is higher, because we have more files. However, the percentage of total disk space used up in slack is much lower because the disk is so much bigger--that's the advantage of FAT32. As before, when you increase the cluster size, you end up with bigger partitions, and more slack. 32 kiB clusters means four times as much slack as 8 kiB clusters. However, with the total slack still so low--2.8%--and with the huge size of the disk being contemplated (64 GB) the entire matter is of arguable importance. On a 2 GB disk, 450 MB is a big deal; on a 64 GB disk, 1.8 GB really is not, at least to most people. While most people wouldn't put an entire 64 GB hard disk in one partition (there are other reasons to avoid doing this, not just slack), it just isn't the big deal it used to be.

The examples above show 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 little bits; on an older (FAT16) system I usually use 8 kiB or 16 kiB 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.

On a FAT32 system with a large hard disk, I usually go with partitions no more than 16 GiB, sticking to 16 kiB clusters. The difference between 8 kiB and 16 kiB clusters is not significant enough to warrant all the volumes needed to divide a very large disk into 8 GiB units, in my estimation. On some disks, such as backup volumes or ones where I will be storing large multimedia files, I will use 32 kiB clusters and very large volumes. This is an example of tailoring one's partition size and cluster size to the type of data being stored on the partition. If the files being put on the volume are very big, the cluster size becomes essentially irrelevant.

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

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. I discussed this in some detail here. Using partitioning to reduce slack is pretty much a necessity if your operating system does not support FAT32, and with very large hard disks it is also commonly done even with FAT32, 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 F:? Or was it I:?) 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. This situation is far more common when chopping up partitions under FAT16, because if you have a hard disk that needs partitioning for efficiency under FAT32, it's pretty large. But at the same time, FAT32 systems tend to have much larger files overall too.

Let's suppose that our comfort factor for free space at the end of a volume is 20 MiB. (For me, this is way too low. I get nervous if a regular volume ever gets below about 50 MiB, and now that I am using much bigger disks and files I really like to have 500 MiB free if at all possible!) Now, let's re-examine the same 2 GiB disk 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 20 MiB 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 kiB	128 MiB	16	28 MiB	320 MiB	336 MiB
4 kiB	256 MiB	8	56 MiB	160 MiB	216 MiB
8 kiB	512 MiB	4	112 MiB	80 MiB	192 MiB
16 kiB	1 GiB	2	225 MiB	40 MiB	265 MiB
32 kiB	2 GiB	1	450 MiB	20 MiB	470 MiB

The answer to the question of efficiency isn't so clear-cut now, is it? I mean, the 32 kiB cluster size option is still ridiculous, but really, so is the 2 kiB cluster size option. Even if you use 8 partitions with 4 kiB clusters, you'll be wasting a lot of space at the end of each volume. In a FAT32 system with a large hard disk you have the same general issue, but it's less of a concern, because each partition is so much larger. If you have a 64 GiB hard disk and chop it into eight 8 GiB partitions then you will need to "save space" at the end of each partition, but each partition is pretty big. Then again, if you are using these partitions to store much larger files than you did under FAT16 (which often happens) then you are back in the same place again anyway.

The matter of "end of volume" space 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.

Next: Do More Partitions Keep the Disk "Organized"?

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 different 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, at least in my opinion. :^)

The reason 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 for "images", 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 20 GB disk in eight 2.5 GB 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. :^) Sure, you can alleviate this to some extent by resizing partitions. But since hyper-partitioning isn't really buying you anything anyway, why bother in the first place?

So overall, from an organizational and flexibility standpoint, I think you are generally better off with a single large partition, or at most two or three if the hard disk is really large. I think the only reason to use multiple partitions at all is for performance reasons, including some of the other issues I discuss on the next page. 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.

Next: Special-Purpose Partitions and Other Partitioning Issues

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 issues you may want to take into account when considering how to partition your disks, aside from the frequently-discussed matter of slack:

- **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. Then again, this is really an optimization, and of less importance in newer systems than older ones.
- **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, or at the very least, not worry about making the partition small in order to avoid large cluster sizes. There is less overhead when using larger clusters--doing a sequential read of a 10 MB file on a volume that uses 32 kiB clusters means 319 "next cluster" lookups in the FAT. Reading this entire file on a volume with 2 kiB clusters increases this to 5,119 lookups. Another issue 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. Frequent defragmentation of a disk with smaller clusters will mitigate this effect, but using larger clusters is easier.

- **File System Utility Performance:** With modern large hard disks, enormous partitions are possible under FAT32. Partitions with tens of gigabytes and hundreds of thousands of files can take a very long time to process by disk scanning utilities, defragmenters and the like. Using smaller partitions cuts down on the time required to perform these tasks. If you have some data that changes frequently, and other data that is mostly static, separating them onto different partitions may make sense, since you can defragment the static files less frequently.

The points above mean that the ideal place in many cases for the swap file under Windows, for example, is in a dedicated partition at the start of the second disk in a two-disk system, and this is what I have set up myself in the past. On one of my PCs, I had two disks of approximately the same size and speed, and the swap file was arranged to take up the entire contents of the first partition of my second hard disk, about 63 MiB. I used Partition Magic to set this partition's cluster size to 32 kiB, even though a partition of this size would normally only use 2 kiB clusters. See here for more on swap file optimization.

Today, I don't generally bother with such minor optimizations. All PC hardware is much faster than it was, and I don't really see much need to optimize to this degree. On a system with a single hard disk and a modern operating system, you may be better off to just leave the swap file on the C: drive, especially if you defragment regularly. Modern defragmenters will optimize the swap file in a special way, moving it to the start of the partition where transfer performance is the best. Of course, you can still use a dedicated swap file partition if you want to, and some people still like to do this.

Next: Partition Type Conversion

Partition Type Conversion

Under certain circumstances, you may want--or even need--to change a partition from one file system type to another. The most common type of partition conversion is from FAT16 to FAT32, as PC users upgrade to newer operating systems like Windows 98 or Windows ME. I discuss conversion between FAT partitions here, on this page. In some situations you may also need to convert from FAT to NTFS, which I cover in the NTFS section. Conversions between FAT and other types of partitions are also possible, but are usually handled using special utilities or tools within the operating system used by the target file system.

There are two usual reasons for converting a partition from FAT16 to FAT32. The first, and most common, is to take advantage of the slack reduction and other features associated with FAT32. For example, converting a 2 GiB FAT16 partition to FAT32 will result in cluster size being reduced from 32 kiB to 4 kiB, and most of the slack on the partition being eliminated. The other reason is to allow a partition to be *expanded* in size after upgrading to a newer hard disk. If you have a 2 GB FAT16 and copy it to an 8 GB drive in a system that supports FAT32, you can resize the partition to 8 GB using a partition management program, but the partition must be converted to FAT32 first, since FAT16's maximum partition size is 2 GB.

To convert a partition from FAT16 to FAT32, you can either use a third-party partition management utility (which also provides many other partition-related features as described here) or the built-in facilities provided within certain versions of Windows. The Windows partition utility is "free" (not really free, but you don't pay extra for it) and accessed from the Start menu by going to Programs, then selecting Accessories and then System Tools. You'll see an entry for "Drive Converter". The process may take some time, and is non-destructive, meaning that your data will not be lost in the process. (Be sure to read the warning below, however!)

Note: The Drive Converter utility is *not* provided with Windows 95 OSR2, even though that operating system supports FAT32. The stated reason is that OEM SR2 is supposed to be only for OEMs and new hardware, and OSR2 wasn't intended for upgrading existing systems. Of course, millions of people upgraded to OSR2 anyway. In that case, a partition management program would be necessary to do the partition conversion.

The drawback to the Microsoft Drive Converter is that it has significant limitations associated with it. First, it is a one-way road: you can convert from FAT16 to FAT32, but not back again. Furthermore, doing this removes the ability to "uninstall" Windows 98/ME if it was an upgrade install. Neither of these are usually an issue for most users, but if you require the ability to go from FAT32 to FAT16, you can again use one of the partition management programs I just mentioned. If you are planning on any advanced partition work, such as resizing or moving the partition after converting it, you'll need one of these tools anyway.

Warning: All partition modification activities carry with them some risk of data loss, especially if power is interrupted during the process. Always back up your important data before doing a partition conversion. Avoid doing conversions if you are experiencing power instabilities, or if violent weather is in the vicinity. If at all possible, use an uninterruptible power supply when undertaking this sort of task.

Next: Drive Letter Assignment and Choosing Primary vs. Logical Partitions

Drive Letter Assignment and Choosing Primary vs. Logical Partitions

Most people know that disk volumes using the DOS or Windows operating systems are accessed using drive letters, such as C:, D: and so on. However, it can be a bit confusing trying to figure 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 a 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 the operating system assigns drive letters using a specific sequence when the PC is booted. Drive letters are not permanently assigned to the drive, so adding new hard disk volumes can interrupt the previous order the next time the machine is started.

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--some other removable drives may be able to use it, but it cannot be assigned to hard disk volumes.

Hard disks are lettered starting from C:, with each partition getting a separate letter. It is essential to realize that the system will first assign a letter to all primary partitions on all hard disks in the system, before it will assign any letters to any logical volumes (in extended 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. (To understand the difference between primary partitions and logical volumes, see here).

Let's take an example system that contains a single 1.2 GB hard disk broken into three partitions. The first partition is the 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 applications record what disk and directory they are in and expect that they will remain stationary (which stinks, but hey, that's life.) The rise in importance of the Windows registry has made this situation even worse, since programs routinely record pointers to files, including drive letters, in the registry.

There is one relatively simple way to avoid having this letter-shifting 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 may 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.

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: it can convert a primary partition into a logical volume. 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-ROM drives, DVD units, 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 their assignments can be changed by using different parameters for the driver software. For example, the drive letter for a CD-ROM drive under Windows can be set in software using the Device Manager. You cannot do this for hard disks under Windows 9x/ME.

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.

Next: Disk Partitioning and Formatting Programs

Disk Partitioning and Formatting Programs

Every operating system ships with a set of utilities for partitioning and formatting all types of disks, including hard disks, floppy disks and removable media as well. The basic tools used for operating systems that use the FAT family of file systems have not changed much in the last decade. If you are using DOS, Windows 3.x or Windows 9x/ME, you will be using pretty much the same set of utilities. These programs allow you to set up partitions and make them ready for use, though their capabilities are generally rather limited.

Over the last several years, many third-party tools (meaning, applications not part of DOS or Windows and not written by Microsoft) have appeared on the market. These programs expand upon the restricted functionality in the standard DOS/Windows and allow for powerful manipulation of hard disk partitions. They are essential for those of us who frequently upgrade or troubleshoot PCs. Once you have one, you'll wonder how you ever did without it (though that is true of so *many* things in the PC world, isn't it? :^))

This section takes a look at both built-in "standard" utilities, and third-party tools, which are related to partitioning and formatting hard disks and other media. This includes a look at the FDISK, FORMAT and SYS commands built into DOS/Windows, and a discussion of third party tools that create, delete, rearrange and convert partitions, copy partitions, or create disk images for copying or backup purposes.

Note: You may also want to refer to this procedure for step-by-step instructions for partitioning and formatting an empty hard disk.

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 a very good idea; they are so cheap today that any serious PC user should strongly consider running with a UPS all the time anyway. If you don't have one, try to borrow one while doing partitioning work (which is not something you're likely to do very often.). If you can't, then at the very least, don't do the obviously silly (say, trying to resize all of your hard disk's partitions during a big thunderstorm... :^))

Next: FDISK

FDISK

Every operating system comes with an appropriate utility for partitioning hard disks. The program used on most PCs is the one supplied with DOS and consumer versions of Windows (Windows 9x/ME). It is called *FDISK*, which stands for "fixed disk", an older term for hard disk. FDISK is used only for partitioning FAT family file systems (FAT12/FAT16/VFAT/FAT32), and allows you to perform the following functions:

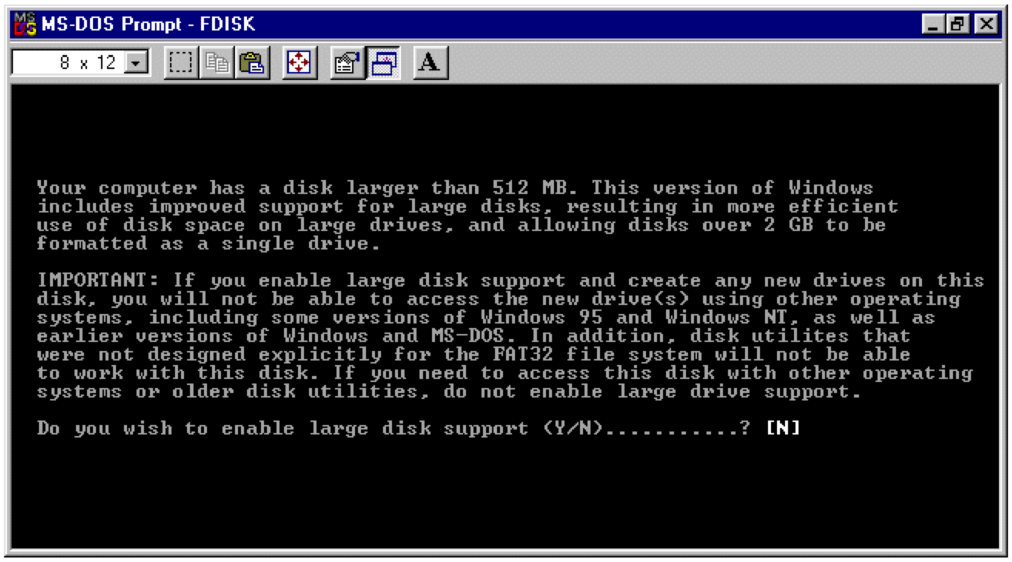
- **Create Partitions:** FDISK allows you to create a primary partition or logical volumes. To create a logical volume you must of course first create an extended DOS partition, since the logicals are contained within the extended partition.
- **Set Active Partition:** You can use FDISK to set the primary 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 enabled 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... This is one reason why third-party partitioning programs have been so successful.
- **Display Partition Information:** The last basic 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 a DOS command line or Windows DOS box. 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 9x allows you to run FDISK direct from the graphical user interface, and even use it 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, as mentioned above).
- **FAT32 Support:** The version of FDISK that comes with newer versions of Windows supports the creation of partitions that use the

FAT32 enhanced file system for larger volumes. Some genius at Microsoft, however, decided not to call it FAT32 within this program. Instead, when you run FDISK on a system that has FAT32 support, 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. If you accidentally hit "N" or don't understand the question, FAT32 will be disabled (which routinely causes confusion on the part of many newer PC users...)

Tip: 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.



```
Your computer has a disk larger than 512 MB. This version of Windows
includes improved support for large disks, resulting in more efficient
use of disk space on large drives, and allowing disks over 2 GB to be
formatted as a single drive.

IMPORTANT: If you enable large disk support and create any new drives on this
disk, you will not be able to access the new drive(s) using other operating
systems, including some versions of Windows 95 and Windows NT, as well as
earlier versions of Windows and MS-DOS. In addition, disk utilites that
were not designed explicitly for the FAT32 file system will not be able
to work with this disk. If you need to access this disk with other operating
systems or older disk utilities, do not enable large drive support.

Do you wish to enable large disk support (Y/N)...........? [N]
```

Introductory page to the Windows 9x FDISK program, displayed when FAT32 is supported. Be sure to change the "N" to "Y" before proceeding!

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. FDISK's primary advantage is, of course, that it is free (well, built-in anyway).

There is one other option for FDISK, which is undocumented--Microsoft doesn't tell you about it, and it doesn't even show up if you type "FDISK /?". This is the "/MBR" option. If you run "FDISK /MBR", FDISK will rewrite the code in the master boot record (MBR), while leaving the partitions intact. This can be useful for eliminating some types of viruses that infect the master boot record. However (and there's always a however, isn't there?) it can also cause problems in some situations. For example, some viruses *encrypt* certain disk structures, and if you run FDISK /MBR you may have a more difficult time recovering from the infection. As always, backups are prudent (but don't overwrite ones created prior to the virus in such an instance!)

Warning: Be careful before using the FDISK /MBR command. It is a good idea to do this only if it is specifically recommended for fixing a particular virus or other problem.

Finally, note that Windows NT and Windows 2000 don't use FDISK. They make use of 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. See the section on NTFS for more details.

Next: FORMAT

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. As described here, 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. The reason for this difference is that for hard disks, partitioning must be done in between the two steps. Modern hard disks are low-level formatted at the factory; they are then partitioned and high-level formatted by the user (or system builder).

There are many different parameters that the FORMAT command can use; these 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 ready for booting. 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. In most cases today, you do not need to use any of the optional flags associated with the FORMAT command; it will "figure out" what it needs to do by examining your hardware. Even the "/S" flag is not needed for newer operating systems, which take care of all the operating system file setup for you.

There's one FORMAT command parameter that bears special mention. This is the undocumented "/Z" switch, which allows you to specify a particular cluster size for a partition as you format it, overriding the defaults that are established by the system based on the size of the partition (see here for a table of standard cluster sizes for FAT16 based on partition size, and here for FAT32.) For example, if you have a 12 GiB partition, the normal cluster size would be 16 sectors, or 8 kiB. If you format such a partition with the command "FORMAT /Z:8" then it will format it with eight-sector, 4 kiB clusters instead. While this can be useful in some circumstances, be sure to read the warning below carefully before using this option: it is undocumented for a reason.

Warning: While the "/Z" switch has become increasingly popular amongst those that like to tweak their systems, I do *not* recommend the use of this switch. In theory, it should allow you to create a large partition with small cluster sizes, or a small partition with large clusters. The problem is that this switch creates *non-standard* partitions that can cause problems with some software that isn't expecting them. In the example given above, the file allocation table would have twice as many entries as normal for a FAT32 partition. There have been problems reported with programs "breaking" when attempting to use partitions that have been modified using the "/Z" parameter to make small clusters, because the programs can't handle the increased numbers of clusters in the partition. If you really care about getting smaller cluster sizes that much, just break the disk into more partitions. Using "/Z" to *increase* the cluster size should work, but again, it's a minor performance tweak and should be approached with caution.

Next: SYS

SYS

The operating system files that allow a hard disk to be booted are normally 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, essentially, the "/S" parameter of the FORMAT command, without the rest of the FORMAT command itself--it only moves the system files, and doesn't erase the target disk. Note that it only copies the very basic startup files, either for DOS or the DOS that underlies Windows. It will not copy an entire Windows installation, for example!

Today, SYS is rarely used for hard disks. Its most common use is to create bootable floppy disks from non-bootable floppy disks. It is also sometimes used to upgrade DOS versions on hard disks of older machines; you boot the floppy of the new DOS version and then SYS the new operating system files to the hard disk. Again, modern operating systems take care of this sort of work internally, and don't require this sort of manual operation.

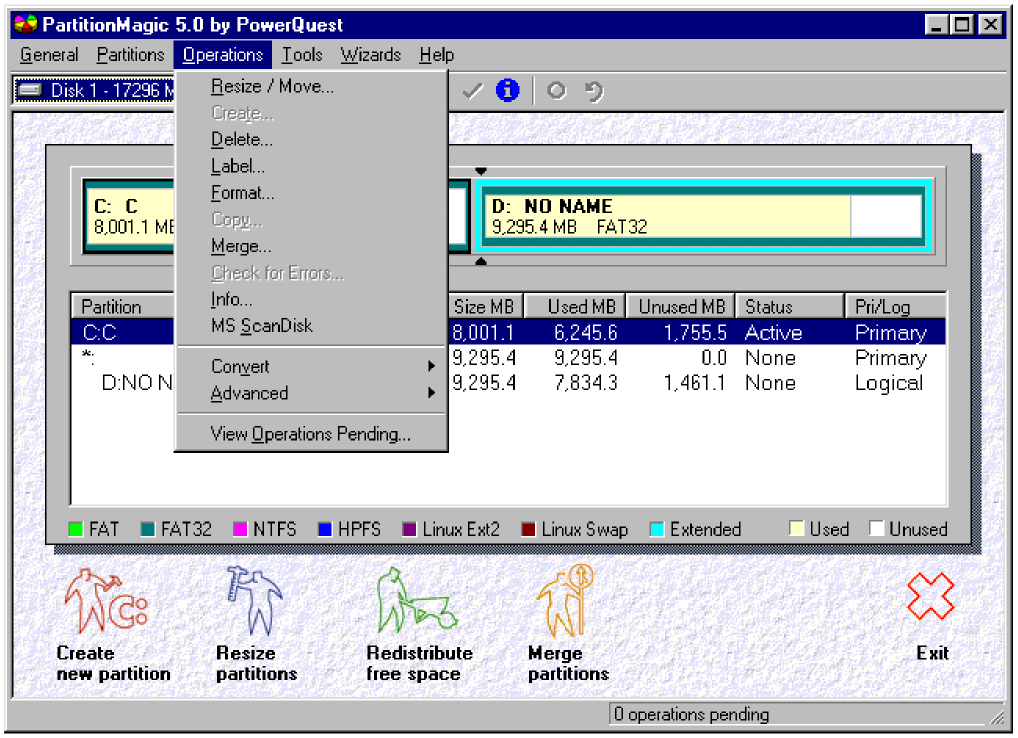
Tip: You can make a bootable floppy from within Windows as well, so you don't really need SYS for this either. Just open the Control Panel, double-click "Add/Remove Programs" and then click the "Startup Disk" tab for instructions.

Next: Partition Management Utilities (Partition Magic, etc.)

Partition Management Utilities (Partition Magic, etc.)

Microsoft provides basic utilities that let you partition and format hard disks for use with their operating systems. These include FDISK, which creates and deletes partitions, and FORMAT, which allows you to format a hard disk to ready it for use. These programs are functional, but rather crude. They let a "typical user" do the basics, but nothing else. If you are someone who tinkers with hard disks a great deal, or works with many systems, you will quickly discover that the tools built into DOS and Windows are inadequate for doing many things you will need and want to do.

To fill this gap, several companies have created third-party partition management utilities, which can do everything that FDISK and FORMAT can do, and a lot more as well. The most famous of these is the Partition Magic program, which is produced by *PowerQuest*. There are also other programs that compete with Partition Magic, though PM is the leader in this segment of the market. (Quarterdeck used to have a program called Partition-It that was similar, but Quarterdeck was swallowed up by Symantec and the Partition-It product seems to have disappeared.)



Partition	Size MB	Used MB	Unused MB	Status	Pri/Log
C:C	8,001.1	6,245.6	1,755.5	Active	Primary
*:	9,295.4	9,295.4	0.0	None	Primary
D:NO N	9,295.4	7,834.3	1,461.1	None	Logical

Main screen of Partition Magic 5.0, with a pull-down menu open to show some of the operations that the program offers.

In addition to the mundane--letting you see the partitions on each of your hard disks, partitioning and formatting disk volumes, assigning labels and so on--partition management utilities typically include the following features:

- **Non-Destructive Partition Manipulation:** You can shrink, expand or move partitions without losing the data on them. This is the primary claim of fame of Partition Magic and programs like it, because this addresses a major weakness of Microsoft's FDISK: it does not allow you to *change* partitions in any way.
- **Partition Copying:** You can copy a partition from one hard disk to another. This is very useful for those upgrading systems.
- **File System Conversion:** If your system supports FAT32, you can convert disks of the appropriate size from FAT16 to FAT32, or vice-versa. You can also convert from FAT16 or FAT32 to NTFS, or vice-versa, on newer versions. This conversion is also non-destructive.
- **Cluster Resizing:** You can change the cluster size of an existing partition, again, non-destructively.
- **Other:** Depending on the software and its specific version, you can also do things like setting up a boot manager for multiple operating systems, creating rescue disks, and so on.

As you can see, this software provides you with a host of capabilities; you probably knew just from reading that list if this type of program is right for you. A typical PC user who buys a retail PC, uses it for a few years and buys a new one, doesn't need this kind of software. For the hobbyist, homebuilder, upgrader or system administrator, it's hard to do without...

Warning: An important caveat about partitioning utilities: some of their operations can take a fair bit of time. This applies particularly to tasks 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.)

Warning: Some companies that produce hard disk controllers, such as Promise (the maker of the popular Promise Ultra series of IDE/ATA hard disk controllers) have publicly made warnings that third-party partitioning and formatting utilities should not be used on drives connected to their hardware. Frankly, I have never seen a reasonable explanation for why this should be a problem, and I know that many people do use Partition Magic and other similar programs on such drives with no problems. I suspect that this exclusion exists primarily to let Promise "cover themselves" in the event of bugs in third-party software, or problems that arise due to software they have not tested. So, I provide this warning so you know their position, and you can decide for yourself what you want to do. Be aware that due to this statement, they will probably not be of much help if you encounter problems with their hardware after using third-party software.

Next: Partition Copying Utilities (Drive Copy, etc.)

Partition Copying Utilities (Drive Copy, etc.)

One of the major weaknesses of the Microsoft operating systems is the lack of a reliable, efficient mechanism for copying the entire contents of a hard disk volume from one disk to another. This is an essential requirement for those who are upgrading to a new hard disk, or perhaps replacing a disk that is in the process of failing. Back in the "good old days" of DOS, it was possible to just copy an entire disk volume file-by-file, using a simple utility. If you did this correctly, there was every reason to expect that the new disk would function properly. Today, however, with long file names, the use of swap files and the Windows registry, it is no longer possible to safely copy a partition from one disk to another file-by-file. I discuss the problems with this in my article entitled Xcopy Xposed. You can copy data files one at a time, but to properly move an entire operating system installation from one hard disk to another, you must copy the entire partition sector by sector, at a low level.

One way to accomplish this task is to use a third-party partitioning program such as Partition Magic. However, such a utility is rather expensive, and if you are just doing a one-time hard disk upgrade, you don't really need all the facilities it provides. Instead, you can purchase a *partition copying utility*, such as the Drive Copy product by PowerQuest (the makers of Partition Magic). This program is basically Partition Magic greatly stripped down in capability. It will let you make a sector-by-sector copy of one or more hard disk partitions from one hard disk to another. It also automatically resizes partitions if the source and target hard disks are of different sizes. With the reduced capabilities comes a reduced price, of course, which is how this product can co-exist with its "big brother", Partition Magic, in the utility software market.

These copying utilities are fairly simple, and pretty easy to use. Most of the warnings that relate to partitioning utilities don't really apply to these programs, as they don't generally change any existing data, just copy it to a new location. The only caveat I would give about something like Drive Copy is this: make sure you really need it before you make your purchase. Many hard disk manufacturers now include a program with their hard disks that will do what Drive Copy does--they include it as a courtesy to their customers. It may be more limited, but it might just do the job for you and save you a few dollars over buying another piece of software.

Note: You can also use drive imaging utilities to copy partitions, though this is generally going to be less efficient due to it being a two-step process: you must create the image on one disk, and then restore it on the other.

Next: Drive Imaging Utilities (Drive Image, Norton Ghost, etc.)

Drive Imaging Utilities (Drive Image, Norton Ghost, etc.)

System builders and administrators--people who create and deploy many PCs, typically in a business or corporate setting--often have a need for special tools to do particular tasks related to hard disk setup. Imagine that you have purchased 100 new PCs for your office, and licensed 100 copies of each of the appropriate pieces of software to be installed on them. Normally, you would have to install the operating system and each application individually onto each machine. This is boring, repetitive work that would take hundreds of hours to do manually.

A better approach is to do the following instead. Install the operating system and applications on one PC, and thoroughly test the installation. When that system is verified to be working, create an *image* of the partition on that PC. Copy the partition to each of the other 99 systems, and then create a new partition from the image on each local PC. This would still take many hours to do, but would take only a fraction of the time required for 100 separate manual system setups.

The power of this sort of automation is such that several different *drive imaging utilities* have been created over the years to enable administrators to do exactly what I described. The program lets you "capture" a partition and save it as a simple file. The file can be moved to other PCs, using a removable storage device, network, or even the Internet. On the other PC it can be restored into a proper partition. Two of the most common imaging utilities are Drive Image by *PowerQuest*, and Ghost, which was purchased from a smaller developer by *Symantec* several years ago and included in the Norton line of disk utilities.

As I mentioned in the page on disk copying utilities, programs such as Drive Copy are basically "stripped down" versions of full-featured third-party partitioning programs like Partition Magic. In a similar way, drive imaging programs are "subsets" of full partitioning programs, but include the ability to save the partition as an image file, which Partition Magic doesn't allow. These utilities also usually include built-in compression, which allows the target image file to be made smaller than the partition it contains, to save disk space and transfer time when the file is copied.

While imaging programs were developed primarily for large-scale systems deployment, they have also found a niche as a backup product. While most conventional backup software concentrates on copying the files on a hard disk one by one, disk imaging utilities can be used to backup an entire hard disk partition, as a whole. I use Drive Image as part of my own backup strategy. In addition to regularly backing up files that have changed, I periodically take an image of my entire hard disk and store it on an external drive. The advantage of this is that if I ever have a hard disk crash, I can replace the disk and rebuild the partition from the stored image file. Everything will be exactly as it was when I made the image. (In fact, I had an opportunity to test this when I had a disk go south on my notebook in 2000.)

Note: You can use a drive imaging program to copy a partition from one disk to another. Just create an image file on one disk and then restore it to the second. However, if you only require this copying ability, you may be better off just using a disk copying utility. You may save some money as well, since disk copying utilities are generally cheaper than imaging programs.

Next: Disk Compression

Disk Compression

There's nothing quite as alluring as the thought of getting something for nothing. If you don't believe me, well, then tell me why lotteries are so successful? :^) In the PC world overclocking has become all the rage of late, promising to let folks run their PCs faster at little additional cost. In the storage world, the enticement comes in the form of a technique that lets you store up to twice as much data in the same amount of hard disk space. Sound good? :^) Well, it does to many others as well, and this is why *disk compression* has been a popular technology for many years.

There are actually many different types of disk compression. It is possible to compress individual files or directories on a hard disk, or even compress an entire disk volume. Today, hard disks are so cheap and so large that compression has become much less of an issue than it once was--few people need to bother compressing a disk volume when you can get a 40 GB hard disk for so little money! However, we are still only a few years removed from the time when PCs had hard disks one hundred times smaller than this. In the 1990s, many PC users needed disk compression just to allow them enough room for the latest operating system, and for their applications and data files. You may still encounter PCs with compressed volumes on them, especially if you maintain a number of PCs in an environment that includes older hardware.

Even though volume-based compression is not used much any more, compression itself remains widely used. Compression not only saves disk space, it can help you organize and archive older files that you don't use regularly. It has also become a standard for allowing the easy downloading of large numbers of files from the Internet. Compression allows them to be packed into a single file that takes less time to transmit than if the files were sent in their regular format.

In this section I talk about disk compression in detail. First, I discuss why disk compression works, describing its fundamental concepts of operation. I then talk about the different types of compressions that are commonly found in the PC world. I then describe volume compression, how it works and what products exist to implement. I then explain file-based compression. Several pages follow that describe how to use compression to reduce slack on FAT16 partitions, how to adjust the compression level in volume compression products, and memory and reliability issues associated with volume based compression.

Note: The NTFS file system includes compression as part of the operating system. See here for a description of this NTFS feature.

Next: Why Disk Compression Works

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.

mehedi hasan
mehedi@joinme.com

Next: Compression Types

Compression Types

There are several different ways that files can be compressed on the hard disk, in terms of the logical mechanisms for performing the compression and decompression. (There are also many different compression algorithms that can be used to perform the compression, but the details of how the compression is actually done are hidden entirely from the user.) Which of these methods you choose depends entirely on the nature of the system you are using, and your compression needs of course.

The most common compression methods used on PCs are as follows:

- **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 program is used to compress one or more files into a compressed file (often called an *archive*), and is also used to decompress the compressed file (in some cases two complementary programs are used). 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, use 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 and Windows 2000 support 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. See here for more on NTFS compression.
- **Volume Compression:** This option is distinctly different from compressing individual files. Using the appropriate operating system, it is also possible to create entire disk volumes that are compressed. This has traditionally been done either through utilities provided with the operating system, or through third-party packages. Volume compression allows you to save disk space without having to individually compress files to use them. 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. As mentioned in the introduction to this section, it is not used much on newer machines any more, because disks today are so large and cheap.

Of these types of compression, utility-based file compression is the most commonly used. It is 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. Newer utilities and operating systems can actually let you access the files contained within compressed files without decompressing them! See this page for more on file-based compression products.

Volume compression, on the other hand, is less commonly used today, though it was once quite popular. It 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. Several other pages in this section also discuss various features of volume compression.

Next: File Compression Products

File Compression Products

Over the last several years, volume compression has taken a nose-dive in popularity. At the same time, however, file-based compression has *increased* in popularity dramatically. The reason that these two technologies have "passed in the night" is simple: the last five years have seen hard disks dramatically drop in terms of price per megabyte, making volume compression much less important. The same time period, however, has seen the rise of file-sharing and distribution using both conventional media and the Internet. File-based compression is an essential component in the easy transmission of files over the Internet, because it allows blocks of files to be both packaged into a single unit, and to be sent more quickly over relatively slow networking technologies, like analog modem dial-up.

There are many different file compression technologies in existence, each of which compresses files in a slightly different way. And for each of these compression algorithms, there are many specific software products that can be used to compress or decompress. I'm not going to even attempt to delineate all the different products that exist, but will instead explain this software in general terms, focusing on what is most commonly used in the PC world.

The most popular family of compression products, by far, is the one based on the *ZIP* format, first widely implemented in the PKZIP and PKUNZIP shareware utilities created by PKWARE, Inc. If you have been using computers since the dark days of DOS, you probably recall using these two programs at one point or another. The ZIP format is, today, pretty much the standard for distributing files over the Internet; programs using this format usually have a ".ZIP" file extension.

With the rise of Windows, PKZIP and PKUNZIP have fallen out of favor, since they use that scary old DOS command line interface. :^) (I still use them, but then I like using old software. :^)) As Windows has become popular, many Windows programs have entered the market that can let you zip or unzip files using a graphical interface. The most popular of these is probably WinZip, though there are literally dozens of products that can access the contents of ZIP files now. While ZIP files are the ones you will most often encounter, you may occasionally run into other compressed files, using formats such as ARJ or RAR. Other compression formats may require the use of a special utility program, though many of the programs that can handle ZIP files can also read these alternative formats.

One final class of file compression product bears special mention. Some utilities are now on the market that integrate ZIP file support directly into the operating system. When such a utility is installed and activated, ZIP files are turned into special Windows folders, and the files within them can be accessed directly without requiring explicit decompression. You can even save files from an application to the interior of a ZIP file, or run a program from within a ZIP file! I use a utility called ZipMagic (originally produced by *Mijenix*, which was bought by *OnTrack Data International* a few years back) that gives me this functionality, and I consider it a life saver. I have it set up so that with a special keystroke I can change all my system's ZIP files into folders, and I can turn it off when I want them to be seen as regular ".ZIP" files again. You can also easily compress a regular folder (directory) by changing it into a compressed "ZIP folder". Very useful stuff, and the closest thing to NTFS's file compression that you can get under the consumer Windows versions.

By the end of the 1990s, ZIP file support was becoming pretty much a necessity. One of the first things most people who set up new systems would do is attempt to download an updated driver or other files, and find that they were packed into ZIP files. So, one of the most common "first installs" in a new PC was a program like WinZip. Recognizing this, Microsoft began to add ZIP file support to Windows. First, it was placed into the "Plus!" add-on for Windows 98. Later, it was incorporated directly into Windows ME. To enable this support you must install or enable the "Compressed Folders" feature within the operating system. Doing this will allow you to access the contents of ZIP files in a format similar to how the Windows Explorer shows regular files. I have not used this feature myself, but it appears to be similar to the way that utilities like ZipMagic function.

Note: There seems to be some dispute as to how closely this integrated product matches what ZipMagic does; some users have said that the Microsoft feature does allow ZIP files to be used like regular folders in a way that is the same as ZipMagic, while others have said that ZipMagic is more full-featured than Microsoft's "Compressed Folders". For their part, OnTrack maintains that ZipMagic provides several features that Microsoft never implemented, and therefore that ZipMagic is still a useful purchase under Windows ME. As usual, your mileage may vary; you should decide for yourself if what ME includes is good enough, based on your needs and wants. (Although it does seem clear that the advanced archive management facilities built into ZipMagic were not implemented by Microsoft.)

Next: Volume Compression Products

Volume Compression Products

Volume compression is no longer nearly as popular as it was in the 1990s. Back then, there were two main products on the market that were frequently used for volume compression in DOS and Windows. You may still run into these if you are maintaining older systems. 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 (or was) a product of *Stac Electronics*, which is now called *Previo*. I don't plan to get into a lengthy review of the two products because that is really beyond the scope of what I am trying to do here. Both do a good job of providing compressed volumes, and each has some advantages and disadvantages compared to the other.

I personally prefer the use of Microsoft's DriveSpace 3 product, for Windows 95 or Windows 98, 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 in the past that Stacker was 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 felt, at the time that I made my selection, that I would be able to rely on the compression being supported fully by the operating system since they were made and tested by the same company. (Well, that's the theory anyway. With Microsoft, you never know. ;^))

It should be noted that the older (DOS-based) 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 or Windows 98.

It's important to realize that volume compression really *is* out of favor today, and this is reflected in the status of volume compression products here in the 21st century. Recognizing that their product was on its way to obsolescence, Stac Electronics stopped producing and supporting Stacker a long time ago; in fact, they aren't even Stac Electronics any more! As for Microsoft, they have been doing their usual "steering" maneuver, where they try to encourage people to stop using older technologies that they don't want to deal with any more. They decided not to implement DriveSpace support for FAT32 disk volumes, which all but eliminated it as a viable option for modern large drives. (In their defense, it isn't really necessary if you have a big hard disk anyway.) The "final straw" came when Microsoft removed support entirely for compressed disk volumes under Windows ME. If you want to keep using DriveSpace volumes on hard disks, you have to stick with Windows 95 or Windows 98.

Next: Volume Compression Operation

Volume Compression Operation

Disk volume compression works by setting up a *virtual volume* on your hard disk. 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 asks 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 because of all the operations required. The more full the disk, the longer it will take, of course. :^)
4. The CVF is hidden from view by the user, through the use of 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 used to set up older systems with, say, 340 MB hard disks, I would often split the disk by creating a compressed drive called D: from say, 150 MB of space from C:. So then C: would 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. D: would typically be able to hold somewhere between 200 MB and 300 MB itself, depending on the compression ratio of the files.

Another option generally provided by the compression software is the ability 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 with the compression.

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

Next: Free Space and the Estimated and Actual Compression Ratios

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. Consider that the amount of free space on a real disk volume is easily determined by examining how many clusters are free in the file allocation table, and multiplying by the cluster size. In contrast, 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, of course, you have not changed at all the capacity of the compressed volume. You have just changed the current guess. :^) As soon as you copy files to the compressed disk, 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, so we lost some of the "estimated" free space the driver had anticipated earlier.

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 JPG image files to it. That person will become confused when he or she cannot fit 1000 MB of JPGs onto such a compressed disk. The reason? JPG files are *already* compressed and cannot be compressed further. As soon as you copy about 500 MB of JPG files to a compressed volume of that size, it will be full. (Most of the time in the computer world, there is no free lunch. With compression you do get a free lunch, but you only get to eat it once! ;^))

Next: Slack Reduction Using Volume Compression

Slack Reduction Using Volume Compression

Slack is the space wasted due to unused storage at the end of the last cluster used to store the data in each file. 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 this overhead. To eliminate this inefficiency, many people use partitioning, for example, to reduce the cluster size of each partition. Another common way that slack is reduced is through the use of FAT32, which allows much smaller clusters on large partitions.

Compression can also be used to reduce slack, in a couple of different ways. The easiest and simplest way to do this is to simply archive infrequently used files. If you have 1,000 files sitting around that you think you may need at some point, but don't access regularly, stick them into a ZIP file. This will automatically reduce the slack taken up by these files by a factor of over 99%, because slack is proportional to the number of files. One file takes up much less slack than 1,000. Of course, this is not a very practical solution for files you use frequently, though tools like ZipMagic can make infrequent access more convenient.

Volume compression can also be helpful for reducing slack in older systems that cannot use FAT32. Volume compression tools like DriveSpace 3 can in fact save space even when set not to compress any files at all, simply 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).

Next: Compression Level and Performance Considerations

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 essential 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 older 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.

Next: Memory Issues with Compression Drivers

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 driver is responsible for "mounting" the compressed volumes and making them appear like regular disk volumes to the operating system and the applications that run on it. This normally occurs automatically when a system with compressed volumes loads. While these drivers generally work pretty well, there can be some issues with them. One is simply the fact that they can be rather large, and can exacerbate conventional memory problems.

Normally, DriveSpace 2 (or DoubleSpace) is used for older DOS versions, and DriveSpace 3 is used for Windows 9x. 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 9x, 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 9x drops down to MS-DOS mode; here the real-mode DriveSpace driver must be loaded to conventional memory, and it is quite large (over 100 kB). This problem is a valid one and cannot be eliminated. Good conventional memory management can reduce the problem, but cannot remove it completely; see here for tips on conventional memory management under Windows 95. Of course, many people today no longer run much of anything at all under straight DOS. For those that do use DOS, however, this is another reason to consider skipping disk compression, unless there is a pressing need for it.

Next: Compatibility and Reliability Issues in Volume Compression

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 $10 per gigabyte and prices falling fast, there is much less of a need to "stretch" the hard disk using compression. Another reason is related to 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, and these are few and far between.

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 most part.

I will state that I have used compression, from DoubleSpace to DriveSpace to DriveSpace 3, on several of my PCs for many years, and I have set up or maintained several dozen PCs that have used one version or another. In all of that time, 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. One of my PCs 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).

Next: New Technology File System (NTFS)

New Technology File System (NTFS)

The FAT family of file systems, including FAT12, FAT16 and FAT32, have been the file systems underlying Microsoft operating systems since the first IBM PCs in the early 1980s. Virtually everyone who has ever used a PC is familiar with at least the basics of this venerable file system. On the whole, it does an *adequate* job of managing the files on a typical PC, especially for machines that are not subjected to demanding use. For home PC users, and for the typical PC in a small business, FAT is generally "good enough". I use it myself every day, and have few complaints about it as a whole.

However, while FAT is acceptable for most uses, it is also a very old, limited and relatively simplistic file system. It has few of the security, capacity and reliability features that are needed by high-end users, and especially, servers and workstations in a corporate networking environment. Recognizing that FAT was not a worthy basis upon which to build its new Windows NT operating system, Microsoft created the *New Technology File System*, or *NTFS*. The goals behind NTFS were to provide a flexible, adaptable, high-security and high-reliability file system, to help position Windows NT as a "serious" operating system for business and corporate users.

In this section, I provide a fairly comprehensive description of the key characteristics of the NTFS file system. I begin with an overview and history of the NTFS file system, and a discussion of the different versions of NTFS. I then describe NTFS's architecture and major structures, and explain how directories and files are organized in NTFS. I then move on to discuss NTFS's security and permissions system, and then several reliability and management features associated with the file system. I then describe some of the "additional" features associated with certain versions of NTFS, and conclude with a discussion of various NTFS implementation issues.

I should point out that the structure of the subsections in this section roughly parallel the structure of the file systems section as a whole. You may want to read the pages discussing the FAT file system before you read about NTFS, because this will provide you with some useful background. However, it isn't strictly necessary.

Note: The NTFS file system is, in many ways, tied inextricably to the Windows NT and Windows 2000 operating systems. It is hard to "draw the line" between what exactly is a discussion of the file system specifically, and what is the operating system as a whole. However, to the degree possible, I have tried to do exactly that, as attempting to explain those operating systems in detail is a task beyond the scope of this site. :^)

Next: Overview and History of NTFS

NTFS Versions

Like most file systems, NTFS has gone through some evolutions. Over time, Microsoft has made changes to it, for many different reasons. These include correcting problems with the file system, adding support for newer hardware, and enabling new operating system features. The biggest change to NTFS came with the introduction to the market of Windows 2000. NTFS changes enable some of the most important features of that operating system.

In this section I describe the two versions of NTFS that are commonly used on PCs. I first talk about NTFS 1.1, which is also called NTFS 4.0 (since it is most commonly used with Windows NT version 4.0). I then discuss the newest NTFS version, NTFS 5.0, which is an integral part of Windows 2000. I conclude with a brief discussion of compatibility between NTFS versions.

Note: The NTFS versions discussed here are used by almost all Windows NT and Windows 2000 installations. Apparently, there was also an earlier version of NTFS, called NTFS 1.0 or NTFS 3.1 (or so I believe), which was used on the first version of Windows NT: Windows NT 3.1. I have been unable to find any information about this version and how it differs from NTFS 1.1 / 4.0. Since Windows NT 3.1 is not really widely used, I am limiting my coverage of that older NTFS version to this note. :^) Still, if you know anything about this first version of NTFS, and are willing to share that knowledge with me, I'd appreciate it!

Next: NTFS 1.1 / 4.0

NTFS 1.1 / 4.0

The most widely-implemented version of the NTFS file system has two different names. It is "officially" called NTFS version 1.1. However, it is also commonly called NTFS 4.0, in reference to its use by the most popular version of the operating system, Windows NT 4.0. NTFS 1.1 is also used natively by the prior version of Windows NT, Windows NT 3.51. It is also supported by several other operating systems, for compatibility purposes, though generally only in "read only" mode. (See this comparison table for more information.)

All of the fundametal characteristics and features explained in the other sections in our coverage of NTFS are supported by NTFS 1.1. When people talk about NTFS, they typically consider NTFS 1.1 to be the "default" version of the file system. Reflecting this, I have not specifically identified each NTFS feature as being supported by NTFS 1.1. Instead, I have labeled the few features particular to NTFS 5.0 appropriately, so you can watch for these notes if you want to be sure to identify where the differences in NTFS versions are. Of course, the following page on NTFS 5.0 also lists the main changes made in NTFS 5.0 as well.

Windows 2000 is designed specifically to use NTFS version 5.0. Because of this requirement, and because Windows 2000 (and its successors) will likely replace Windows NT over time, it seems likely that NTFS 1.1 will eventually be relegated to a "background" role. However, the vast popularity of Windows NT, and its millions of current installtions, mean that NTFS 1.1 partitions will be around for a long time to come.

Next: NTFS 5.0

NTFS 5.0

When Microsoft finally shipped its long-awaited new Windows NT operating system, they surprised a few people by naming it not Windows NT 5.0, as had been expected, but Windows 2000. However, the version of NTFS that shipped with Windows 2000 provided a hint at the original name of the operating system: NTFS 5.0. The fact that Microsoft created a new version of the file system to go along with its new operating system demonstrates just how important the NTFS file system is to Windows NT and Windows 2000. Several of the new features in Windows 2000 actually depend on features built into NTFS 5.0, such as the Active Directory service.

The following are the most important of the new features and capabilities that were added to NTFS with version 5.0. Each has been linked to the page where you can read about it in more detail:

- **Reparse Points:** Files and directories within the file system can have actions associated with them, so that when the file system object is accessed in a particular way, the action is carried out.
- **Improved Security and Permissions:** The mechanisms for managing file system security and assigning permissions were improved.
- **Change Journals:** Disk volumes can be set to keep track of all operations performed on the files and directories they contain.
- **Encryption:** NTFS 5.0 allows you to encrypt files and automatically decrypt them as they are read.
- **Disk Quotas:** Administrators can track how much disk space is being used by users or groups of users, and even limit disk space use if necessary.
- **Sparse File Support:** To save space on the disk, support was added for the more efficient storage of *sparse files*, which are large files that are mostly empty.
- **Disk Defragmenter:** Windows 2000 includes a disk defragmentation program, where Windows NT did not. (Arguably, this is an operating system feature of Windows 2000, and not a file system enhancement, but I thought I'd mention it anyway, since it is obviously file system related.)

Of course, this list isn't comprehensive; there were also some other, minor improvements made in a variety of areas within the file system. This includes fixing some bugs that existed in the older version of the file system, though the new features undoubtedly mean that new ones were included as well. :^) Even the items above represent a rather substantial set of enhancements to what was already a very powerful file system. Of course, NTFS 5.0 also supports all of the features included in older versions of the file system.

As I mentioned above, NTFS 5.0 is required by Windows 2000. In fact, if you install Windows 2000 on top of Windows NT, Windows 2000 will convert any older NTFS partitions to NTFS 5.0. For more on compatibility issues between NTFS 5.0 and older versions of the file system, see the next page.

Next: NTFS Version Compatibility

NTFS Version Compatibility

Microsoft's decision to greatly enhance the NTFS file system under Windows 2000 resulted in a number of new features that most users consider advantageous. However, in creating the new 5.0 version of NTFS, compatibility issues became a concern under some circumstances. In particular, file system compatibility becomes an issue when mixing disk volumes between systems that have different versions of Windows installed. Multiple operating system PCs that have both Windows NT and Windows 2000 installed also may run into difficulties.

There are in fact several different compatibility issues here, which are related. In no particular order:

- **Windows 2000 Automatic Conversion:** Windows 2000 will automatically convert to NTFS 5.0 any NTFS 1.1 file systems it sees when it boots. Even well after the operating system has been installed, if you add an NTFS 1.1 partiton to a Windows 2000 system, it will be converted to NTFS 5.0. This can cause problems, as mentioned above, on systems that boot to both Windows NT and Windows 2000. In some circumstances it may be better to avoid using NTFS under Windows 2000 to avoid this situation.
- **Automatic Conversion of Removable Disks:** Apparently, the behavior above also applies to *removable* media that has been formatted using the older versions of NTFS! This means that those who move files between Windows NT and Windows 2000 machines may need to pay attention to how they use their media. :^)
- **Windows NT Compatibility with NTFS 5.0:** In order for Windows NT to be able to read or write NTFS 5.0 partitions, Service Pack #4 (SP4) or higher must be installed on the system. This patch contains a new version of the NTFS.SYS driver file. However, even though this gives Windows NT access to NTFS 5.0 partitions, the operating system components needed to enable the new features in NTFS 5.0 are not installed. This means that the new features described here do not function under Windows NT, even with SP4 installed.
- **Non-Microsoft Operating System Compatibility:** As shown on this cross-reference chart, certain non-Microsoft operating systems can access both NTFS 1.1 and NTFS 5.0 partitions, in read-only fashion. Being newer, NTFS 5.0 support generally requires a newer version or build number than NTFS 1.1 support.

The PC Guide is not a site that focuses specifically on operating systems, so I do not come even close to discussing all the nuances of Windows NT or Windows 2000 installations. If you are going to be working extensively with these operating systems, I would recommend that you consult more comprehensive documentation on the operating systems, and in particular, issues involved with file system installation and support. This applies even more to those who will be setting up complex systems, such as those that boot more than one operating system.

Next: NTFS Architecture and Structures

NTFS Architecture and Structures

In order to achieve the many goals that Microsoft had for its new file system, it was necessary to be very diligent in the way that NTFS was designed, starting with the fundamentals. The term *architecture* is often used to refer to the basic structures and concepts used to organize a complex system, be it a file system, operating system, microprocessor or what have you. To ensure that NTFS meets the needs of its many demanding users, it incorporates a very specific set of structures and techniques.

In this section, I provide a description of the architecture of NTFS volumes, and the key structures that make up an NTFS partition. This includes an overview of NTFS architecture and an explanation of how NTFS uses certain key file system structures to store information. I also discuss NTFS partition size and cluster size considerations, and contrast these to the way FAT partitions work.

Next: NTFS Architecture Overview

NTFS Architecture Overview

Most of the weaknesses of the FAT file system result directly from the rather simplistic and outdated architecture it uses. No provisions were made in the internal structures of FAT partitions to allow for security and reliability features, making it very difficult to add such capabilities later on. In contrast, NTFS has a special architecture that not only allows for these advanced abilities, but also uses a simple conceptual scheme that makes it easier for more features to be added in the future with a minimum of changes. (In fact, this characteristic was employed when NTFS 5.0 came out with several new options.)

The elegance of the NTFS architecture can be seen in how information is stored in an NTFS partition. Virtually every structure in NTFS is a *file*, including the structures used to manage the partition and maintain statistics and control information about the partition itself. The control information is stored in a set of special files that are created when an NTFS partition is first created; these are called metadata files and include such items as lists of files on the partition, volume information, cluster allocations, and so forth. One exception to the "everything is a file" rule is the partition boot sector, which precedes the metadata files on an NTFS partition and controls the most basic of NTFS operations, such as loading the operating system.

The same simple conceptual model used for files and control structures is extended to the internal level of files as well. Every file in an NTFS partition is a collection of attributes; this even includes the data that the file contains, which is just considered one of many attributes. Other attributes include items such as the file's name and size. This arrangement really a database-like setup--the operating system view files as being objects with various characteristics, and manages them accordingly. This makes it easy to manage files and add attributes if needed in the future.

Internally, NTFS stores all files (including metadata files) using a cluster system--each file is broken into clusters, each of which contain a binary number of 512-byte sectors. On the surface, this is somewhat similar to how FAT stores data, but the implementation of clusters in NTFS is somewhat different. You can read more about clusters under NTFS here.

Next: NTFS Volume Boot Sector

NTFS Volume Boot Sector

When an NTFS partition is first created, the first block of information created on the partition is the *volume boot sector*. This fundamental structure, which is part of a very small block of NTFS management information that is not stored within the master file table on an NTFS volume, is very much the analog of the volume boot sector under the FAT file system. Like volume boot sectors in FAT, the NTFS volume boot sector is sometimes called the *partition boot sector*, *volume boot record* or other similar names.

Note: Despite the use of the word "sector" in the name of this structure, the volume boot sector can in fact be up to 16 sectors (8 kiB) in length.

The NTFS volume boot sector begins in the first sector of the partition, and consists of two different primary structures. Again, these are similar to the structures in a FAT volume boot sector:

- **BIOS Parameter Block:** This is a block of data that contains fundamental information about the volume itself. This block identifies the volume as an NTFS partition, and includes such information as the volume label and its size. In addition, NTFS provides for an *extended* BIOS parameter block, which contains additional information about the volume such as the location of the key metadata files.
- **Volume Boot Code:** This is a small block of program code that instructs the system on how to load the operating system. With an NTFS volume, this code will be specific to Windows NT or 2000, whichever is installed on the system. It will generally load NTLDR, the NT loader program, and then transfer control to it to load the rest of the operating system. Note that this code is also present in the partition as a system (metadata) file.

The volume boot code on a FAT file system partition is a favorite target of virus writers, as changing this code can allow a virus to automatically load whenever the system has started. The higher-security design of Windows NT and 2000, however, makes it more difficult for viruses in the boot sector code to spread, due to the fact that the operating system maintains tighter control over disk access routines once it has loaded. (Viruses are still possible on Windows NT/2000 systems, of course.)

Next: NTFS System (Metadata) Files

NTFS System (Metadata) Files

As I mentioned in the architectural overview, the NTFS file system stores virtually all data, both user data and internal management data, in the form of files. The most important of these are a set of special system files, which are also called *metadata files*. The prefix "meta-" generally refers to something "transcendent" or "beyond"--or merely self-referring. So "metadata files" are files that contain data *about* data. And that's exactly what these files do. They contain internal information (data) about the "real" data stored on the NTFS volume.

These metadata files are created automatically by the system when an NTFS volume is formatted, and are placed at the beginning of the partition. Now, explaining how these files work is a bit complicated. :^) Understanding their location in an NTFS volume requires that I mention another key structure, the NTFS Master File Table (MFT). The MFT is actually one of these metadata files, but it also contains descriptions of the other metadata files, and in some cases entire other metadata files. Yes, it's a bit confusing. :^) Reading the page on the MFT will help you understand how it works, but in a nutshell, here's what happens: the MFT contains a record describing every file and directory in an NTFS volume, and if the file is small enough, its actual contents may be stored in the MFT itself. Since the metadata files are just "files" to NTFS (albeit special ones), they too have records in the MFT. In fact, the first 16 records of the MFT are reserved for metadata files.

Phew, sorry about that, but hey, I didn't design this thing. :^) Actually, the system is hard to explain, but it is logically consistent, and does work well. The table below provides the important information about the metadata files, including their English names, file names, MFT record numbers and a brief description of what each does:

Metadata File Name	File Name	MFT Record #	Description
Master File Table (MFT)	$MFT	0	This is the MFT itself. This seems to be a bit of a chicken-and-egg problem--how can a record in the MFT contain the MFT? :^) It doesn't. This first MFT record contains descriptive information *about* the MFT. This is consistent with how NTFS works--since the MFT itself is just "a file", so it also needs a record in the MFT!
Master File Table 2 (MFT2) or Master File Table Mirror	$MFTMirr	1	This is a *mirror* of the first 16 records of the real Master File Table. It is stored either in the middle of the partition (for Windows NT 3.5 and earlier) or the end of the partition (for Windows NT 4.0 and later). The mirror is a "backup" that is used to ensure that the first few records of the MFT, which of course describe the

			metadata files themselves, can be accessed in case of a disk error on the original MFT.
Log File	$LogFile	2	The transaction logging file for the volume. This is part of NTFS's file system recoverability feature.
Volume Descriptor	$Volume	3	Contains key information about the volume (partition) itself, such as its name, NTFS version, creation time, and so on. See the complete list of NTFS file attributes for more information.
Attribute Definition Table	$AttrDef	4	This table contains the names and descriptions of the various types of NTFS file attributes used on the volume. (It doesn't contain the attributes themselves, but rather descriptions of what the attributes mean. Remember--*meta*data.)
Root Directory / Folder	"." (single period)	5	This is a pointer to the root directory or folder of the volume.
Cluster Allocation Bitmap	$Bitmap	6	Contains a "map" showing which clusters on the volume are used and which are available for use.
Volume Boot Code	$Boot	7	This record contains a copy of the volume boot code (or a pointer to it). The volume boot code is also found in the volume boot sector.
Bad Cluster File	$BadClus	8	A list of all clusters on the volume that have been marked as "bad" (meaning, an error was detected on the volume somewhere in those clusters, so the file system wants to be sure not to use them again.)
Quota Table	$Quota	9	Table containing quota information, if disk quotas are being used on the volume. Only used for NTFS 5.0 or later.
Upper Case Table	$UpCase	10	Table containing information for converting file names to the Unicode (16-bit) file naming system for international compatibility.

Note: Records 11 through 15 in the MFT (the 12th through 16th records, since MFT records are numbered starting from zero) are reserved for future metadata files.

The elegance of the metadata system is that by storing internal information in files, it is possible to expand on the capabilities of the file system--changing "dedicated" internal structures is more complicated. Also, these files do not need to be stored in a specific location on the hard disk, so if a specific area of the hard disk is damaged, they can be moved.

Next: Master File Table (MFT)

Master File Table (MFT)

Probably the most important of the key system (metadata) files that define an NTFS volume, the *Master File Table* or *MFT* is the place where information about every file and directory on an NTFS volume is stored. The MFT is in essence a relational database table, containing various attributes about different files. It acts as the "starting point" and central management feature of an NTFS volume--sort of a "table of contents" for the volume, if you will. It is *somewhat* analog to the file allocation table in a FAT partition, but is much more than just a list of used and available clusters.

When any file or directory is created on the NTFS volume, a record is created for it within the MFT. The size of each record in the MFT seems to be a matter of some controversy; the best that I can tell is that each record is equal to the cluster size of the volume, but with a minimum of 1,024 bytes and a maximum of 4,096. (Meaning that even if 512 byte clusters are used, each MFT record is still 1,024 bytes, and even if clusters greater than 4,096 bytes are used, each MFT record is limited to 4,096 bytes.) However, some sources say that the size of each MFT record is fixed at either 1,024 or 2,048 bytes.

The system uses these MFT records to store information about the file or directory; this information takes the form of *attributes*. Since the size of each MFT record is limited, there are different ways that NTFS can store a file's attributes: as either *resident attributes* that are stored within the MFT record, or *non-resident* attributes, stored either in additional MFT records or in *extents* that lie outside the MFT. See the discussion of file attributes for more details.

Remember that under NTFS, there is no special distinction between the data in a file and the attributes that describe the file--the data itself is just the contents of the "data attribute". This has an interesting implication for small files. If the amount of space required for *all* of the attributes of a file, including the data it contains, is smaller than the size of the MFT record, the data attribute will be stored resident--within the MFT record itself. Thus, such files require no additional storage space on the volume, and also do not require separate accesses to the disk to check the MFT and then read the file, which improves performance.

Larger files get more complicated. As additional attributes are added to a file--either standard attributes defined by the system or new ones created by the user--and as the existing attributes are expanded in size, they may no longer fit into the MFT record for the file. If this occurs, the attributes will be moved out of the MFT and be made non-resident by the file system. Large files will have their data stored as external attributes, and *very* large files may even get so large that the attributes containing pointers to the file data become external attributes themselves! I discuss this nested structuring of files on the page describing NTFS files and data storage.

As more files and directories are added to the file system, it becomes necessary for NTFS to add more records to the MFT. Since keeping the MFT contiguous on the disk improves performance, when an NTFS volume is first set up, the operating system reserves about 12.5% of the disk space immediately following the MFT; this is sometimes called the "MFT Zone". This is a substantial chunk of real estate to reserve, but bear in mind that it is still usable. Regular files and directories will not use this space until and unless the rest of the disk volume space is consumed, but if that occurs, the "MFT Zone" *will* be used. Eventually, if there are enough entries placed in the MFT, as it expands it will use up the "MFT Zone". When this happens, the operating system will automatically allocate more space elsewhere on the disk for the MFT. This allows the MFT to grow to a size limited only by the size of the volume, but this fragmentation of the MFT may reduce performance by increasing the number of reads required for some files, and the MFT cannot generally be defragmented.

Note: The first sixteen records in the MFT are always reserved for the volume's metadata files.

Next: NTFS Partitions and Partition Sizes

NTFS Partitions and Partition Sizes

NTFS partitions are very different from FAT file system partitions on the inside--in terms of their structures and how they function. However, *externally*, they conform to the general rules that apply to all partitions. This is necessary to ensure that the PC boot process can handle an NTFS partition in pretty much the same way that it does a FAT partition. Therefore, like FAT partitions, you can have primary or logical NTFS partitions, and logical NTFS partitions fit within an extended partition. You can find more information about these terms and the rules for partitions on this page.

Since NTFS was designed from the start to be a file system suitable for use in corporate and business environments, it is no surprise that the file system allows very large partitions to be created. Recall that at the time Windows NT was released, the only versions of FAT that existed were FAT12 and FAT16--FAT32 had not yet been created. The maximum partition size of FAT16 under

Windows NT is 2 GiB using 32 kiB clusters, or 4 GiB using the non-standard 64 kiB clusters that other versions of Windows do not support. Considering the large storage needs of businesses, and also considering that many businesses servers use RAID to create even larger volumes, such small size limits would have been completely unacceptable in NTFS, even a decade ago when NTFS was being developed.

Under NTFS, the maximum size of a partition (volume) is in fact 2 to the 64th power. This is equal to 16 binary exabytes, or 18,446,744,073,709,551,616 bytes. Does that seem large enough for your needs? :^) Well, many problems with PC hard disks occurred when unintentional size limits were imposed by engineers who figured that, for example, "2 gigabytes ought to be enough". However, with 18 billion gigabytes it would seem that you should be safe for a while in NTFS. :^)

Then again, it always pays to be careful when looking at such huge numbers. For example, I pointed out elsewhere that FAT32 claims to be able to support up to 2 TiB partitions, but does so at the cost of tremendous slack waste and an enormous file allocation table. NTFS is a completely different file system, but since it was designed at a time when hard disk sizes were measured in single gigabytes, there's no real way to know how well it will scale to disk volumes that are thousands or even millions of gigabytes in size. I suppose we will get there in due time, but as the paragraph below explains, there are already provisos on these large volume limits.

Under Windows NT there are significant restrictions imposed on the size of the boot partition--the first partition on the drive. During installation, Windows NT always first creates a FAT16 partition. Even if you tell NT that you want to install to an NTFS partition, it first creates a FAT16 partition and then *converts* it to NTFS. Since the maximum size of a Windows NT FAT16 partition is 4 GiB, this limits the size of your boot partition as well. Even if you use a third-party tool to increase the size of the partition, you run into another limitation: Windows NT can't boot from a partition larger than 7.88 GiB period, regardless of how you create it (this is associated with Windows NT's older design, which predates the implementation of the Int 13h Extensions required for large hard disk access).

Due to these limitations, many NT setups use at least two partitions--a smaller one for the operating system and a larger one for applications and data. Many people find this a good way to set up the disk anyway, as it keeps operating system files separate from others. These boot size limit problems have been eliminated in Windows 2000. You can format and use an entire large hard disk in a single partition under Windows 2000.

Tip: I have a full page on partitioning strategies under NTFS in the section on implementation.

Note: BIOS issues associated with IDE/ATA hard disks can impose partition size limits irrespective of the operating system or file system. In particular, support for IDE/ATA hard disks over 7.88 GiB requires a BIOS with support for Int 13h Extensions. See here for more details.

Next: NTFS Clusters and Cluster Sizes

NTFS Clusters and Cluster Sizes

One of the ways that NTFS is similar to the FAT file system is that it does not manage individual 512-byte sectors on the hard disk volume independently. Instead, it groups them into blocks that are called *clusters*, or sometimes, *allocation units*. The main reason for doing this is performance: if individual sectors were used for data storage on a very large disk, many resources would be required to keep track of what was in each one, and fragmentation of the disk would be much more of a problem. For a more thorough discussion of clusters and how they work, see this discussion of clusters in the FAT file system section.

While both FAT and NTFS use clusters, they use them in a very different way, of course. This is due to the differences in the internal structures of the two file systems. Some of the performance issues associated with very large FAT file system partitions are due to the fact that the file allocation tables grow to a very large size, and FAT was never created with extremely large volumes in mind. In contrast, NTFS is designed to be able to better handle the large internal structures (such as the MFT) that occur with large partitions.

Like FAT, NTFS chooses a default cluster size based on the size of the partition. However, it uses a different method for selecting the cluster size for a partition than the default cluster assignment system for FAT16 and the one for FAT32. The table below shows the default cluster sizes that NTFS chooses for various partition sizes:

Partition Size Range (GiB)	**Default Number of Sectors Per Cluster**	**Default Cluster Size (kiB)**
<= 0.5	1	0.5
> 0.5 to 1.0	2	1
> 1.0 to 2.0	4	2
> 2.0 to 4.0	8	4
> 4.0 to 8.0	16	8
> 8.0 to 16.0	32	16
> 16.0 to 32.0	64	32
> 32.0	128	64

Now, I am sure you noticed rather quickly that there are some colors in this chart that you normally don't see on the pages of this site. :^) This was done to distinguish the two halves of the chart, because NTFS uses two different assignment systems, depending on the version of the operating system:

- **Windows NT 3.5 and Earlier:** The first versions of the file system use the entire table above. So if you are running Windows NT 3.5 and create a single 6 GB partition, it will use 16 sectors per cluster (8 kiB).
- **Windows NT 3.51 and Later (Including Windows 2000):** Only the first four entries in the table are used. The maximum cluster size is 4 kiB for all partitions over 2.0 GiB, regardless of their size.

The reason for the difference between operating systems is perhaps a bit surprising: it has to do with NTFS's built-in file-based compression. Compression is not supported on file systems with cluster sizes over 4 kiB. Since most people like this feature, the setup and formatting programs in newer Windows operating systems will not choose a cluster size over 4 kiB in size. Windows NT 3.5 and earlier do not support file-based compression, so they don't have to worry about this, and they use the full table. (In practice, it's atypical to find a hard disk much bigger than 4 GB running such an old version of Windows NT anyway...)

The size of the clusters in a partition has an important impact on the performance of the system. While the table above shows the *default* cluster size for NTFS partitions, the default can be overridden by using the "/A" parameter of the FORMAT command. For example, "FORMAT D: /A:8192" will format the D: drive with 8,192-byte clusters. However, you should be careful before overriding the defaults and choosing a larger cluster size for a partition. You will lose the ability to use NTFS file compression, and the amount of *slack* will increase as well. Slack refers to wasted space due to files of variable size being placed into clusters of fixed size. The bigger the clusters, the more space that is wasted; while typically considered a FAT file system issue, this becomes relevant to NTFS as well if you use larger clusters. See this FAT file system page for a full explanation of slack. I also have written a more comprehensive discussion of NTFS partition size and cluster size selection issues. The default of 4 kiB for modern hard disk volumes is generally considered a decent overall compromise.

Note: One final issue has an impact on the cluster size of NTFS partitions, if you are using Windows NT (any version, including 4.0.) Windows NT only uses the cluster table above if you are creating a *new* NTFS partition. If you are *converting* a partition from FAT to NTFS, Windows NT will always make the NTFS volume use the smallest clusters: 512 bytes. In fact, this even applies if you initially install Windows NT to an NTFS partition, because during installation NT always first creates a FAT partition and then converts it to NTFS! See here for more on NTFS conversion. Windows 2000 does not have this limitation.

Next: NTFS Directories and Files

NTFS Directories and Files

Yes, NTFS volumes have directories *and* files. Isn't that good to know? :^) Well, you probably want to learn a bit more about them than *that*, I am sure, and in this part of the NTFS guide I will endeavor to do just that. If you are experienced with the FAT file system used in other versions of Windows, then as a user of NTFS partitions, you will find much that is familiar in the way directories and files are used. However, internally, NTFS stores and manages directories and files in a rather different way than FAT does.

In this section I will explore the fundamentals of NTFS directories and files. I will begin with a look at directories and how they are stored on NTFS volumes. I will then discuss user data files in some detail, including a look at how files are stored and named, and what their maximum size can be. I will then describe some of the more common standard attributes associated with files. Finally, I will discuss *reparse points*, a special enhanced feature present in NTFS 5.0 under Windows 2000.

Next: NTFS Directories (Folders)

NTFS Directories (Folders)

From an external, structural perspective, NTFS generally employs the same methods for organizing files and directories as the FAT file system (and most other modern file systems as well). This is usually called the *hierarchical* or *directory tree* model. The "base" of the directory structure is the *root* directory, which is actually one of the key system metadata files on an NTFS volume. Within this root directory, references are stored to files, or to other directories. Each directory can in turn store any combination of files or more sub-directories, allowing you to create an arbitrary tree structure. I describe these general concepts in more detail on this page discussing the FAT file system.

Note: Directories are also often called *folders*.

While NTFS is similar to FAT in its hierarchical structuring of directories, it is very different in how they are managed internally. One of the key differences is that in FAT volumes, directories are responsible for storing most of the key information about files; the files themselves contain only data. In NTFS, files are collections of attributes, so they contain their own descriptive information, as well as their own data. An NTFS directory pretty much stores only information about the directory itself, not about the files within the directory.

Everything within NTFS is considered a file, and that applies to directories as well. Each directory has an entry in the Master File Table, which serves as the main repository of information for the directory. The MFT record for the directory contains the following information and NTFS attributes:

- **Header (H):** This is a set of low-level management data used by NTFS to manage the directory. It includes sequence numbers used internally by NTFS and pointers to the directory's attributes and free space within the record. (Note that the header is part of the MFT record but not an attribute.)

- **Standard Information Attribute (SI):** This attribute contains "standard" information stored for all files and directories. This includes fundamental properties such as date/time-stamps for when the directory was created, modified and accessed. It also contains the "standard" attributes usually associated with a file (such as whether the file is read-only, hidden, and so on.)
- **File Name Attribute (FN):** This attribute stores the name associated with the directory. Note that a directory can have multiple file name attributes, to allow the storage of the "regular" name of the file, along with an MS-DOS short filename alias and also POSIX-like hard links from multiple directories. See here for more on NTFS file naming.
- **Index Root Attribute:** This attribute contains the actual index of files contained within the directory, or part of the index if it is large. If the directory is small, the entire index will fit within this attribute in the MFT; if it is too large, some of the information is here and the rest is stored in external index buffer attributes, as described below.
- **Index Allocation Attribute:** If a directory index is too large to fit in the index root attribute, the MFT record for the directory will contain an index allocation attribute, which contains pointers to index buffer entries containing the rest of the directory's index information.
- **Security Descriptor (SD) Attribute:** This attribute contains security information that controls access to the directory and its contents. The directory's Access Control Lists (ACLs) and related data are stored here.

So in a nutshell, small directories are stored entirely within their MFT entries, just like small files are. Larger ones have their information broken into multiple data records that are referenced from the root entry for the directory in the MFT. NTFS uses a special way of storing these index entries however, compared to traditional PC file systems. The FAT file system uses a simple *linked-list* arrangement for storing large directories: the first few files are listed in the first cluster of the directory, and then the next files go into the next cluster, which is linked to the first, and so on. This is simple to implement, but means that every time you look at the directory you must scan it from start to end and then sort it for presentation to the user. It also makes it time-consuming to locate individual files in the index, especially with very large directories.

To improve performance, NTFS directories use a special data management structure called a *B-tree*. This is a concept taken from relational database design. In brief terms, a B-tree is a balanced storage structure that takes the form of trees, where data is balanced between branches of the tree. It's kind of hard to explain what B-trees are without getting far afield, so if you want to learn more about them, *try this page*. (Note that the "B-tree" concept here refers to a tree of storage units that hold the contents of an individual directory; it is a different concept entirely from that of the "directory tree", a logical tree of directories themselves.)

From a practical standpoint, the use of B-trees means that the directories are essentially "self-sorting". There is a bit more overhead involved when adding files to an NTFS directory, because they must be placed in this special structure. However, the payoff occurs when the directories are used. The time required to find a particular file under NTFS is dramatically reduced compared to an unsorted linked-list structure--especially for very large directories.

Next: NTFS Files and Data Storage

NTFS Files and Data Storage

As with most file systems, the fundamental unit of storage in NTFS from the user's perspective is the *file*. A file is just a collection of any sort of data, and can contain anything: programs, text files, audio clips, database records--and thousands of other kinds of information. The operating system doesn't distinguish between types of files. The use of a particular file depends on how it is interpreted by applications that use it.

Within NTFS, all files are stored in pretty much the same way: as a collection of *attributes*. This includes the data in the file itself, which is just another attribute: the "data attribute", technically. Note that to understand how NTFS stores files, one must first understand the basics of NTFS architecture, and in particular, it's good to comprehend what the Master File Table is and how it works. You may also wish to review the discussion of NTFS attributes, because understanding the difference between *resident* and *non-resident* attributes is important to making any sense at all of the rest of this page. ;^)

The way that data is stored in files in NTFS depends on the size of the file. The core structure of each file is based on the following information and attributes that are stored for each file:

- **Header (H):** The header in the MFT is a set of low-level management data used by NTFS to manage the directory. It includes sequence numbers used internally by NTFS and pointers to the file's other attributes and free space within the record. (Note that the header is part of the MFT record but not an attribute.)
- **Standard Information Attribute (SI):** This attribute contains "standard" information stored for all files and directories. This includes fundamental properties such as date/time-stamps for when the file was created, modified and accessed. It also contains the "standard" FAT-like attributes usually associated with a file (such as whether the file is read-only, hidden, and so on.)
- **File Name Attribute (FN):** This attribute stores the name associated with the file. Note that a file can have multiple file name attributes, to allow the storage of the "regular" name of the file, along with an MS-

DOS short filename alias and also POSIX-like hard links from multiple directories. See here for more on NTFS file naming.

- **Data (Data) Attribute:** This attribute stores the actual contents of the file.
- **Security Descriptor (SD) Attribute:** This attribute contains security information that controls access to the file. The file's Access Control Lists (ACLs) and related data are stored here.

These are the basic attributes; others may also be associated with a file (see this full discussion of attributes for details). If a file is small enough that all of its attributes can fit within the MFT record for the file, it is stored entirely within the MFT. Whether this happens or not depends largely on the size of the MFT records used on the volume. If the file is too large for all of the attributes to fit in the MFT, NTFS begins a series of "expansions" that move attributes out of the MFT and and make them non-resident. The sequence of steps taken is something like this:

1. First, NTFS will attempt to store the entire file in the MFT entry, if possible. This will generally happen only for rather small files.
2. If the file is too large to fit in the MFT record, the data attribute is made non-resident. The entry for the data attribute in the MFT contains pointers to *data runs* (also called *extents*) which are blocks of data stored in contiguous sections of the volume, outside the MFT.
3. The file may become so large that there isn't even room in the MFT record for the list of pointers in the data attribute. If this happens, the list of data attribute pointers is *itself* made non-resident. Such a file will have no data attribute in its main MFT record; instead, a pointer is placed in the main MFT record to a second MFT record that contains the data attribute's list of pointers to data runs.
4. NTFS will continue to extend this flexible structure if very large files are created. It can create multiple non-resident MFT records if needed to store a great number of pointers to different data runs. Obviously, the larger the file, the more complex the file storage structure becomes.

The data runs (extents) are where most file data in an NTFS volume is stored. These runs consist of blocks of contiguous clusters on the disk. The pointers in the data attribute(s) for the file contain a reference to the start of the run, and also the number of clusters in the run. The start of each run is identified using a *virtual cluster number* or *VCN*. The use of a "pointer+length" scheme means that under NTFS, it is not necessary to read each cluster of the file in order to determine where the next one in the file is located. This method also reduces fragmentation of files compared to the FAT setup.

Next: NTFS File Size

NTFS File Size

One of the most important limiting issues for using serious business applications--especially databases--under consumer Windows operating systems and the FAT file system, is the relatively small maximum file size. In some situations the maximum file size is 4 GiB, and for others it is 2 GiB. While this seems at first glance to be fairly large, in fact, neither is even close to being adequate for the needs of today's business environment computing. Even on my own home PC I occasionally run up against this limit when doing backups to hard disk files.

In the page describing how data is stored in NTFS files, I explained the way that NTFS first attempts to store files entirely within the MFT record for the file. If the file is too big, it extends the file's data using structures such as external attributes and data runs. This flexible system allows files to be extended in size virtually indefinitely. In fact, under NTFS, there is *no* maximum file size. A single file can be made to take up the entire contents of a volume (less the space used for the MFT itself and other internal structures and overhead.)

NTFS also includes some features that can be used to more efficiently store very large files. One is file-based compression, which can be used to let large files take up significantly less space. Another is support for sparse files, which is especially well-suited for certain applications that use large files that have non-zero data in only a few locations.

Next: NTFS File Naming

NTFS File Naming

Microsoft's early operating systems were very inflexible when it came to naming files. The DOS convention of eight characters for the file name and three characters for the file extension--the so-called "8.3 standard"--was very restrictive. Compared to the naming abilities of competitors such as UNIX and the Apple Macintosh, 8.3 naming was simply unacceptable. To solve this problem, when NTFS was created, Microsoft gave it greatly expanded the file naming capabilities.

The following are the characteristics of regular file names (and directory names as well) in the NTFS file system:

- **Length:** Regular file names can be up to 255 characters in NTFS.
- **Case:** Mixed case is allowed in NTFS file names, and NTFS will preserve the mixed case, but references to file names are case-insensitive. An example will make this much more clear. :^) Suppose you name a file "4Q Results.doc" on an NTFS volume. When you list the directory containing this file, you will see "4Q Results.doc". However, you can refer to that file by both the name you gave, as well as "4q results.doc", "4q ReSulTS.dOc", and so on.
- **Characters:** Names can contain any characters, including spaces, except the following (which are reserved because they are generally used as file name or operating system delimiters or operators): **? " / \ < > * | :**
- **Unicode Storage:** All NTFS file names are stored in a format called *Unicode*. Recall that conventional storage for characters in computers is based on the ASCII character set, which uses one byte (actually, 7 bits) to represent the hundred or so "regular" characters used in Western languages. However, a single byte can only hold a couple of hundred different values, which is insufficient for the needs of many languages, especially Asian ones. Unicode is an international, 16-bit character representation format that allow for thousands of different characters to be stored. Unicode is supported throughout NTFS.

Tip: For more information about Unicode, *see this web site*.

You may recall that when Windows 95's VFAT file system introduced long file names to Microsoft's consumer operating systems, it provided for an aliasing feature. The file system automatically creates a short file name ("8.3") alias of all long file names, for use by older software written before long file names were introduced. NTFS does something very similar. It also creates a short file name alias for all long file names, for compatibility with older software. (If the file name given to the file or directory is short enough to fit within the "8.3", no alias is created, since it is not needed). It's important to realize, however, that the similarities between VFAT and NTFS long file names are mostly superficial. Unlike the VFAT file system's implementation of long file names, NTFS's implementation is not a kludge added after the fact. NTFS was designed from the ground up to allow for long file names.

File names are stored in the file name attribute for every file (or directory), in the Master File Table. (No big surprise there!) In fact, NTFS supports the existence of multiple file name attributes within each file's MFT record. One of these is used for the regular name of the file, and if a short MS-DOS alias file name is created, it goes in a second file name attribute. Further, NTFS supports the creation of *hard links* as part of its POSIX compliance. Hard links represent multiple names for a single file, in different directories. These links are each stored in separate file name attributes. (This is a limited implementation of the very flexible naming system used in UNIX file systems.)

Next: NTFS File Attributes

NTFS File Attributes

As I mention in many places in this discussion of NTFS, almost everything in NTFS is a file, and files are implemented as collections of attributes. *Attributes* are just chunks of information of various sorts--the meaning of the information in an attribute depends on how software interprets and uses the bits it contains. Directories are stored in the same general way as files; they just have different attributes that are used in a different manner by the file system.

All file (and directory) attributes are stored in one of two different ways, depending on the characteristics of the attribute--especially, its size. The following are the methods that NTFS will use to store attributes:

- **Resident Attributes:** Attributes that require a relatively small amount of storage space are stored directly within the file's primary MFT record itself. These are called *resident attributes*. Many of the simplest and most common file attributes are stored resident in the MFT file. In fact, some are required by NTFS to be resident in the MFT record for proper operation. For example, the name of the file, and its creation, modification and access date/time-stamps are resident for every file.
- **Non-Resident Attributes:** If an attribute requires more space than is available within the MFT record, it is not stored in that record, obviously. Instead, the attribute is placed in a separate location. A pointer is placed within the MFT that leads to the location of the attribute. This is called *non-resident* attribute storage.

In practice, only the smallest attributes can fit into MFT records, since the records are rather small. Many other attributes will be stored non-resident, especially the data of the file, which is also an attribute. Non-resident storage can itself take two forms. If the attribute doesn't fit in the MFT but pointers to the data do fit, then the data is placed in a *data run*, also called an *extent*, outside the MFT, and a pointer to the run is placed in the file's MFT record. In fact, an attribute can be stored in many different runs, each with a separate pointer. If the file has so many extents that even the pointers to them won't fit, the entire data attribute may be moved to an *external attribute* in a separate MFT record entry, or even multiple external attributes. See the discussion of file storage for more details on this expansion mechanism.

NTFS comes with a number of predefined attributes, sometimes called *system defined attributes*. Some are associated with only one type of structure, while others are associated with more than one. Here's a list, in alphabetical order, of the most common NTFS system defined attributes:

- **Attribute List:** This is a "meta-attribute": an attribute that describes other attributes. If it is necessary for an attribute to be made non-resident, this attribute is placed in the original MFT record to act as a pointer to the non-resident attribute.
- **Bitmap:** Contains the cluster allocation bitmap. Used by the $Bitmap metadata file.
- **Data:** Contains file data. By default, all the data in a file is stored in a single data attribute--even if that attribute is broken into many pieces due to size, it is still one attribute--but there can be multiple data attributes for special applications.
- **Extended Attribute (EA) and Extended Attribute Information:** These are special attributes that are implemented for compatibility with OS/2 use of NTFS partitions. They are not used by Windows NT/2000 to my knowledge.
- **File Name (FN):** This attribute stores a name associated with a file or directory. Note that a file or directory can have multiple file name attributes, to allow the storage of the "regular" name of the file, along with an MS-DOS short filename alias and also POSIX-like hard links from multiple directories. See here for more on NTFS file naming.
- **Index Root Attribute:** This attribute contains the actual index of files contained within a directory, or part of the index if it is large. If the directory is small, the entire index will fit within this attribute in the MFT; if it is too large, some of the information is here and the rest is stored in external index buffer attributes.
- **Index Allocation Attribute:** If a directory index is too large to fit in the index root attribute, the MFT record for the directory will contain an index allocation attribute, which contains pointers to index buffer entries containing the rest of the directory's index information.
- **Security Descriptor (SD):** This attribute contains security information that controls access to a file or directory. Access Control Lists (ACLs) and related data are stored in this attribute. File ownership and auditing information is also stored here.
- **Standard Information (SI):** Contains "standard information" for all files and directories. This includes fundamental properties such as date/time-stamps for when the file was created, modified and accessed. It also contains the "standard" FAT-like attributes usually associated with a file (such as whether the file is read-only, hidden, and so on.)
- **Volume Name, Volume Information, and Volume Version:** These three attributes store key name, version and other information about the NTFS volume. Used by the $Volume metadata file.

Note: For more detail on how the attributes associated with files work, see the page on file storage; for directories, the page on directories.

In addition to these system defined attributes, NTFS also supports the creation of "user-defined" attributes. This name is a bit misleading, however, since the term "user" is really given from Microsoft's perspective! A "user" in this context means an application developer--programs can create their own file attributes, but actual NTFS users generally cannot.

Next: NTFS Reparse Points

NTFS Reparse Points

One of the most interesting new capabilities added to NTFS version 5 with the release of Windows 2000 was the ability to create special file system functions and associate them with files or directories. This enables the functionality of the NTFS file system to be enhanced and extended *dynamically*. The feature is implemented using objects that are called *reparse points*.

The use of reparse points begins with applications. An application that wants to use the feature stores data specific to the application--which can be any sort of data at all--into a reparse point. The reparse point is tagged with an identifier specific to the application and stored with the file or directory. A special application-specific filter (a driver of sorts) is also associated with the reparse point tag type and made known to the file system. More than one application can store a reparse point with the same file or directory, each using a different tag. Microsoft themselves reserved several different tags for their own use.

Now, let's suppose that the user decides to access a file that has been tagged with a reparse point. When the file system goes to open the file, it notices the reparse point associated with the file. It then "reparses" the original request for the file, by finding the appropriate filter associated with the application that stored the reparse point, and passing the reparse point data to that filter. The filter can then use the data in the reparse point to do whatever is appropriate based on the reparse point functionality intended by the application. It is a very flexible system; how exactly the reparse point works is left up to the application. The really nice thing about reparse points is that they operate transparently to the user. You simply access the reparse point and the instructions are carried out automatically. This creates seamless extensions to file system functionality.

In addition to allowing reparse points to implement many types of custom capabilities, Microsoft itself uses them to implement several features within Windows 2000 itself, including the following:

- **Symbolic Links:** *Symbolic linking* allows you to create a pointer from one area of the directory structure to the actual location of the file elsewhere in the structure. NTFS does not implement "true" symbolic file linking as exists within UNIX file systems, but the functionality can be simulated by using reparse points. In essence, a symbolic link is a reparse point that redirect access from one file to another file.

- **Junction Points:** A *junction point* is similar to a symbolic link, but instead of redirecting access from one file to another, it redirects access from one directory to another.
- **Volume Mount Points:** A *volume mount point* is like a symbolic link or junction point, but taken to the next level: it is used to create dynamic access to entire disk *volumes*. For example, you can create volume mount points for removable hard disks or other storage media, or even use this feature to allow several different partitions (C:, D:, E: and so on) to appear to the user as if they were all in one logical volume. Windows 2000 can use this capability to break the traditional limit of 26 drive letters--using volume mount points, you can access volumes without the need for a drive letter for the volume. This is useful for large CD-ROM servers that would otherwise require a separate letter for each disk (and would also require the user to keep track of all these drive letters!)
- **Remote Storage Server (RSS):** This feature of Windows 2000 uses a set of rules to determine when to move infrequently used files on an NTFS volume to archive storage (such as CD-RW or tape). When it moves a file to "offline" or "near offline" storage in this manner, RSS leaves behind reparse points that contain the instructions necessary to access the archived files, if they are needed in the future.

These are just a few examples of how reparse points can be used. As you can see, the functionality is very flexible. Reparse points are a nice addition to NTFS: they allow the capabilities of the file system to be enhanced without requiring any changes to the file system itself.

Next: NTFS Security and Permissions

NTFS Security and Permissions

One of the most important advantages that you gain when choosing the NTFS file system over older file systems such as FAT, is *much* greater control over who can perform what sorts of operations on various data within the file system. FAT was designed in the era of single-user PCs, and contains virtually no built-in security or access management features. This makes it quite poorly-suited to multi-user business environments--can you imagine running a business where any user in the company was free to roam through the file system and open any documents he or she found? This is not a wise way to run a server! In contrast to FAT, NTFS offers a secure environment and flexible control over what can be accessed by which users, to allow for many different users and groups of users to be networked together, with each able to access only the appropriate data.

In this section I take a detailed look at NTFS's security features and how they operate. I begin with a general discussion of NTFS security concepts. I then describe the various NTFS permissions and permission groups that can be assigned to various file system objects. I talk about ownership and how permissions are assigned, and also explain how the inheritance of permissions works, and how NTFS handles resolving multiple permission settings for the same object. Since Windows NT and Windows 2000 handle permissions differently I distinguish between their security models where appropriate.

Note: NTFS security and permission issues are to some degree inextricably linked to features and aspects of the operating system, and also touches upon issues related to Windows NT/2000 networking. A full discussion of Windows NT or Windows 2000 domains, directory services, groups, login procedures and so on is far beyond the scope of our coverage of NTFS. Therefore, I am attempting to limit myself to a description of how security works within NTFS itself--even in this attempt I have probably gone far too much into operating system details. I will not describe Windows NT/2000 security in general; you may wish to consult a broader NT/2000 reference if you need more detail on the operating system's security, account management and access control features than I provide here. In fact, even NTFS permissions themselves can get very complicated, especially under Windows 2000 with its greater and more complex security settings. If you want to know all the ins and outs of controlling permissions you will want to consult a Windows NT or Windows 2000 operating system reference.

Next: General NTFS Security Concepts

General NTFS Security Concepts

NTFS security is really only one part of a much bigger picture: security under Windows NT and 2000 in general. It's no exaggeration to say that security is one of the most important aspects of these operating systems--period. Security, including controlling access to the system and its various resources, is a subject that gets a lot of attention in any NT or 2000 system. Managing security issues such as user accounts and groups is a big part of the job of any Windows NT or 2000 system administrator.

Security in NTFS, like security in the Windows NT or 2000 operating systems themselves, is oriented around the key concept of assigning rights to specific *users* or *groups of users*. Consider a network consisting of a Windows NT or Windows 2000 server, to which are connected various client machines in a network. Any user who sits down at one of these client machines can connect to the server computer, but he or she must *log in* to the server in order to access any of its resources, including NTFS volumes it contains. In fact, the same applies to someone who uses the server machine directly, again, assuming it has been correctly configured.

The manager of the server sets up user accounts for everyone who will use the network. He or she also sets up *group accounts*, to which are added lists of individual users. These groups are used to allow rights to be given to multiple users who share something in common; for example, they may all be in the same department or logical sub-unit in the organization. Someone who does not have a user account on the network may be allowed to use a *guest account*, but the rights assigned to such an account are generally quite minimal, for obvious reasons. If someone does not have even the guest account password, that person will quickly discover that they can do nothing on the server!

The access rights for files and directories on NTFS volumes are assigned based on these same user or group accounts. When a user logs in to a Windows NT or 2000 network, the account that is used becomes the key to what that person can access, including NTFS objects. By looking at the name of the account used to log in to the network, the system determines who the person is and also what groups the person is a member of, and assigns rights accordingly. A user can be a member of a number of different groups simultaneously (just like in "real life"). Several predefined groups are also set up in the system by default, which have specific access rights. One of these is the *Administrators* group, members of which have access to pretty much everything. Other groups that are set up depend on the specific role played by the computer: whether it is a domain controller for example. (Here we start to drift away from NTFS into NT/2000 generalities and networking, so I am going to stop. ;^))

For example, consider a small company of 20 people, with a server that contains a variety of data. There may be a folder on the D: drive on this server called "D:\Budget", which contains budgeting information for the company. This is sensitive data, which is only supposed to be accessible to the President and Vice-President of the company, and their Administrative Assistant. Under NTFS, this is easy to set up by assigning specific permissions to that folder for only those persons' accounts. In fact, it is also easy to arrange the folder's permissions so that, say, the President and Vice-President can read or modify files in the folder, but the Assistant can only read the files. All others in the company can be easily blocked from the folder entirely. A full discussion of how permissions work is provided on the pages describing NTFS permissions and standard permission groups.

There are three other important overall concepts in NTFS security: object *ownership*, *permission inheritance* and *auditing*. Ownership is a special property right for NTFS objects that gives file owners the capability of granting permissions to others. NTFS is also designed to propagate permissions down the hierarchy of the directory structure, under the control of the user. This permission inheritance feature allows permissions to be assigned to groups of objects automatically. It also allows permissions to be automatically applied to new files that are created within an existing directory structure. NTFS 5.0 extended the control that administrators and users have in dealing with permission inheritance. Finally, auditing allows administrators to monitor changes to files or directories.

Next: Access Control Lists (ACLs) and Access Control Entries (ACEs)

Access Control Lists (ACLs) and Access Control Entries (ACEs)

Management of security and access to NTFS objects begins in the same place where everything else begins in NTFS: in the Master File Table (MFT). The MFT record for every file and directory on an NTFS volume contains a security descriptor (SD) attribute. The name of this attribute makes rather clear what it contains: information related to security and permissions for the corresponding object.

One of the most important elements within the security descriptor for any object is the set of lists within it, which dictate which users may access the object, and in what manner. These are called *access control lists* or *ACLs*. Every object in an NTFS partition has two different types of access control lists:

- **System Access Control List (SACL):** This ACL is managed by the system (thus the name) and is used to control auditing of attempts to access the object.
- **Discretionary Access Control List (DACL):** This is the "real" ACL. :^) Well, it is the one that most people are primarily concerned with, because it is where permissions are stored that control what users and groups of users are allowed what type of access to the object. If you hear someone refer to an object's ACL in the singular, this is the one they mean.

Each entry in an ACL is called an *access control entry* or *ACE*. Each ACE contains an ID code that identifies the user or group to which the ACE applies, and then information about the specific permission settings that are to be applied to that user or group. Many different ACEs can be placed into a list, allowing the access of various types to be granted or denied for a variety of different users and groups. Some groups have special meaning, such as the self-evidently named group "Everyone".

The ACL for every object is a combination of various access control settings contained in different ACEs. A typical object may have different sets of permissions assigned for various users or groups of users. In fact, some sets of permissions may conflict with each other, since users can be members of more than one group, and groups may have differing permissions. When an object is accessed, a process of permission resolution takes place, which determines which permissions take precedence and therefore, whether any given attempted access should be allowed or disallowed.

ACLs are also greatly affected by the particular *inheritance model* being used by the operating system. Windows NT uses a static inheritance model, which defaults the ACL for a new object from the ACL of its parent folder. Windows 2000 uses a more advanced dynamic inheritance scheme that provides better control over how the ACLs for an object work, lets subfolders and files have their ACLs change automatically when their parent folder's ACL changes, and allows finer control over inheritance in general. This more advanced functionality can also be applied to Windows NT 4.0 installs using Service Pack 4 and the Security Configuration Manager (SCM).

Next: NTFS Permissions

NTFS Permissions

Access control lists (ACLs) are used to manage which users and groups of users are allowed to access different files and folders (objects) within NTFS volumes. These ACLs contains entries that specify what rights each user or group has for the object in question. These access rights are called *permissions*.

When Windows NT was built, six different permission types were created for NTFS objects. The NT user interface was designed to allow these permissions to be associated with objects. Each permission type controls a different kind of access to an object, and each has an abbreviation letter. These permission types are sometimes called *special permissions*, to differentiate them from standard permission groups that are applied at a higher level.

In some cases, the meaning of a permission is the same for both files and directories (folders); in others, the meaning is different, depending on if the permission is applied to a folder or a file. This table shows the different NT permissions and how they apply to folders and files:

Permission Type	**Abbreviation Letter**	**Permission Granted For Files**	**Permission Granted For Folders**
Read	R	Read file contents	Read folder contents
Write	W	Change file contents	Change folder contents (create new files or subfolders)
Execute	X	Execute (run) a program file	Traverse subfolder structures of folder
Delete	D	Delete file	Delete directory
Change Permissions	P	Change file's permission settings	Change folder's permission settings
Take Ownership	O	Take file ownership	Take folder ownership

Note: There is also one other fundamental permission type: *Delete Subfolders and Files*. This permission, when applied to a parent folder, allows a user to delete files and subfolders within it, even if they do not have delete permission on those files and subfolders. Under Windows NT this permission type cannot be individually applied to folders. It is only available as part of the "Full Control" standard permission group.

Until Windows 2000 was released, these six basic permissions were the lowest level that an NTFS user could access. When Windows 2000 was introduced, the six permission types above were "broken down" into 13 different permission components, to allow for more "fine-tuned" control over different kinds of access. While some people believe this "breaking down" was part of Windows 2000, in fact, these 13 components have *always been present in NTFS*! Under Windows NT, they were just hidden under the six permission types above. The table below lists the different permission components and shows how they correlate to the six Windows NT permission types:

Permission Components (Windows 2000 and Windows NT 4.0 SCM)	**Permission Types (Windows NT)**					
	Read (R)	**Write (W)**	**Execute (X)**	**Delete (D)**	**Change Permissions (P)**	**Take Ownership (O)**
Traverse Folder / Execute File			✓			
List Folder / Read Data	✓					
Read Attributes	✓		✓			
Read Extended Attributes	✓					
Create Files / Write Data		✓				
Create Folders / Append Data		✓				
Write Attributes		✓				
Write Extended Attributes		✓				
Delete Subfolders and Files						
Delete				✓		

Read Permissions	✓	✓	✓			
Change Permissions					✓	
Take Ownership						✓

A few notes about this table:

- Some of the permission components are "combination" permissions; they are illustrated by having two different names, such as "Create Files / Write Data". For these, the first term explains how the permission works when it is applied to a folder, and the second describes its application to a file. As the first table on this page shows, this sort of "double meaning" has been present since the start, but the new names just make it more explicit.
- *Delete Subfolders and Files* can now be applied as an individual permission to folders.
- There is actually a 14th permission component, called *Synchronize*. This permission is used to control synchronization of access to file or folder handles for multithreaded applications. It is sort of a "different bird" from the other permissions, which is why I mostly ignore it. :^)

As you can see, Windows 2000 gives you much more "granularity" of control over individual permissions. The Read, Write and Execute permissions have been broken down into several components. Of course, it's pretty unusual for someone to really need control this fine over most objects. (For example, how often do you think you would want to give permission to someone to write data but not append data to a file? Not frequently.) In fact, even the six Windows NT "special permissions" are often more detail than is really necessary. For convenience, Windows provides several pre-defined standard permission groups to allow commonly-desired sets of permissions to be applied to files and folders quickly.

Tip: The finer permissions granularity introduced with Windows 2000 are also available to Windows NT 4.0 users who have installed Service Pack 4 or later, through the Security Configuration Manager (SCM).

Next: Standard Permission Groups

Standard Permission Groups

Windows NT provides a set of six individual permissions for controlling access to files and folders. Windows 2000 refines these individual permissions even further, into a set of over a dozen different permission components. These NTFS permissions allow for fine control of the access rights of users and groups to NTFS objects, but in many cases they are "overkill". To force administrators to always deal with these fine-grained permissions would be a time-consuming chore.

To avoid the necessity of always setting low-level permissions, Windows defines *standard permission groups*. These are simply collections of the low-level permissions that are given names and can be applied to objects. When you use a permission group, all the components contained in the group are applied to the object automatically.

First, let's look at the standard permission groups for Windows NT:

Standard Permission Group	**Object Types Affected**	**Permission Types Granted (Applies Only To Appropriate Object Types)**						**Description**
		Read (R)	**Write (W)**	**Execute (X)**	**Delete (D)**	**Change Permissions (P)**	**Take Ownership (O)**	
No Access	Folders or Files							Denies all access to the file or folder. The user can see the name of the object, but cannot do anything with it.
List	Folders Only	✓		✓				Users can see the list of files in the folder and traverse subfolders, but cannot view or execute files.
Read	Folders or Files	✓		✓				Users can read files and folders, execute files and traverse

								folders, but cannot change anything.
Add	Folders Only		✓	✓				Users can add files or subfolders to the folder, and can traverse subfolders, but cannot read or execute files.
Add & Read	Folders Only	✓	✓	✓				Users can add files or subfolders to the folder, and can read and execute files in the folder as well.
Change	Folders or Files	✓	✓	✓	✓			The user can read, write, execute or delete the file, or if applied to a folder, the files and subfolders within the folder. Note that this does *not* grant access to delete the folder itself. The user also cannot change permissions on the file or folder, or take ownership

								of it.
Full Control	Folders or Files	✓	✓	✓	✓	✓	✓	All permissions are granted. This also includes the special permission "Delete Subfolders and Files", which can only be given through the "Full Control" group under Windows NT.

Well, that table is probably a bit overwhelming at first glance, but it's not all that confusing if you consider it carefully. Under Windows NT, applying the permission group gives the users the permission types indicated by the checkmarks. Note that the checkmarks apply only to the object type specified. Of particular note, "Add & Read" grants the write permission to the folder, but *not* to the files contained within the folder. Also, the "No Access" group is a "trump card" of sorts; it will override other permission settings. See the discussions of permission settings and inheritance for more on how permission conflicts are addressed.

Under the more advanced Windows 2000 scheme, there are 13 different permission components, which are collected into six different standard groups, as the table below illustrates:

Permission Components (Windows 2000 and Windows NT 4.0 SCM)	**Standard Permission Groups (Windows 2000 and Windows NT 4.0 SCM)**					
	Read	**Write**	**List Folder Contents**	**Read and Execute**	**Modify**	**Full Control**
Traverse Folder / Execute File			✓	✓	✓	✓

List Folder / Read Data	✓		✓	✓	✓	✓
Read Attributes	✓		✓	✓	✓	✓
Read Extended Attributes	✓		✓	✓	✓	✓
Create Files / Write Data		✓			✓	✓
Create Folders / Append Data		✓			✓	✓
Write Attributes		✓			✓	✓
Write Extended Attributes		✓			✓	✓
Delete Subfolders and Files						✓
Delete					✓	✓
Read Permissions	✓	✓	✓	✓	✓	✓
Change Permissions						✓
Take Ownership						✓

Notes: "List Folder Contents" and "Read and Execute" have the same permission components, which is a bit confusing. The differences between them have to do with how NTFS handles inheritance of these permissions. "List Folder Contents" is only used for folders and is not inherited by files within the folder. "Read and Execute" applies to folders and files and is inherited by both. Also, the oddball, 14th permission component, "Synchronize", is a member of all of the groups above.

You may notice, in looking at this table, that the "No Access" group is missing under the Windows 2000 scheme. In Windows NT, all permission groups *except* "No Access" provide "positive access"--saying, in effect, "you are allowed" to do something. "No Access" is the only one that says "you are *not* allowed" to do something. Unfortunately, it is very broad; it really says "you cannot do *anything*". This inflexibility was corrected under Windows 2000 by giving users the ability to allow or disallow any permission group or individual permission. Under this setup, "No Access" simply isn't required. See the discussion of permission assignment for more information on this.

Next: Ownership and Permission Assignment

Ownership and Permission Assignment

Permissions and permission groups control access to files and folders on NTFS volumes, but what controls who can assign permissions? Would you *really* be surprised if I told you that it was, in fact, other permissions? :^) Yes, that is indeed the case. These special permissions work in combination with another key NTFS security concept: *ownership*.

Every object within the NTFS volume has an owner, which is a user identified by the object as being the one who controls it. By default, the user who creates a file or folder becomes its owner. The significance of ownership is that the owner of a file or folder always has the ability to assign permissions for that object. The owner can decide what permissions should be applied to the object, controlling others' access to the file or folder.

The two special permissions that are associated with ownership and permission assignment are "Change Permissions" (P) and "Take Ownership" (O). If a user is granted the "Change Permissions" permission, the user can change the permission settings for the object even if he or she does not own it. If a user has "Take Ownership" permission, the user has the ability to take over ownership of the resource, and of course, once it is owned the user can do anything he or she wants with the permissions. Both of these special permissions are most commonly granted through the standard permission group "Full Control". Note that ownership of an object cannot be assigned or given away. The owner of the object can only give the right to others to *take* ownership. There's an important but subtle distinction there. :^)

Deciding how to assign permissions to various files and folders is an important system administration task. Very careful thought needs to go into how user groups are created and permissions assigned to various objects. One common mistake that many administrators make is misusing the "No Access" permission group under Windows NT. If used incorrectly, this can lock everyone out of large areas of an NTFS volume. Problems can also occur if different users take ownership of files or change permissions when they should not--this is in fact the reason for the distinction between the "Full Control" permission group, and the slightly more restricted groups "Change" or "Modify". One should be very careful when granting "Full Control" permission. Note that by default, members of the "Administrators" user group can always take ownership of, or change permissions on, any file or folder. This allows administrators to fix permission problems if they occur.

The mechanics of assigning permission are rather straight-forward. The most common method is to right-click an object in the Windows Explorer, select Properties, and then click the "Security" tab to access the permissions settings for the object. The exact way that permissions are assigned depends on whether you are using Windows NT or Windows 2000. Since I have already probably gone into far too much detail on permissions, I am not going to delve into the details on exactly how permissions are set. However, I do think it's important to highlight that the NT and 2000 permissions models work differently in one fundamental respect.

Under Windows NT, there is really only one kind of permission assignment possible. Generally speaking, you can only "allow" users to do things. For example, you can allow someone read permission on a folder. By not granting the write permission as well, the system infers that the person may not write the folder. However, there is no way to explicit say "no write permission on this folder for this user". The difference is important, because it has implications in multiple-level hierarchies of folders. The only way to disallow access to something in the Windows NT NTFS security model is to use the "No Access" permission group. This group will over-ride any "allow" permissions also defined for the object, and cut off access to the item for the user or group who is tagged with "No Access". Unfortunately, this is a sledgehammer: it takes away *all* permissions. There is no way to explicitly select a folder and say "regardless of other permission settings and groups that User X may be in, X may not write any files in this folder".

Windows 2000 greatly improved the control you have in assigning NTFS permissions by creating two explicit settings for each permission and permission group: *allow* and *deny*. Using these setting types, it is easy to specifically allow read access and deny write access to a file or folder, enabling the example I just mentioned above to be implemented. This improved control is also the reason why the "No Access" group does not exist in Windows 2000: it isn't needed. Windows 2000 has specific rules for dealing with the various access and deny permission settings that may be set on objects or inherited by them, especially when they may conflict. See the discussion of advanced inheritance for more information on this.

Tip: Windows NT 4.0 users can gain the advantages of the Windows 2000 permission assignment method by installing Service Pack 4 or later and using the Security Configuration Manager (SCM).

Next: Static Permission Inheritance

Static Permission Inheritance

A typical NTFS volume will have thousands of folders and files on it--if not tens of thousands or hundreds of thousands. Can you imagine using such a volume, and being forced to assign permissions to all of those objects? Worse, can you imagine being the *administrator* of such a system? Well, I can, and it's not a pretty sight. :^) Fortunately, the folks at Microsoft made NTFS so that you don't have to worry about this sort of nightmare.

When you are using Windows NT and create a new subfolder or file within a folder, the new object is given a default set of permissions by copying the current set of permissions associated with the object's parent folder. This is called permission *inheritance*, or sometimes, *propagation*. Under NT's inheritance model, this only happens once, at the time the object is created. For this reason, conventional inheritance under NT is also called *static permission inheritance*, to distinguish it from the dynamic inheritance used by Windows 2000.

While static inheritance is certainly much better than *no* inheritance, it creates serious difficulties for administrators who need to manage large directory structures. Imagine a large tree structure of folders. Under static inheritance, after any subfolder is created, its permissions are no longer linked to those of the parent object. This makes it easy for any grouping or "branch" of the tree to have its permissions changed after the face. The administrators have no easy way to keep track of these changes or, in some cases, to even tell that they have occurred. Problems are particularly likely to occur if the "Full Control" permission group has been used, as this means users are free to play around with the permissions on parts of the directory structure. Also, the static inheritance makes it very difficult to add new permissions to an existing structure. Suppose you create a new user group and want to give everyone in that group access to an existing set of directories: how do you do it?

To address these concerns, Windows NT provides a special feature when permissions are being assigned. If you select the "Replace Permissions on Subdirectories" and "Replace Permissions on Existing Files" options when changing the permissions of a folder, NT will reset the permissions of all child objects to match those of the parent. So if you add a new user group and want to give it access to the existing structure, you can use these options to force NT to propagate the new permissions down the directory tree, giving the new user group access to every folder and file.

However, this solution has one *very* big problem with it: whenever you propagate permissions in this manner, you wipe out any custom permissions that may have been set on child objects. This makes it a very inelegant tool--a "permissions sledgehammer", if you will. Now, if your aim is to address the deeds of adventurous users who have been playing with permissions, erasing permissions from subfolders and files may be exactly what you want. However, in many instances, resetting permissions in this way can lead to a disaster. You could have a document storage directory called "C:\Documents" on a server, with a tree containing dozens of subfolders belonging to different people, and permissions set to ensure only the appropriate users and groups can access the files they need. If you need to add a new user group and use the "Replace..." feature just once on "C:\Documents", you will destroy all the customization on these child subfolders and files, an reduce them all to homogeneity.

Unfortunately, there's no real way around this with Windows NT's conventional inheritance scheme. In practice, it means that changes to the directory structure that require using the "Replace..." features must be done many more times, deeper within the directory structure, to avoid unwanted changes. The limitations of this system led to Microsoft's creating the more advanced dynamic inheritance system in Windows 2000 (which is also available to Windows NT 4.0 users who install Service Pack 4.)

Next: Dynamic Permission Inheritance and Advanced Inheritance Control

Dynamic Permission Inheritance and Advanced Inheritance Control

The static permission inheritance method used by Windows NT on NTFS volumes addresses some of the concerns involved in managing large directory structures, but also has some very serious weaknesses. It does not allow an administrator to easily customize the permissions of branches of a directory tree while also allowing the administrator to assign new permissions to an entire existing structure. To correct some of the problems with the static permission inheritance system, Microsoft replaced it with a *dynamic permission inheritance* system in Windows 2000.

The word "dynamic" in the name of this feature tells you much of what you need to know about it. When you create a subfolder or file in a Windows 2000 folder, the child object inherits the parent's permissions, but *remains linked to the parent*. Furthermore, the parent's permissions are stored separately from any permissions that are manually set on the child object. This dynamic linking method solves the two biggest problems with the static inheritance model. First, any changes to the parent folder are automatically inherited by the child objects. Second, any changes that were made to the child object are not destroyed by this automatic propagation. You get the best of both worlds.

Under dynamic inheritance, an administrator or user is able to manage a hierarchical tree of permissions that matches the hierarchical tree of directories. Since each child inherits permissions from its parent, when you set up a hierarchy of three or more levels of folders, the objects deep within the structure will inherit permissions from their parent, "grandparent", "great grand-parent" and so on. This is called *recursion*.

As an example, consider the document folder "C:\Documents". Generic permissions can be applied to this folder that will be automatically inherited by subfolders. At the next level down, say a sensitive folder for executive-level documents called "C:\Documents\Exec", more specific permissions can be applied. And below that one, say in "C:\Documents\Exec\Payroll-Projections", an even more restrictive set of permissions. The lowest level will have explicit permissions that were applied directly to the "Payroll-Projections" folder, plus some permissions that were inherited from "Exec" and some from "Documents". If changes are later made to the highest-level folder, they will be passed down to "C:\Documents\Exec" automatically, and to "C:\Documents\Exec\Payroll-Projections" as well. However, the explicitly-set lower-level permissions will be retained.

In addition to this powerful dynamic inheritance feature, Windows 2000 offers several advanced inheritance control features that give the administrator more power over how inheritance works:

- **Child Protection:** The main security properties dialog box for each object contains a check box labeled "Allow inheritable permissions from parent to propagate to this object". If the check in this box is cleared, this *breaks* the normal inheritance link between this child and its parent (and higher-level ancestors as well). When this is done, the child will no longer dynamically inherit permissions from higher up in the directory tree. Such a child object is said to be *protected* from inheritance changes.
- **Object Selection Control:** When changing permissions on a folder, you can choose if the permissions will be applied to any combination of the folder itself, files within it, or subfolders within it.
- **Recursion Control:** An option exists in the dialog box where individual permissions are assigned called "Apply these permissions to objects and/or containers within this container only". The name of this option is horribly confusing. What it means is that, if selected, permissions you choose are applied only to the folder's *immediate* children, but *not* lower-level objects. So if this were chosen as we selected a permission for the "C:\Documents" folder in the example above, changes would propagate to "C:\Documents\Exec" but *not* "C:\Documents\Exec\Payroll-Projections", the item two levels down.
- **Forced Propagation:** An option called "Reset permissions on all child objects and enable propagation of inheritable permissions" is provided. This works the same way as the "Replace Permissions on Subdirectories" and "Replace Permissions on Existing Files" options from the older Windows NT static permission model. When selected, NTFS will force propagation down to all child objects and remove any permissions that were directly assigned to those child objects. This allows administrators to easily "fix" permission problems in large directory structures.

The downsides to dynamic inheritance and these advanced inheritance control features are few. One is increased complexity: the static permission model is much simpler to understand and apply conceptually, while the added functionality I have described above is more complicated. Another disadvantage of the new system is performance: dynamic inheritance requires more processing resources to deal with changes to files and folders, and to determine which permissions take precedence each time access to an object is attempted.. In fact, this extra overhead is likely one of the reasons that Microsoft chose static inheritance for Windows NT in the first place.

The complexity of the dynamic inheritance system also has an impact on how the system determines whether a given user is allowed a particular type of access to an object. Since child objects can have both explicitly-set and inherited permissions that may conflict, special rules had to be developed to resolve these permissions and determine which have priority.

Next: Permission Resolution

Permission Resolution

Every time a user attempts a particular type of access to an object on NTFS, the system must determine if the access should be allowed. In theory, this is a simple matter of looking at the access control lists for the object, seeing what the permission settings are for the user, and determining if the desired access is allowed. Unfortunately, in reality, it's not this simple. :^) Since every object can have many different permission settings, it is possible that several different permission settings might apply to a particular object and access method. Furthermore, it is possible that these permission settings might conflict. When this occurs, the system must engage in a process of *resolving* the various permissions to determine which ones should govern the access.

Under the Windows NT permissions scheme, inheritance is static, so there is no issue with multiple inherited permission settings. Conflicts can still occur, however, because a particular user can have a permission associated with his user account and also a group of which he or she is a member--or, he or she could be a member of more than one user group. For example, user John may have permissions allowing him read permission on a particular file called "Struct01.acd". However, he may also be a member of the "Engineering" group, and that group may have both read and write access to that same file. There are two rules that are used for resolving permissions in the Windows NT scheme:

1. With the exception of the "No Access" permission group, Windows NT permissions are all "positive"--meaning, they grant permissions, rather than taking them away. Therefore, the more "inclusive" of the permission settings takes precedence. In the example above, John would be allowed read and write access on "Struct01.acd" because the permission he gains as a member of the "Engineering" group is more inclusive than the one he was granted personally.
2. The "No Access" permission group is the only one that has "negative" permissions--it denies access instead of giving it. As such, it is given special status: it trumps rule #1. If someone has "No Access" permission to an object, they are denied *all* access to the object regardless of any other permissions. So in the example above, if John was also a member of the "Accounting" group, and that group had "No Access" permissions for the file "Struct01.acd", John would be totally out of luck, regardless of his other rights settings. This is the reason why "No Access" must be used very carefully!

Windows 2000 offers much better control over how permissions are assigned, as well as the benefits of dynamic inheritance and advanced inheritance control. However, these extra features make permission resolution *much* more complicated. In addition to the potential conflicts caused by users being in more than one user group, as above, you can have conflicts between permissions that were set for the object directly and those that were inherited from any of the object's predecessors: its parent, grandparent and so on. Furthermore, the existence of both "allow" and "deny" permissions complicates matters further. To deal with these complexities, Windows 2000 uses an algorithm that follows these general rules:

1. "Deny" permissions take precedence over "allow" permissions.

2. Permissions applied directly to an object take precedence over permissions inherited from a parent object.
3. Permissions inherited from near relatives take precedence over permissions inherited from distant predecessors. So permissions inherited from the object's parent folder take precedence over permissions inherited from the object's "grandparent" folder, and so on.
4. Permissions from different user groups that are at the same level (in terms of being directly-set or inherited, and in terms of being "deny" or "allow") are cumulative. So if a user is a member of two groups, one of which has an "allow" permission of "Read" and the other has an "allow" of "Write", the user will have both read and write permission--depending on the other rules above, of course. :^)

The system combines these rules into a process that it uses to resolve various permission settings. Since directly-applied permissions take precedence over inherited ones, and "deny" permissions take precedence over "allow" permissions, it first looks for directly-set "deny" permissions, combining them all together for all groups the user is a member off. If it finds sufficient deny permission to refuse access, it is done--the access is refused. Otherwise, it looks at directly-set "allow" permissions. If it finds sufficient permission to allow access, the access is allowed. If not, it continues on; the sequence is as follows:

1. All directly-set "deny" permissions.
2. All directly-set "allow" permissions.
3. All "deny" permissions inherited from the parent.
4. All "allow" permissions inherited from the parent.
5. All "deny" permissions inherited from the grandparent.
6. All "allow" permissions inherited from the grandparent.
7. etc...

Well, this is certainly quite a bit more involved to explain than the NT permission resolution process--but that's the price you pay for the much more capable system implemented in Windows 2000. It's also not that difficult to understand once you get used to it. ;^) Here's an example that may help you to understand what's going on. Note that I am going to use only non-overlapping user groups to try to keep things somewhat manageable and not confuse you further. :^) Let's suppose we have the following permissions set on a structure:

- For folder "C:\Documents", group "Everyone" has an "allow" permission of "Read".
- For folder "C:\Documents\Exec", group "Employees" has a "deny" permission of "Modify", groups "Exec" and "Top Exec" have "allow" permission of "Modify" and group "Assistants" has "allow" permission of "Write".
- For folder "C:\Documents\Exec\Payroll-Projections", group "Assistants" has "deny" permission of "Modify" and group "Exec" has "deny" permission of "Write".

Every member of the company is a member of "Everyone". All department managers and higher executives are members of group "Exec", and their assistants are members of "Assistants". All lower-level managers and workers are members of "Employees". The president and vice-president are members of groups "Exec" and "Top Exec". Now, consider the following access attempts:

- Randy, who is a member of "Employees", tries to read a document in the "C:\Documents\Exec\Payroll-Projections" folder. There are no directly-set permissions that apply to him, but that folder inherits a "deny" on "Employees" from its parent. Randy will fail in his attempt.
- Jane, who is the director of Marketing and a member of "Exec", tries to write a document in the "C:\Documents\Exec" folder. She will succeed because "Exec" has an "allow" permission of "Modify".
- Jane tries to modify a payroll projection in the "C:\Documents\Exec\Payroll-Projections" folder. She will fail because "Exec" is denied permission to write in that folder. The directly-set deny takes precedence over the "allow" at the parent level.
- Lisa, who is the vice-president, attempts to modify a payroll projection in the "C:\Documents\Exec\Payroll-Projections" folder. She will succeed because "Top Exec" has *not* been denied at that folder level; it will inherit the "Top Exec" "allow" permission of "Modify" from the parent.

That probably isn't the best of examples, but this page has already taken *way* too long to write, so it will have to do. :^) At any rate, it gives you an idea of the power you have with the Windows 2000 NTFS permissions system--and how much extra work the system has to do when it needs to figure out whether someone is allowed to do something with a particular file or folder.

Next: Auditing

Auditing

The biggest part of NTFS file system security revolves around controlling access to different types of objects. Obviously, it is quite important to deal with security in the *present*: managing what users are doing and ensuring that access is correct for various files and folders. However, there's another important aspect to security that also deserves attention: keeping records of the *past*. There are many situations where it is essential for system administrators to be able to not only manage what security happenings are occurring immediately, but what they have been in recent days as well. To allow administrators and managers this capability, NTFS includes a feature called *auditing*.

When auditing is enabled, the system can be set to keep track of certain events. When any of these events occur, the system will make an entry in a special auditing log file that can be read by administrators or others with the appropriate permission level. Each entry will indicate the type of event, the date and time that it occurred, which user triggered the event, and other relevant information.

Auditing within NTFS is really just a small part of the various auditing features offered by the Windows NT and Windows 2000 operating systems. These tools allow administrators to keep track of everything from logins, to the use of printers, to system errors. Within NTFS, auditable events are generally accesses of various types, roughly corresponding to the different types of permissions. Auditing can be selected for files and for folders, and can be selected for individual objects or hierarchies of folders, just like permissions can.

Next: NTFS Reliability Features and System Management

NTFS Reliability Features and System Management

One of the biggest problems with the FAT file system is that it includes virtually no built-in reliability features. Anyone who has used FAT has likely experienced file system problems that have occurred as a result of the system crashing, or even operating system or application errors. While these problems can be corrected, they often result in a loss of data. In a business or high-performance computing environment, having the file system become corrupted on a regular basis is simply not acceptable. One of the most important design goals of NTFS was to make the file system more reliable. While reliability features may not be very exciting, they are of paramount importance to many NTFS users.

In this section I take a fairly comprehensive look at NTFS's key reliability and system management characteristics. I begin with a discussion of NTFS's recoverable file system features, including how its transaction-based system operates, and how it uses transaction logging to recover the file system if that becomes necessary. I then describe the change journal feature of NTFS. From there, I move on to talk about error correction and fault tolerance features, and finally, I talk a bit about fragmentation and defragmentation under NTFS.

Next: Transactional Operation

Transactional Operation

Whenever you are using your PC, the operating system (and hence the file system) are constantly reading files and writing files. Each operation that you do on the system usually involves changes to a number of different files and disk structures. In many cases, these changes are *related*, meaning that it is important that they all be executed at the same time. In a traditional file system such as FAT, if something happens to interrupt these operations, there is the possibility that only some of these related structure changes will have been performed. This can leave the file system in an inconsistent state, producing file system errors, such as when clusters are allocated but never linked to a file.

To avoid these problems, NTFS was designed as a *transaction-based* or *transactional* file system, using a concept very similar to that used in many database applications. Take as an example, transferring $100 from your checking account to your savings account. Simplistically, this transaction involves reducing your checking account balance by $100, and increasing your savings account balance by the same amount. However, it is essential that both changes occur--or neither. If only one occurs then either you or the bank are out $100. To ensure that this cannot happen, the database systems used at banks are designed to treat such transactions as *atomic units*--either all the changes in the transaction occur, or none of them do.

NTFS works in a very similar way. Every operation that results in a change to anything on the operating system is considered a *transaction*. Each transaction is made of several different components, each of which results in a change to part of the file system, such as a file's data or other attributes, or the volume's metadata files. A special activity log is maintained by the system (in fact, it is one of NTFS's metadata files). Every time a change is made to any part of the volume, the system records the change in the activity log. These changes include the creation, deletion, or modification of files or directories.

As a particular transaction progresses, NTFS records each of the changes it makes to any part of the volume. Once all of the changes are complete, the transaction is also complete, and a marking is placed in the activity log to indicate that the transaction was successful. This is called *committing* the transaction. However, suppose a problem occurs before all portions of the transaction are complete--for example, a power failure. In this case, the activity log will be left with some of the components of the transaction having been marked as completed, but the transaction not having been committed. This tells the file system that the transaction was interrupted. It can then use the information in the activity log to "undo" the partially-completed transaction (or in some cases, to "redo" changes that need to be reapplied after a system problem). This is called *rolling back* the transaction, so that the file system appears as it did prior to the start of the transaction. This is all part of the NTFS recovery procedure.

Note: It's important to realize that the transactional operation system only guarantees that transactions will be completed in their entirety *if they are completed*. If a transaction isn't completed, it will be rolled back, which may result in the loss of some data. NTFS does not guarantee that data will never be lost, only that files won't be left in an inconsistent state.

Of course, there is a bit of a downside to this transactional system, as there usually is. The necessity of writing information to the transaction log and managing it during the course of every disk operation results in a slight degradation of file system performance. The impact isn't generally considered major, and for most users, the slight reduction in performance is well worth the reliability improvements. NTFS makes this tradeoff in implementing many of its different features.

Next: Transaction Recovery

Transaction Recovery

NTFS is a transaction-based file system. The use of activity logging and transaction management of file system changes allows the file system to maintain its internal integrity by preventing incomplete transactions from being implemented. One key to the operation of the transaction system is the process that is employed to check for and undo transactions that were not properly completed. This is sometimes called *transaction recovery*. Recovery is performed on NTFS volumes each time they are mounted on the system. Most commonly, this occurs when the system is booted or rebooted.

I mentioned in the discussion of NTFS's transactional operation that the overhead from this system reduces performance somewhat. To partially mitigate this impact, NTFS uses caching of some of the logging operations--in particular, it uses a system called "lazy commit" when transactions are completed. This means that the "commit" information associated with a completed operation is not written directly to the disk for each completed transaction, but rather cached and only written to the log as a background process. This reduces the performance hit but has the potential to complicate recovery somewhat, since a commit may not get recorded when a crash occurs. To improve the recovery process, NTFS adds a *checkpoint* functionality. Every eight seconds, the system writes a *checkpoint* to the log. These checkpoints represent "milestones" so that recovery does not require scanning back through the entire activity log.

When recovery is performed, the file system examines the NTFS volume, looking at the contents of the activity log. It scans all log entries back to the last checkpoint, and performs a three-pass recovery procedure:

- **Analysis Pass:** The system analyzes the contents of the log to determine what parts of the volume need to be examined and/or corrected.

- **Redo Pass:** The system "redoes" all completed transactions that were recorded since the last checkpoint.
- **Undo Pass:** The system "undoes" (or *rolls back*) all incomplete transactions to ensure file integrity.

Once again, I think it's important to point out that the transaction logging and recovery features of NTFS do *not* guarantee that no user data will ever be lost on an NTFS volume. If an update of an NTFS file is interrupted, the partially-completed update may be rolled back, so it would need to be repeated. The recovery process ensures that files are not left in an inconsistent state, but not that all transactions will always be completed.

Next: Change (USN) Journals

Change (USN) Journals

When Windows 2000 was released, Microsoft created NTFS version 5.0, which included several new features and improvements over older versions of the file system. One of these was a new system management feature that is very useful for certain types of applications. Under Windows 2000, NTFS 5.0 partitions can be set to keep track of changes to files and directories on the volume, providing a record of what was done to the various objects and when. When enabled, the system records all changes made to the volume in the *Change Journal*, which is the name also used to describe the feature itself.

Change Journals work in a fairly simple manner. One journal is maintained for each NTFS volume, and it begins as an empty file. Whenever a change is made to the volume, a record is added to the file. Each record is identified by a 64-bit *Update Sequence Number* or *USN*. (In fact, Change Journals are sometimes called *USN Journals*.) Each record in the Change Journal contains the USN, the name of the file, and information about what the change was.

It's important to point out what the Change Journal does not contain, as much as what it does. The Change Journal describes the changes that took place, but does not include all the data or details associated with the change. So for example, if you write data to a particular file, the Change Journal will contain an entry that indicates that the data was written, but not the contents of the data itself. For this reason, the Change Journal cannot be used to "undo" operations on file system objects within NTFS. (It's frightening to think about just how much overhead the file system would consume if it tried to keep "undo" data for every file operation!)

The applications that could potentially make use of the Change Journal are many. For starters, it could be very useful for system-level utilities. For example, anti-virus programs could make use of change journals to detect unauthorized changes to files. Backup programs could also make use of the facility to determine which files had changed since the last time a backup was performed. Programs that perform system management tasks such as archival or replication could also make good use of this feature.

Next: Error Correction and Fault Tolerance

Error Correction and Fault Tolerance

In order to increase the reliability of the file system, and hence the operating system as a whole, NTFS includes several *fault tolerance* features. As the name suggests, these are capabilities that improve the ability of the file system to deal with error conditions that may arise when the system is in use. Fault tolerance is very important for business applications where a primary goal is to keep systems running smoothly with a minimum of downtime.

In no particular order, here are some of the fault tolerance and error-handling features that NTFS includes. Note that some of these capabilities are implemented through the use of the NTFS fault-tolerant disk driver, called "FTDISK":

- **Transactional Operation:** The way that NTFS handles transactions as atomic units, and allows transaction recovery, are key fault tolerance features that I have described elsewhere in this section. Recovery is performed automatically whenever the system is started.
- **Software RAID Support:** NTFS partitions can be set up to use software RAID if the appropriate version of Windows NT or 2000 is used. For more information, see the full discussion of RAID.
- **Dynamic Bad Cluster Remapping:** When the fault-tolerant disk driver is used, the file system has the ability to automatically detect when bad clusters have been encountered during read or write operations. When a bad cluster is found, the file system will automatically relocate the data from the bad location and mark the cluster bad so it will not be used in the future. Now, the FAT file system includes the ScanDisk utility that can do this as well, but you must run it manually--with NTFS this can be done automatically. Furthermore, ScanDisk can only identify clusters that have already gone bad, at which point, data may be lost. The FTDISK driver will actually read back data as it is written (sometimes called a "verify" operation) ensuring that data is unlikely to be lost due to a bad cluster at the time of a write. (Bear in mind, however, that it is possible for an area of the disk to "go bad" between the time that the data is written and the time that it is read back.)

These features all help to make NTFS a reliable file system. Of course, they only protect against certain types of problems, and only in certain ways. These capabilities should certainly not be considered replacements for proper backups and other system maintenance!

Next: Fragmentation and Defragmentation

Fragmentation and Defragmentation

Under ideal conditions, file system read and write transfer performance is maximized when files are contiguous on the disk. This means that all of the data in each file would be located in consecutive clusters or blocks within the volume. Contiguous storage improves performance by reducing unnecessary seek motions that are required when data is located in many different places. When files are broken into many pieces they are said to be *fragmented*. Fragmentation is a common occurrence in the FAT file system; if you are unfamiliar with the concept, you may wish to review this full discussion of fragmentation and defragmentation.

The NTFS file system handles the storage of files and directories in a very different way than the FAT file system does. FAT is a very simple, and relatively "unintelligent" file system, that pays little attention to how much fragmentation will result from how it operates. In contrast, NTFS is smarter about how it manages the storage of data. For example, NTFS reserves space for the expansion of the Master File Table, reducing fragmentation of its structures. Overall, fragmentation is less of a concern in NTFS than it is under FAT.

The superior disk management capabilities of NTFS mean that fragmentation is *reduced* compared to FAT. Unfortunately, this led to a popular myth--that NTFS volumes have *no* fragmentation, and therefore never need defragmentation. Microsoft unwittingly exacerbated this problem by not providing any utility to defragment NTFS partitions in Windows NT, implying that defragmentation was unnecessary. But this is simply not the case: NTFS partitions definitely *are* subject to fragmentation. Many users of NTFS have never defragmented their partitions at all, leading to avoidable performance slowdowns over time.

In fact, due to their complexity, NTFS volumes suffer from a variety of different types of fragmentation. Unlike FAT, where a simple cluster allocation system is used, NTFS uses the Master File Table and a combination of resident and non-resident attributes to store files. Due to the flexible way that data is stored, and that additional data storage areas are added as needed, the result can be pieces of data spread out over the volume, particularly when small files grow into large ones. Remember that while NTFS has a much better design than FAT, at its core it does still store data in clusters. The addition and removal of data storage *extents* causes much of the fragmentation of files and directories. As the MFT grows, it itself can become fragmented, reducing performance further.

The solution to the problem of fragmentation under NTFS is the same as it is under FAT: use a defragmenter. :^) For Windows NT, you will need to use a third-party program, one of the most popular being the Diskeeper program by

Executive Software. In its wisdom, Microsoft decided to license the Diskeeper defragmenter technology and include it in Windows 2000, so the operating system now includes a built-in defragmenter, though it is likely either less capable or slower than the full Diskeeper program sold by Executive Software. (As an interesting aside, Microsoft found themselves in the hot-seat as a result of this licensing decision. The German government took issue with the defragmenter because the CEO of Executive Software is a member of the Church of Scientology, and a big hoo-ha resulted. How bizarre. As the old expression says, "I am not going there", but you can read about this strange episode *here*.)

Next: Other NTFS Features and Advantages

Other NTFS Features and Advantages

"But wait! There's more!"
-- TV infomercial pitchmen

Most of the advantages of NTFS compared to simpler, older file systems such as FAT relate to its fundamental characteristics. NTFS offers a superior architecture, support for larger files, security features such as access control and logging, enhanced reliability, and more. Basically, these are all the features and characteristics of the file system that I have been describing in the other sections on NTFS. However, in addition to those key attributes of NTFS, Microsoft has also added some other little goodies. They don't necessarily fit all that well into the other categories so I put them in this convenient "miscellaneous" area.

In this section I discuss some of the "extra" features that help make NTFS an efficient, useful and secure file system. I begin by discussing NTFS's convenient file-based compression feature. I then talk about NTFS POSIX support and attempt to explain what exactly that means. :^) I then talk about how NTFS encryption works, describe NTFS disk quotas, and also explain NTFS's sparse file support.

Note: Several of the features described in this section are found only in the newer NTFS 5.0 version of the file system found in Windows 2000.

Next: Compression

Compression

Disk compression is a technique that exploits a known characteristic of most types of data: they have repeating patterns. By using software that implements special compression and decompression algorithms, it is possible to store most files in a way that they take up less space on the disk than they normally would. Compression became very popular in the late 1980s and early 1990s, when software and data began to grow in size much faster than hard disks were. Products were created to allow for the compression of entire FAT volumes, and also files or groups of files. I discuss disk compression in general terms, including how it works and how it is implemented, in this section of the site.

One of the most useful features that is built into NTFS is file-based compression that can be used to compress individual files or folders on almost any NTFS partition, under Windows NT 3.51 or later. While Microsoft's consumer operating systems such as Windows 3.1 and Windows 9x support compression of entire disk volumes (under some circumstances), they do not let you easily decide to compress a portion of a volume. Doing that has required resorting to the use of a third-party file compression utility. Under NTFS, you can easily compress one or more files or folders by opening their properties and telling the operating system to compress them. The compression is handled by the operating system during writes, and decompression is automatic whenever an application needs to read the file.

As is the case for FAT volume compression, the performance impact of NTFS compression is a complex question. Performance when working with compressed files can be degraded compared to regular files, as a result of the overhead required to compress and decompress the file on a regular basis--it takes CPU time. On the other hand, compressing a file means that it takes up less space on the disk, which reduces the amount of time required to write to the volume or read from it, potentially counteracting the compression overhead. In general, since hard disks today are quite large, most people use NTFS compression only on infrequently-used files. For example, if you have large numbers of old database files that you aren't using, you can save a lot of space by compressing them. Of course, you might be better off archiving them, to CD-RW or other storage media. Still, most NTFS users consider compression a useful feature of the file system.

Note: NTFS compression is not supported on NTFS volumes that use a cluster size greater than 4 kiB. This is a primary reason why the default cluster size for NTFS volumes is never made larger than 4 kiB under Windows NT 3.51 or later.

Next: POSIX Support

POSIX Support

One of the "miscellaneous" features that was designed into the NTFS file system was support for the *Portable Operating System Interface* standard, abbreviated *POSIX*. POSIX is a set of standards that was created to enhance the portability of applications between computer systems. As the name implies, POSIX is based somewht on the UNIX operating system and its constructs. The POSIX standards are administered by the *IEEE*.

There are actually a number of different POSIX standards. NTFS specifically supports the POSIX.1 standard, which is the standard for application program interfaces based on the "C" programming language. In practical terms, POSIX support is manifested most obviously in NTFS's support for special file naming provisions. For example, NTFS allows for file names to be case-sensitive, and also allows for hard links to be established, enabling multiple names for the same file within the file system. The "last accessed" date/time stamp for files is also part of POSIX support.

You may be wondering--what does POSIX support do for *me*? In general, the answer is "not much". :^) The purpose behind NTFS's POSIX support is to facilitate the migration of software to the Windows NT environment. Unless you are a programmer, you probably will never need to know anything more about POSIX support under NTFS than what I am telling you here. Based on my research, it appears that even if you *are* a programmer, NT POSIX support may be much more about sizzle than steak. The POSIX support as implemented is considered very rudimentary, meaning that the portability of programs is also limited. Some more cynical observers believe that the POSIX support in NTFS was included solely to enable Microsoft to satisfy a requirement for sales to United States federal government agencies.

Next: Encryption

Encryption

One of NTFS's design goals was to allow for proper access control and security, something that was sorely lacking under the FAT file system. In fact, a rather complex security and permissions system is used by NTFS to ensure that only authorized system users can gain access to, or control of, various system objects. This system works well in most cases, but it has a very serious shortcoming: it only works when users "play within the system". As long as Windows NT/2000 is booted normally, the protections offered by the NTFS security mechanisms work well. Unfortunately, it has always been possible for a malicious user to try to access an NTFS partition using a low-level disk utility, bypassing the NTFS security methods entirely. Since NTFS structures were not encrypted, security could be compromised without a lot of trouble--the average person would not have any way of making sense of an NTFS partition when looking at raw bits and bytes, but knowledgeable people certainly could.

To correct this deficiency, Microsoft introduced an encryption capability in NTFS 5.0, as part of Windows 2000. This feature is called the *Encrypting File System* or *EFS*. Using EFS, it is possible to encrypt important data before storing it on the NTFS partition. Without the proper decryption key, the data cannot be accessed. This makes it impossible for anyone to easily access data stored on NTFS volumes by booting the PC with a floppy disk and using a disk sector editor, for example. It also offers some peace of mind to those who carry critically sensitive information around on notebook PCs, which are frequently lost--or "liberated", if you know what I mean...

Not surprisingly, the details of how EFS works are fairly complicated--there's no way to make a capable and secure encryption system without it being fairly complex. The system uses a public key and private key algorithm, with 128-bit security domestically (in North America) and 40-bit keys internationally. The "public key / private key" mechanism is a common one, also used for example in the *PGP* ("Pretty Good Privacy") encryption system. Trying to explain the encryption system would lead me down a long tangential path I would like to avoid. :^) In a nutshell, it works this way. Each user has a public and a private key; the public key can be known to others, while the private key is, of course, private. When a file is encrypted, this is done using the public key. In order to decrypt the file, the private key must be known. EFS carefully guards these private keys in order to ensure that only the person who encrypted the file can decrypt it. The two-key system means that you can encrypt a file using the public key, but you cannot decrypt the file using it!

Fortunately, the internal details aren't necessary in order to use the feature. Enablying encryption is generally as simple as "turning on" encryption for one or more files or folders, much the way NTFS compression works--using the object's properties settings. Encrypting a folder means that any files added to that folder in the future will be automatically encrypted as well.

Tip: If you are interested in learning more about the details of how EFS works, *try reading this article*.

Technically, EFS is not considered a "built-in" part of NTFS. EFS runs as a system service under Windows 2000, and interacts very closely with the internal services and drivers that operate the NTFS file system, but they are really not the same thing. When a file needs to be encrypted or decrypted, the file system works with the EFS service to handle the translation operations for the file. Again, these implementation details are hidden from the user--the operation of EFS is essentially "seamless" and for all intents and purposes can be considered part of NTFS, which is why I described it here. :^)

Next: Disk Quotas

Disk Quotas

In the discussion of the new encryption system added to NTFS 5.0, I mentioned how the lack of encryption in early versions of NTFS was a significant weakness in the NTFS security model. Another weakness of NTFS--this one more of a little, nagging annoyance--is that it did not allow for easy management of disk space usage. While competing products like Novell NetWare offered the ability to control how much of the volume was taken by users, this was not possible under NTFS. Under Windows NT, it is possible for any user to use as much of the disk space on a volume as he or she can grab. This can be a serious administrative problem for systems in some organizations that are shared by many users. In theory, a single user could consume all of the available disk space on a server volume--either intentionally or unintentionally.

To correct this situation, Microsoft introduced quota management as a new feature in NTFS 5.0 under Windows 2000. Microsoft had apparently been planning on this feature for some time, since the metadata file required to implement quotas has been present since Windows NT version 3.5, but it only started to be used in NTFS 5.0. As the name suggests, quota management allows administrators to keep track of how much disk space is consumed by each user, and limit space consumption as well.

The quota system implemented in NTFS 5.0 is quite flexible and includes many capabilities: You have the ability to do the following:

- Set quotas on a per-user or per-volume basis. This lets you limit space used on particular disks, or overall total space use for a person.
- Set a "limit" level and a "warning" level, or both. The user is blocked from using any disk space above the "limit" level. He or she may use space beyond the "warning" level, but a warning will be generated.
- Monitor and log events that cause a user to go over the "limit" or "warning" levels.

Of course, disk quotas are an optional feature; it is not mandatory for administrators to use them. Speaking as someone who has used UNIX systems with quotas before, I can state with authority that they are best thought of as a "necessary evil", and used carefully. :^) It can be annoying to have users fail at what they are doing because they have run out of disk space. It is a good idea to institute quotas only if they are really and truly needed. If you do use them, be sure not to make them too restrictive--if you do, not only will your users be inconvenienced, so will you when they call asking to have their quotas increased. :^) Using the "warning" mode may be a good compromise in some circumstances.

Next: Sparse File Support

Sparse File Support

One of the more interesting features added to NTFS version 5.0 under Windows 2000 is special support for *sparse files*. As the name suggests, these are files that are primarily empty. Now, it is an inherent characteristic of most files that they contain areas of "empty space"--long stretches of zeroes. These zeroes are stored along with the "real data" in the file, and take up space the same way. If files are not too large, storing these zeroes does not represent much of a problem. Sure, some space is expended on these "empty" areas, but this is expected and manageable.

However, there are certain applications that may employ very large files that contain extremely large areas of empty space. A common example is a large database file, which may contain strings of millions of zeroes, representing either deleted data, or a "space-holder" for future data storage--such as unused database fields. There are also scientific applications that make use of very large files where small pieces of data must be spread throughout the file, but much of the file is empty. If such a file is stored in the conventional way, a great deal of disk space is wasted on these repeated "blanks".

You may be wondering--why bother with support for a special feature to address these files when NTFS already includes built-in compression, which could be used to save the space used by the long strings of zeros in sparse files? It's a good question. :^) The most likely reason is that compression introduces overhead that may not be acceptable for critical applications. Much as database systems are likely to have sparse files, they are also often heavily-used applications where compression might introduce unacceptable performance degradation.

To improve the efficiency of storing sparse files without necessitating the use of compression, a special feature was incorporated into NTFS under Windows 2000. A file that is likely to contain many zeroes can be tagged as a *sparse file*. When this is done, a special attribute is associated with the file, and it is stored in a different way from regular files. The actual data in the files is stored on the disk, and NTFS keeps track of where in the file each chunk of data belongs. The rest of the file's bytes--the zeroes--are not stored on disk. NTFS seamlessly manages sparse files so that they *appear* to applications like regular files. So, for example, when an application asks to read a particular sequence of bytes from a sparse file, NTFS will return both regular data and zeroes, as appropriate, just as it would for a regular file. The application can't tell that NTFS is playing "storage games" on the disk.

Note: Tagging a file as sparse is a "one-way trip"--you cannot change the file back into a normal file again.

Next: NTFS Implementation Considerations

NTFS Implementation Considerations

If you've already read the other sections in the site's coverage of NTFS, you have by now probably received a very thorough understanding of what NTFS is all about. Its many features and options make NTFS the choice of a growing number of PC users. However, all of the advanced capabilities of NTFS do not come without a cost. Along with its enhanced operation, NTFS brings considerable *complexity*. There are a number of specific issues and potential difficulties that can arise when employing the NTFS file system in the "real world".

In this section, I take a look at several important issues related to implementing NTFS volumes. I begin with a discussion of NTFS compatibility with various hardware and operating systems. I then describe some of the issues related to converting between NTFS partitions and those of other file systems. I then discuss some of the performance and overhead concerns associated with implementing the advanced features that make NTFS so capable. I proceed from there to discuss NTFS partitioning strategies--how to decide how to split up a disk when using NTFS. I conclude by explaining how NTFS compares to other file systems, and in particular, I tackle the common question of how to choose between NTFS and FAT.

Next: NTFS Hardware and Operating System Compatibility

NTFS Hardware and Operating System Compatibility

NTFS is a file system, and a file system is essentially a set of logical constructs that dictate how the space on a disk volume is to be used. This means that a file system is in many ways more software than hardware. Since it is primarily logical and not physical in nature, this means that NTFS can be used with pretty much any hardware. You won't encounter problems with using NTFS on different types of hard disks for example, and the type of controller or interface is not likely to be an issue either. As long as Windows NT or 2000 can recognize the hard disk, you can use NTFS on it.

The concerns with NTFS hardware compatibility start to show up more when you move out of the realm of hard disks. NTFS is a complex file system that Microsoft designed when the only writeable random access media commonly used were hard disks and floppy disks. Microsoft made a decision early on to not support NTFS on floppy disks. The reason for this is fairly obvious: the special NTFS metadata files represent significant overhead that would consume much of the small storage capacity of a floppy, and writing them would likely increase formatting time considerably. Even in the mid-1990s, floppies had been relegated to chores such as short-term backup or transfer between machines--why would anyone want to bother putting NTFS on a floppy anyway? Since that time, third-party software writers have been busy, and *Sysinternals actually has a freeware utility that will format an NTFS floppy*. However, as even they admit, the utility shows that "NTFS on floppies, while possible, is not a great idea". :^)

Removable media such as Iomega Zip or Jaz drives, or the Castlewood Orb drive, fall in between the universal compatibility of hard disks, and the general incompatibility of floppies. You need to check with the maker of the drive to find out if they support NTFS. For example, Iomega officially does not support NTFS on Jaz or Zip drives, while I believe that Castlewood does support NTFS. It's important to note that in some cases, the lack of support for NTFS is more of a business decision than a technical one--saying "we don't support X" cuts down on technical support calls and potential complaints, so many companies do this today. Often, even if NTFS is officially unsupported, it will work just fine on the drives. Obviously, you are "on your own" if you proceed to format a drive with NTFS when the drive's manufacturer refuses to support NTFS.

In terms of operating systems, it will come as no surprise to you that Windows NT and Windows 2000 are the primary operating systems using the NTFS file system. In order to exploit the advantages of NTFS version 5.0, you need to use Windows 2000. Consumer versions of Microsoft's Windows operating system provide no native support for NTFS, though it may be possible to add this with a third-party driver. Non-Microsoft operating systems now offer some access to NTFS partitions as well, depending on the version--also, this is often just read-only access, not full access. For more information, see this overview of PC file systems, or refer to this operating system and file system cross-reference.

Next: NTFS Partition Conversion

NTFS Partition Conversion

One way of getting an NTFS partition on a system that supports the file system is to create one from scratch, of course. :^) This is pretty easy to do. However, there are also situations where you may wish to transform an existing partition from another file system into NTFS. To do this requires that the partition be *converted*. Conversion causes the operating system to change the structures of the partition to match the NTFS file system, while retaining all of the directories and files that the volume contains. Most commonly, FAT file system partitions are converted to NTFS.

Microsoft provides a tool as part of the Windows NT and Windows 2000 operating systems that allows you to convert a FAT partition into NTFS. Unsurprisingly, it is called "CONVERT.EXE". :^) You run it from the command line using an invocation such as "convert c: /fs:ntfs", which tells Windows to convert the "C:" volume into the NTFS file system. The conversion is fairly straight-forward, but there are some caveats you should keep in mind when performing a conversion:

- Conversion is a "one-way trip". The CONVERT utility does not allow you to convert from NTFS back into FAT (though third-party utilities may; see below.)
- The conversion process requires several megabytes of free space on the volume, to allow for the creation of disk structures and other overhead. Don't try to convert a partition that is completely full or the process may fail.
- NTFS partitions that are converted from FAT partitions under Windows NT are assigned a cluster size of 512 bytes, which may reduce performance compared to larger cluster sizes that are the default for new NTFS partitions. Windows 2000 has corrected this problem. See this page for more discussion of this issue.

Tip: For more information on using the CONVERT program, see *this page in the Microsoft Knowledge Base*. For more general information on the NTFS partition conversion process, *try this one*.

Another option is to make use of third-party partitioning utilities such as Partition Magic. Most of these programs include the ability to convert a partition from NTFS to FAT or vice-versa, and some will also convert to other file system formats as well. Of course, the disadvantage with a third-party program is fairly obvious: you have to pay to buy the software!

Warning: As with any extensive, low-level disk modification, conversion runs the (small) risk of data loss, especially if the process is somehow interrupted. It is advisable that you back up the entire contents of the partition before starting the conversion. It's also a good idea to use a UPS during the process, and avoid such operations during times when the electrical service is unstable (such as during a thunderstorm.)

Next: NTFS Performance and Overhead Considerations

NTFS Performance and Overhead Considerations

One of my favorite expressions is from a Heinlein novel: TANSTAAFL, which stands for "there ain't no such thing as a free lunch". What it means is simply that you usually don't get something for nothing. Life is full of tradeoffs, and certain costs that go with certain benefits. This is true of most things in life; it is true in the computer world in general, and it is also true of NTFS.

When it comes to NTFS, the tradeoff is generally between functionality and performance. NTFS has a lot of interesting and useful features and capabilities. However, most of these abilities come at the cost of *overhead*. The more options you add to a file system, the more disk space and processing is required to implement and manage them. This is simple common sense, but it is important to point out to those considering the use of NTFS. You can't expect to have a file system that allows flexible access control, compression, auditing, advanced reliability and other useful qualities without requiring the system to make some investments in allowing these features to work.

The performance implications of NTFS features mean that using NTFS may result in a slight decrease in performance compared to the use of a simpler file system such as FAT. Here are a few tips and issues to keep in mind when considering how to implement NTFS partitions, which may partially mitigate this performance impact:

- **Use Only What You Need:** Some of the fancier features in NTFS can impose more significant overhead penalties on the system than others, both in terms of processing power and the use of disk space. If you *need* those features, then in most cases they are worth the cost. However, it makes sense to avoid using features that you don't need. For example, don't compress frequently used files if you have lots of disk space; don't enable lots of audit logging unless you really require it, and so on.
- **Use Adequate Hardware:** Trying to implement a complex system on a low-powered system with a small, slow hard disk is probably asking for trouble. The overhead associated with NTFS becomes less and less noticeable on newer hardware (which is part of what is allowing Microsoft to add even more new features to the file system.)
- **Watch The Cluster Size:** While most people think of cluster size as being strictly a FAT file system issue, NTFS also uses clusters, and its performance is impacted by the choice of cluster size. Cluster size is in

turn affected by the choice of partition size. In addition, under Windows NT the cluster size is different if the partition was converted from FAT. See here for a more thorough discussion of these issues.
- **Fragmentation and Defragmentation:** Contrary to a popular myth, NTFS partitions *do* become fragmented and performance can benefit from regular defragmenting. Windows 2000 comes with a defragmenter built into it; for Windows NT you will need to use a third-party tool.

It is wise to keep these performance issues in mind, both when using NTFS, and even in considering if you even want to use it. At the same time, you should not let these concerns "scare you off" from using NTFS. If you manage your system well and use reasonably modern hardware, the difference in performance between using NTFS and FAT partitions should not be a significant one. For a business, NTFS is pretty much indispensable, and if a bit more hardware needs to be thrown at the system to compensate for a small amount of overhead, it's usually an easy decision.

Note: For a more thorough, general discussion of hard disk performance, see this section.

Next: NTFS Partitioning Strategies

NTFS Partitioning Strategies

When setting up a new system using the NTFS file system, one decision that you need to make is how you are going to set up your NTFS partitions. Modern hard disks are very large; in many situations, it makes sense to not put all of the space in a hard disk into a single partition. NTFS is often used for server machines, which may make use of RAID technology to allow the creation of *very* large logical disks. If you have a 500 GB RAID array, it is certainly conceivable that you might not want all of that space put in a single volume!

I discuss partitioning issues, including the various tradeoffs in choosing numbers and sizes of partitions, in this comprehensive discussion of partitioning. While that discussion is oriented around the FAT file system, many of the same points apply to NTFS. A key one is that NTFS is like FAT in that the cluster size of the partition affects its performance, and the default cluster size depends on the size of the partition selected. Other general "common sense" issues are also relevant to NTFS as well as FAT; for example, not splitting up your disk into *too* many partitions to "organize it".

In addition to that section, here are a few specific points that you may want to keep in mind when partitioning a system with NTFS:

- **Limit the Number of Partitions:** NTFS is designed to be able to gracefully handle much larger partitions than FAT can. As a result, you should avoid the temptation to "go overboard" and segment large disks or RAID arrays into many little volumes. Doing this makes the system more difficult to manage in most cases and results in not much improvement in performance.

- **Consider A Dedicated Operating System Partition:** Many people who set up NTFS systems use two partitions. The "C:" (boot) volume is made smaller (a few gigabytes) and dedicated to the operating system and the installation of low-level utilities and administration tools. The other volume (normally "D:") is used for applications and data. This split may make administration easier and avoid problems associated with using too large a boot partition size with Windows NT.
- **Adjust Cluster Sizes If Needed:** The default cluster size on an NTFS partition can be overridden, as described here. You may wish to use larger or smaller clusters than the system would normally select, depending on your needs. For example, you may want a larger cluster size for a partition that will be holding large multimedia files. Do be aware that there can be consequences to going away from the default values. For example, NTFS compression will not work on a volume that uses clusters greater than 4 kiB in size.
- **Beware of Converting Partitions to NTFS Under Windows NT:** Converting a partition from FAT to NTFS under Windows NT results in the NTFS partition being assigned the smallest possible cluster size, 512 bytes. This may cause a degradation in performance.
- **Multiple Operating System Compatibility:** Some systems use multiple operating systems, some of which cannot natively read and write NTFS partitions. For example, Windows 9x/ME cannot read and write NTFS partitions. On these machines, some people create one FAT partition, in addition to any NTFS partitions, for compatibility and file transfer purposes. Note that this is not required for accessing files on an NTFS partition over a network; files on an NTFS partition can be shared across a network even if the network client's operating system cannot use NTFS locally. See this page for more information.

These suggestions should help guide you as you set up an NTFS system and determine how to partition it. Once again, I would encourage you *not* to worry *too* much about fine-tuning your partition selection strategy. Most of the other system setup and administration decisions you make in setting up the PC will have more of an impact on overall performance.

Next: NTFS and Other File Systems

NTFS and Other File Systems

Many systems that are set up to use the NTFS file system use it exclusively. If you consider the reasons why NTFS is often used, this makes a lot of sense. For example, consider a centralized server implemented in a small manufacturing company. The security features of NTFS would be employed to manage and control access to important company information. It wouldn't make a lot of sense to have some of the data stored in this secure environment and some of it also stored in a more mundane FAT partition where anyone could access it.

However, there are situations where you may wish to set up a PC that has both an NTFS partition and another file system partition, such as FAT16 or FAT32. This is commonly done on systems that use multiple operating systems--so called "dual boot" PCs. There is one particular situation that is worth mentioning. In the 1990s, many dual-boot PCs were set up that used Windows 95 OEM Service Release 2 (OSR2) or Windows 98, and also Windows NT. If you do this and set up Windows 9x on a FAT32 partition and Windows NT on an NTFS partition, you will quickly discover that you have a little problem: neither operating system can read the other one's partition. This means that there is no way to share files between the two operating systems! There are several workarounds for this problem: you can upgrade the Windows NT install to Windows 2000 (which supports FAT32) or you can use FAT16 instead of FAT32. Another common solution is to create an additional, small FAT16 partition that serves as "common ground" between the two operating systems (since both Windows NT and Windows 9x can read and write FAT16 volumes.)

Generally speaking, setting up both NTFS and other file system partitions on the same machine does not cause problems. A computer running an operating system that supports NTFS and another file system like FAT can be set up with one or more of each type of partition. The two will not conflict with each other. Of course, you will only get the benefits and features of NTFS when you are using the NTFS volume.

Next: NTFS vs. FAT

NTFS vs. FAT

FAT and NTFS are the two most commonly used file systems in the PC world. Since many PC users are now starting to discover the benefits of Windows NT and Windows 2000 (when compared to their consumer grade counterparts such as Windows 9x or ME) they are learning that they have an important file system choice to make. They must decide whether to keep using FAT, as many have done for years in other Microsoft operating systems, or to "take the plunge" and go for NTFS. In fact, this may well be the most commonly-asked question associated with NTFS: "should I use FAT or NTFS?"

As with many commonly-asked questions, there are no simple answers. :^) As with other popular "X or Y" questions such as IDE vs. SCSI, the reason that the question exists is that both alternatives fit certain niches and needs. I can't give you a definitive answer regarding which file system you should choose, because I don't know the particulars of your situation. That's the honest answer. Many people aren't satisfied with this, of course... they want me to tell them which is "better". This is a bit like asking which is "better": a mid-sized family sedan or a pick-up truck? A Ferrari or a motorcycle?

Most frequently, the question of NTFS vs. FAT is answered by looking at the advantages and disadvantages of NTFS, and comparing it to the simpler FAT file system. This can be made easier by assessing three general questions:

1. Do you need the added features and capabilities that NTFS provides but FAT does not?
2. Are you willing to accept the additional hardware requirements necessary to use NTFS, and to deal with its drawabcks and limitations?
3. Can you invest the additional time and resources for proper administration of an NTFS system?

Again, *I* cannot answer those questions for you, nor will I endeavor to do so. I would encourage you to read the various pages that describe the various benefits and drawbacks of NTFS, and decide for yourself. The questions above will help you.

The only other rule of thumb that I would use is this one: the larger the organization, or the greater the number of people that will be using a PC, the more likely it is that you will want to use NTFS on it. Even leaving aside the other features of NTFS, the security provisions of that file system make it pretty much a necessity if you are going to set up a server with files shared by many different users and groups. For very small organizations, access control is something that may be dispensable, but for a company of say, 20 or more people, it becomes quite important. Of course, in *some* medium-sized organizations security may not be required, but those are pretty atypical. For individual PC users, NTFS may well be overkill, depending on how the PC is being used.