17 DOS File System

The user doesn't see many of the tasks that DOS performs. This is why some users underestimate the complexity of DOS. For example, DOS requires many data structures for handling a mass storage device, although this is not always realized by the user. The file system must perform many steps before executing even small tasks, such as copying files or searching for a file on the hard drive.

Disk drives recognize tracks, sectors, and read/write heads instead of files and subdirectories. Therefore, DOS's file system addresses all available mass storage devices at both physical and logical levels. A file system of this type consists of a series of data structures, which describe the capacity and contents of a disk drive. In this chapter we'll discuss how the DOS file system is designed and how it works.

Basic Structure Of The File System

The volume is the basis of the DOS file system. From the user's viewpoint, DOS addresses mass storage devices as volumes. Each individual volume is assigned a letter. Floppy disk drives are identified by the letters A and B, while the letters C or D usually identify a hard drive. Although a hard drive can be divided into multiple volumes, a floppy disk drive can consist of only one volume.

DOS Versions 3.3 and lower limited volume size to a maximum of 32 megabytes. Therefore, any hard drives with capacities over 32 megabytes were divided into multiple volumes. In this chapter we explain how DOS Version 4.0 broke the 32 megabyte barrier.

Every DOS volume has its own structure, regardless of whether it's intended for diskettes or hard media. The size of the storage media isn't relevant to the structure because it only affects the number of individual data structures required to manage the volume.

Volume label names

Although not required, each volume can be assigned a volume label name when created. The DIR command lists volume label names when they're available. Each volume has its own root directory, which can contain multiple subdirectories and files. These subdirectories and files can be maintained and manipulated by using one or more of the interrupt 21H functions.

Sectors

DOS subdivides each volume into a series of sectors organized sequentially. Each sector contains a specific number of bytes (usually 512) and is assigned a consecutive logical sector number, beginning with sector 0. A 10 megabyte volume contains 20,480 sectors, consisting of logical sectors 0 to 20,479. DOS cannot control the physical arrangement of the sectors. This is controlled by the device driver, which mediates between the volume and DOS. The device driver's distribution of logical sectors (organized or not) is unimportant. It's only important the device driver clearly differentiates between logical and physical sectors.

Since DOS API function calls with interrupt 21H are directed to files instead of individual sectors, DOS converts these file accesses into sector accesses. To do this, DOS uses the volume's directory structure and a data structure known as the FAT (File Allocation Table).

The following table shows the basic structure of a mass storage device:

Mass storage device structure		
Sector number ↓	0	Manufacturer's name, device driver, boot routine First file allocation table (FAT) One or more copies of FAT Root directory with volume label names Data register for files and subdirectories

As we mentioned, every volume is divided into areas containing the various DOS data structures and individual files. The FORMAT command creates these data structures when you format a disk. Since the size of the individual areas can differ depending on the type of mass storage device (and the manufacturer), every volume contains a boot sector.

The Boot Sector

The boot sector contains all the information required to access the different areas and data structures. DOS creates this sector during disk formatting. Boot sectors always have the same structure and are always located in sector 0 so DOS can find and interpret it properly. The table on the right lists the layout of the boot sector.

Boot sector layout

Address	Contents	Type
00H	Jump to boot routine (E9xxx or EBxx90)	3 bytes
03H	Manufacturer's name and version number	8 bytes
0BH	Bytes per sector	1 word
0DH	Sectors per cluster	1 byte
0EH	Number of reserved sectors	1 word
10H	Number of FATs	1 byte
11H	Number of entries in root directory	1 word
13H	Number of sectors in volume	1 word
15H	Media descriptor	1 byte
16H	Number of sectors per FAT	1 word
18H	Sectors per track	1 word
1AH	Number of read/write heads	1 word
1CH	Number of hidden sectors	1 word
1EH-1FFH	BOOT ROUTINE	

The term "boot sector" is used because DOS "boots" (i.e., starts) from this sector. Since DOS usually isn't stored in permanent PC memory (ROM), it's loaded and started from disk. After you switch on the computer, the BIOS takes over system initialization. It loads physical sector 0 (not logical sector 0) of the floppy disk or hard drive into memory. Since device drivers aren't in memory, the BIOS checks physical sector 0 for information. It then loads the boot information from that sector. Once it completes its work, the BIOS starts execution at address 0.

The boot sector always contains an assembly language JUMP instruction at address 00H. Execution of the boot sector's program code begins at this address. After execution, the program continues at a location further into the boot sector. This instruction can be either a normal jump instruction or a "short jump." The field for this jump instruction is 3 bytes long, but a "short jump" only requires 2 bytes. Therefore, a NOP (No Operation) instruction always follows the "short jump" to fill in the extra byte. As its name suggests, this NOP instruction doesn't do anything.

A series of fields, which contain certain information about the organization of the media, follow. The first field is 8 bytes long and contains the manufacturer's name, where the medium was formatted, as well as the DOS version number that performed the formatting. The field may also contain the name of a software manufacturer (e.g., if a program such as PCTools formatted the volume).

The next fields contain the physical format of the media (i.e., the number of bytes per sector, the number of sectors per track, etc.) and the size of the DOS data structures stored on the medium. The physical device information is needed when using the interrupt 13H BIOS functions. These fields

NOTE

Some sources of undocumented DOS structures state the BPB is a parameter table, which can be accessed by using Get Service Data, instead of part of the boot sector. These sources imply the boot sector information is only part of the BPB.

are called the BIOS parameter block (BPB). DOS uses this information for various tasks.

Three additional fields, providing other volume information to the device driver, follow the BPB. However, these fields aren't used directly by DOS.

Bootstrap

Next is the bootstrap routine, to which the jump instruction branches at the beginning of this boot sector. This routine handles the loading and starting of DOS through the individual system components (see Chapter 3).

Several reserved sectors may follow the boot sector. These sectors can contain additional bootstrap code. The numbers of these sectors are recorded in the BPB in the field starting at offset address 0EH. This field terminates the boot sector; a 1 in this field indicates that additional reserved sectors don't follow the boot sector. This applies to most PCs, because no versions of DOS have required a bootstrap loader that cannot fit into the first boot sector.

The File Allocation Table (FAT)

DOS must know which sectors of the volume are still available before it can add new files or enlarge existing files. This information is contained in a data structure called the FAT (File Allocation Table), located immediately next to the media's reserved area. Each entry in the FAT corresponds to a certain number of logically contiguous sectors called clusters, on the media. Location 0DH of the boot sector specifies the number of sectors per cluster as part of the BIOS Parameter Block. Only powers of 2 (1, 2, 4, 8, etc.) are acceptable values. On an XT hard drive, this location contains the value 8 (8 consecutive sectors form a cluster). AT, 386, and 486 hard drives have only 4 sectors per cluster.

As the table on the right shows, the number of sectors comprising a cluster depends on the storage medium.

Sectors per cluster

Device	Sectors per cluster
Single sided disk drive	1
Double sided disk drive	2
AT hard drive	4
XT hard drive	8

Despite the values in the previous table, a formatting utility, such as FORMAT, isn't limited to these cluster values. This applies particularly to volumes containing more than 32 megabytes. DOS Versions 4.0 and higher accommodate larger volumes by adding clusters. A volume larger than 32 megabytes quickly exceeds the 65,536 limit for clusters. So, the capacity of the file system can be extended at will, without changing the current structure, by including more sectors per cluster.

The following table (lower right) shows the clustering used by DOS Version 4.0 for volumes larger than 32 megabytes and up to 2 Gigabytes.

Volume and Cluster sizes of DOS 4.0

Volume size (megabytes)	128	256	8K	16K	32K
Cluster size	2K	4K	8K	16K	32K
Sectors per cluster	4	8	16	32	64

This clustering allocates more memory to the last cluster of a file. This cluster is filled only when its size represents a multiple of the cluster size.

File fragmentation

The idea of joining several sectors into a cluster is based on the logic used by DOS to write files to a medium. Instead of selecting adjoining sectors for file storage, DOS fragments (disassembles) the file so the various pieces can fit into the available sectors.

The following explains the reasons why DOS fragments the file. Users are constantly creating and deleting files. If you start with an empty volume, new files and their respective clusters are stored in sequence. However, when you delete a file, a gap may develop between two sectors. DOS doesn't adjust the clusters on a volume because this process requires too much time. This is especially true for larger volumes. So, DOS places new files in the gaps created when the older files were deleted. If the entire file could be stored in one of these gaps, it could be kept in one contiguous unit. However, this usually doesn't happen. As a result, DOS divides files and fits them into available gaps.

This process slows file access because the read/write head must be repositioned after almost every read function. To avoid an excessive disassembly of the file, DOS gathers several sequential sectors on the media into a cluster. This ensures that at least the sectors of a cluster contain a portion of a file. If DOS didn't use clusters, a file of 24 sectors could be stored in numerous sectors, which would require the read/write head to be positioned a maximum of 24 times to read the entire file. The cluster principle saves a lot of time, since a file comprised of 4 sectors per cluster is stored in 6 clusters and the read/write head must be repositioned only 6 times.

However, there is a problem with this process. Since a file is assigned at least one cluster, some storage space is wasted. Consider the AUTOEXEC.BAT file, which is usually no longer than 150 bytes. Normally, this file could be stored on a single sector (and still waste almost 400 bytes). However, AUTOEXEC.BAT occupies a cluster of 2048 bytes on an AT, which wastes more than 1.5K of hard drive space.

Disk optimizing programs, such as BeckerTools Disk Optimizer or DEFRAG in MS-DOS 6.x, solve this problem by reorganizing the medium and storing all the files in consecutive clusters.

The FAT layout

Now let's return to the file allocation table. The size of individual entries in the FAT under DOS Versions 1.0 and 2.0 is 12 bits. For DOS Versions 3.0 and up, the size of an entry in the FAT depends on the number of clusters. If a volume has more than 4,096 clusters, then each FAT entry is 16 bits; otherwise each FAT entry is 12 bits.

A 12-bit FAT permits control of 4,096 clusters, which corresponds to 4 sectors per cluster, providing a total of 8 megabytes. Although this amount could be expanded by adding more sectors to a cluster, such an expansion isn't recommended. Therefore, you'll find only 16-bit FATs on newer hard drives of 20 megabytes and up, thus allowing the 65,536 maximum of addressable clusters.

The number of bits per FAT entry must be determined before file access. The information in the BIOS parameter block is used for this purpose. The total number of sectors in the volume can be found starting at location 13H. Divide this number by the number of sectors per cluster to obtain the number of clusters in the volume.

The first two entries of the FAT are reserved and aren't related to the cluster assignment. Depending on the sizes of the individual entries, 24 bits (3 bytes) or 32 bits (4 bytes) can be available. The first byte contains the media descriptor, while the value 255 fills in the other bytes. The media descriptor, which is also stored in address 15H of the BPB, indicates the device the media uses (e.g., a diskette). The table on the right shows which codes are possible.

Media descriptor codes

Code	Device	Description
F0H	3.5" disk drive	2 sides, 80 tracks, 18 sectors per track
F8H	Hard drive	Varies
F9H	5.25" disk drive 3.5" disk drive	2 sides, 80 tracks, 15 sectors per track 2 sides, 80 tracks, 9 sectors per track
FAH	5.25" disk drive 3.5" disk drive	1 side, 80 tracks, 8 sectors per track 1 side, 80 tracks, 8 sectors per track
FBH	5.25" disk drive 3.5" disk drive	2 sides, 80 tracks, 8 sectors per track 2 sides, 80 tracks, 8 sectors per track
FCH	5.25" disk drive	1 side, 40 tracks, 9 sectors per track
FDH	5.25" disk drive	2 sides, 40 tracks, 9 sectors per track
FEH	5.25" disk drive	1 side, 40 tracks, 8 sectors per track
FFH	5.25" disk drive	2 sides, 40 tracks, 8 sectors per track

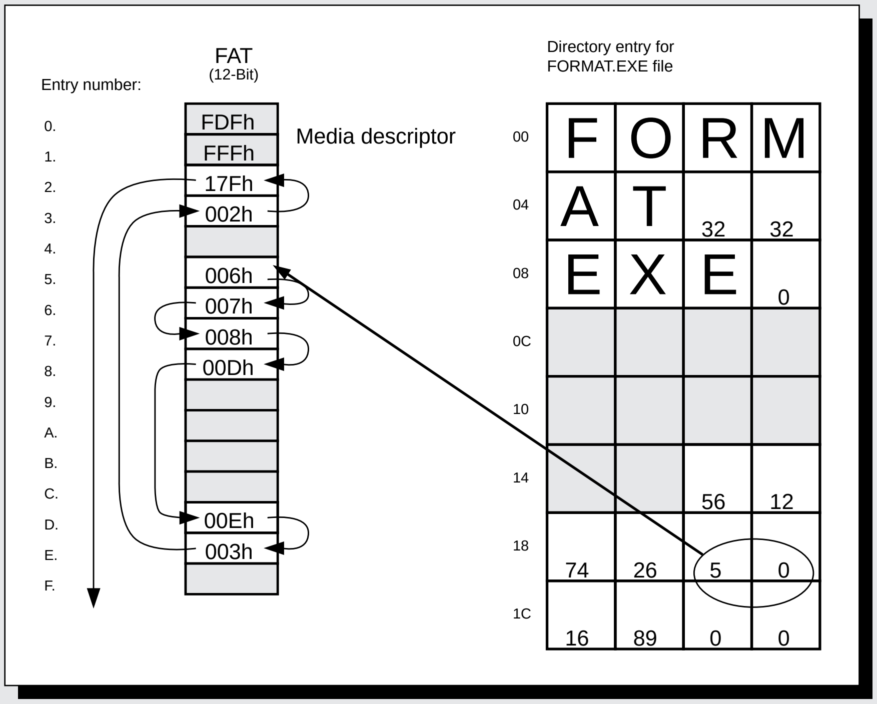
Perhaps you're wondering why the individual entries of the FAT are 12 or 16 bits wide if all they do is show whether a cluster is occupied. This could have been done with one bit; the bit could contain 1 when the cluster is occupied and 0 if the cluster is available.

The entries in the FAT help mark the available clusters and identify the individual clusters containing a specific file. The directory entry of a file tells DOS which cluster holds the first sectors of a file. The number of this cluster corresponds to the number of the FAT entry belonging to it. In this entry is the number of the cluster containing the next sector of file data. This search continues until the last cluster of a file has been reached.

However, the cluster numbers refer to the beginning of the data structure rather than the beginning of the volume (more on this later). For example, assume the beginning of a file resides in the fourth cluster of the volume. You must read the fourth entry of the FAT to obtain the number of the next cluster containing the file, remembering that a FAT entry can be either 12-bit or 16-bit. This fourth entry contains the number of the next file cluster.

As the following illustration shows, a chain is formed in which the individual clusters assigned to a file can be located in the proper sequence. The logical number of each sector can be derived from the cluster number. DOS must send this logical number to the device driver before these sectors can be read or written by DOS. DOS device drivers operate at sector level rather than cluster level. To convert clusters to logical sectors, multiply the cluster number by the number of sectors per cluster.

FAT entry and file clusters



The FAT entry corresponding to the last cluster of a file must contain a special code that tells DOS the file ends here. A cluster number indicates this code, which is either greater than FF8H (12-bit) or greater than FFF8H (16-bit). However, the following tables show the meanings of the various FAT entries. For example, cluster numbers FF0H-FF6H (12-bit) or FFF0H-FFF6H (16-bit) indicate a reserved cluster (i.e., the root directory of a volume). The root directory has a fixed position and size, and the cluster can be independent rather than chained to the FAT.

Cluster codes FF7H (12-bit) and FFF7H (16-bit) also have special significance. These codes identify clusters whose sectors contain errors, ensuring that DOS doesn't write data to these bad clusters.

Cluster code 0H indicates unoccupied clusters, as well as the first cluster of the volume. Like the first cluster, code 0H represents no sectors, because it contains the volume's medium type. As you'll see later in this book, this is why the cluster number must be reduced by two when converting cluster numbers into sector numbers.

12-bit FAT cluster codes		16-bit FAT cluster codes	
Code	Meaning	Code	Meaning
000H	Cluster is available	0000H	Cluster is available
FF0H-FF6H	reserved cluster	FFF0H-FFF6H	Reserved cluster
FF7H	Cluster damaged, not used	FFF7H	Cluster damaged, not used
FF8H-FFFH	Last file cluster	FFF8H-FFFFH	Last file cluster
xxxH	Next file cluster	XXXXH	Next file cluster

Multiple copies of the FAT

The FAT's importance in the DOS file system becomes obvious when a virus or hardware error damages the FAT. All files and subdirectories are still available in principle. However, they remain inaccessible to the user because DOS can no longer gather the individual clusters into one unit.

DOS is designed so several identical copies of the FAT can be kept on a volume. The boot sector's offset address 10H contains the number of FATs. If DOS finds a medium of this type, it automatically updates all copies of the FAT and records this update in offset 10H while creating or deleting files. So, if one FAT is damaged, it can be replaced with another, which minimizes data loss.

The DOS CHKDSK command tests the various FATs to see if they are identical. If the primary FAT is damaged, CHKDSK replaces the damaged primary FAT with another FAT.

The Root Directory

Now let's look at the structure of a directory. The root directory of a volume immediately follows the last copy of the FAT. This root directory (like all subdirectories) consists of 32-byte entries, in which information about individual files, subdirectories, and volume label names can be stored. The maximum number of entries in the root directory, and therefore its size, is stored in the BPB starting at address 11H. The FORMAT command specifies both the size number and the BPB. Before considering individual fields of this data structure, the table on the right provides an overview of a directory entry.

Directory entry layout		
Address	Contents	Type
00H	Filename (blanks padded with spaces)	8 bytes
08H	File extension (blanks padded with spaces)	3 bytes
0BH	File attribute	1 byte
0CH	Reserved	10 bytes
16H	Time of last change	1 word
18H	Date of last change	1 word
1AH	First cluster of file	1 word
1CH	File size	2 words

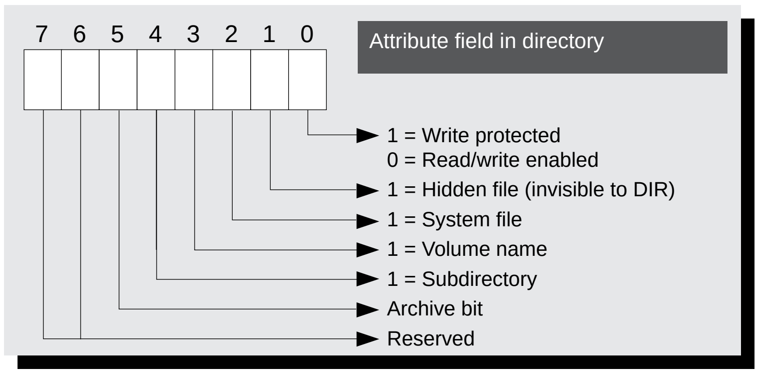
The first eight bytes usually contain the name of the current file. If the filename is shorter than eight characters, DOS fills the remaining characters with spaces (ASCII code 32).

If the directory entry doesn't contain information about a file, but the file is used in another way, the first byte of the filename (therefore the first byte of the directory entry) is identified by a special code (see the table on the right).

The first byte of the directory entry	
Code	Meaning
00H	Last directory entry
05H	First character of filename has ASCII code E5H
2EH	File applies to current directory
E5H	File deleted

The second field contains the three character filename extension. If the extension is less than three characters long, DOS fills in the extra characters with blank spaces (ASCII code 32). The period between filename and extension is displayed by the DOS command DIR but isn't kept in the directory; DIR displays this character so the names are easier to read.

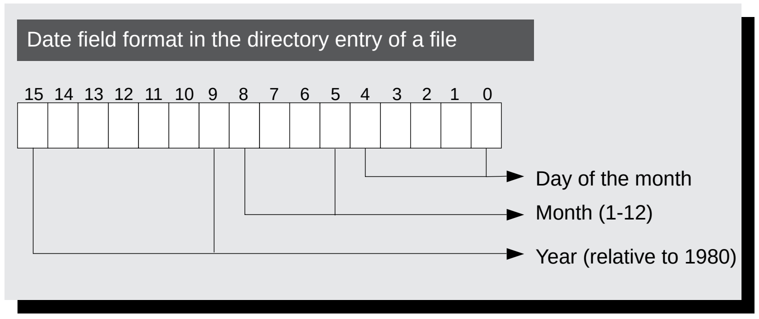
The one-byte attribute field is next. As shown in the following figure, the individual bits of this field define certain attributes. The various attributes can be combined so a file (as in the IBMBIOS.COM file) can have the attributes READ_ONLY, SYSTEM, and HIDDEN.

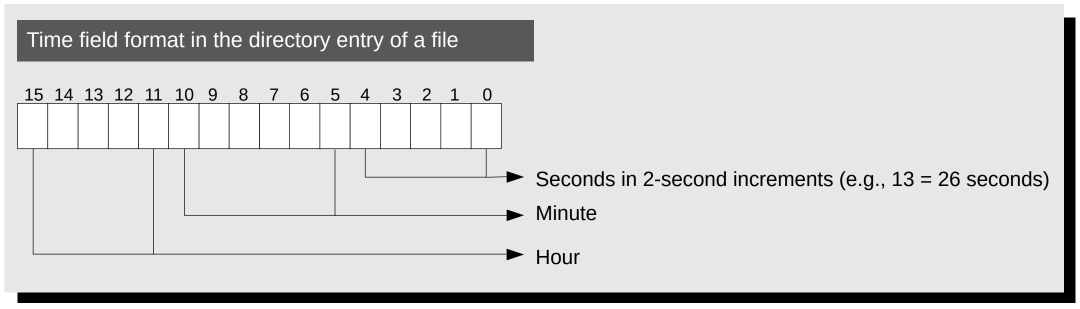


A reserved field follows the attribute field. DOS uses this field for internal operations, and some sources claim that Novell NetWare uses this bit for sharing data.

While the significance of bits 0 to 4 is easy to see, the significance of bit 5 needs additional explanation. The name archive bit comes from its use in making backup copies. Every time a file is created or modified, this bit is set to 1. If a program is used to backup this file, (for example the DOS BACKUP command), the archive bit is reset to 0. The next time the BACKUP command is used, it can determine, from the archive bit, whether this file has been modified since the last backup. If it still contains the value 0, the file doesn't have to be backed up again. If the archive bit contains a 1, the file was modified and should be backed up again.

The attributes volume label name and subdirectory will be discussed in more detail later. A reserved field, which DOS requires for internal operations, follows the attribute field. The time and date fields indicate when the file was last created or modified. Both are stored as words (2 bytes), but have special and different formats.




Time field format in the directory entry of a file
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Seconds in 2-second increments (e.g., 13 = 26 seconds)
Minute
Hour

The next field shows the number of the cluster that contains the first data of the file. It also shows the number of the FAT containing the number of the next cluster assigned to the file. This field forms the beginning of a chain through which all the clusters assigned to a file can be retrieved.

The file size in bytes is stored in two words with the lower word stored first. Using a small formula and the two words, the file size can be calculated as follows:

```
File size = word1 + word2 * 65,536
```

Subdirectories

Both subdirectories and volume label names deserve special consideration. The volume label name can exist only in the root directory. It's indicated by bit 3 of the current directory entry's attribute field. The filename in a volume entry acts as the volume label name. Use the DOS commands DIR, VOL, and TREE to display the volume label name.

If bit 4 of the current directory's attribute field is set, then this entry is for a subdirectory. If bit 1 in this field is also set, the subdirectory can be addressed. However, this subdirectory isn't displayed when you execute the DIR command. For these entries, the filename and extension field contain the subdirectory name; the date and time field contain the time of its creation. The file length field is always 0. The field that usually indicates the first cluster of the file now indicates the cluster that contains the directory entries of this subdirectory. They have the same 32-byte structure as the entries in the root directory.

As in a normal file, the entry in the FAT corresponding to the subdirectory cluster points to the next cluster of the subdirectory. This is true as long as one cluster is enough for the directory of the subdirectory. This doesn't apply to the root directory, which extends through several sectors or clusters that follow each other logically. Also, the individual clusters of the root directory cannot be connected through the FAT, because the FAT only refers to the data area of the volume. This is the area that accepts files and subdirectories, but not the root directory.

The process previously described reveals that DOS separates the individual files in a storage unit according to their directories. Instead of storing the files of one directory in one area, DOS scatters the files across the storage medium.

When a subdirectory is created, two files are created with the names '.' and '..'. These files can be erased only when you remove the entire subdirectory. The first file points to the current subdirectory. Its cluster field contains the number of the first cluster of the current subdirectory. The second entry points to the parent directory located before the current directory in the directory tree. If the parent directory is the root directory, the cluster field contains the value 0. The path to the root directory can be traced through this entry, since, as every subdirectory searches for its parent directory, it comes closer to the root directory.

Now let's return to our discussion of mass storage device structures. The file area follows the root directory. This area, which occupies the remaining storage area of the mass storage device, accepts the individual files and various subdirectories. There is an entry in the FAT corresponding to every cluster in this area. If a file is enlarged, DOS reserves a cluster that is still available to store the additional data of the file. The FAT entry of the last cluster, which previously indicated the end of the file, is changed to point to the new cluster. This in turn contains the new end character.

Both DOS Versions 1.0 and 2.0 search for unused clusters from the beginning. In DOS Versions 3.0 and higher, a more complicated search procedure is used to try to select an unused cluster near the other clusters comprising the file. This reduces the access time to the file. Conversely, when reducing file size or deleting a file, the FAT is updated to indicate the unused clusters are again available. They can be used again when a new file is created or expanded.

Let's begin at the point where DOS finds the first cluster or a file and the FAT entry for the next cluster in the first cluster. We need to calculate the sector from this cluster.

1. Subtract 2 from the cluster number. The first two FAT entries contain the media descriptor, so FAT entry 2 is the actual zero cluster on the volume.

2. Multiply the cluster by the number of sectors per cluster. BIOS obtains this information from the BIOS Parameter Block loaded from the boot sector. The result relates to the first sector of the volume, instead of the first sector of the

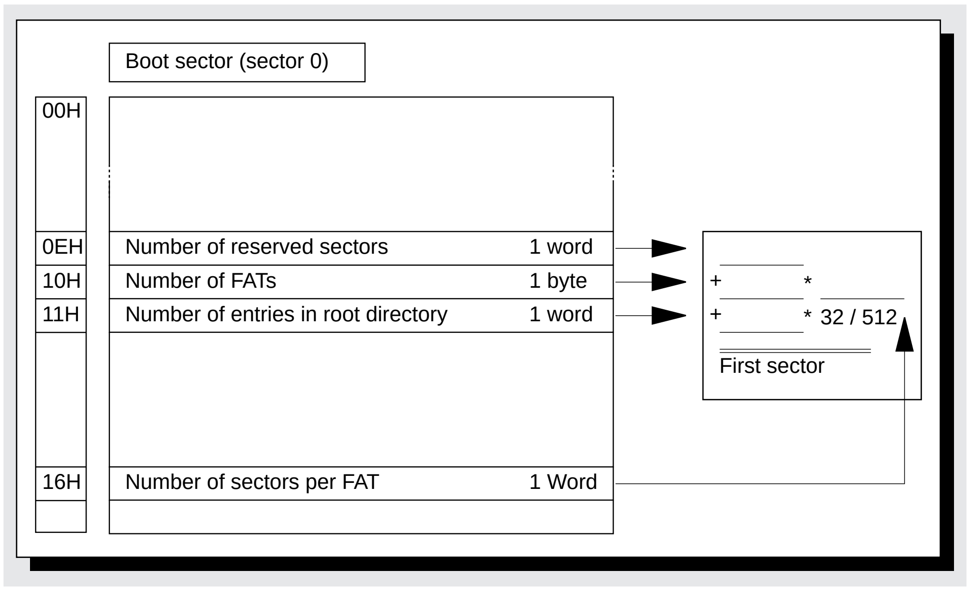
data range (which is the result we want). You can compute the logical sector where the data range starts, also using information from the BIOS Parameter Block.

3. The boot sector, the FAT and its duplicates, and the root directory precede the volume's data range. The lengths of these ranges must be calculated and added together. Read the number of sectors reserved from offset address 0EH of the boot sector. Read the number of FAT sectors from offset address 16H of the boot sector, and multiply the number of FAT sectors by the number of FATs (found in address 10H), then add the total to the number of reserved sectors.

4. DOS also requires the number of sectors occupied by the root directory. This number is stored in the word beginning at offset 11H in the boot sector. Multiply this value by 32 (bytes per entry), and divide the result by 512 (bytes per sector). Add the result to the boot sector and FAT lengths to obtain the number of the first sector in the data range.

DOS performs this calculation only when booting, or when a medium has been changed (e.g., every time you change diskettes in a floppy disk drive).

Calculating the first sector of the data range



By adding the number of the first sector in the data range to the first sector number of the addressed file, you receive the logical sector number. DOS can then pass this logical sector number to the device driver for file access.

When DOS requires access to both the first cluster and subsequent clusters of the file, it must read those subsequent clusters from the FAT. These calculations vary depending on the widths of the FAT entries. Reading the next adjacent cluster is easy to do with a 16-bit FAT. Simply multiply the cluster number by 2 to find the next cluster. The result represents the offset address relative to the beginning of the FAT in memory, where the next cluster can be found.

If DOS is dealing with a 12-bit FAT, multiply the cluster number by 1.5. The whole number part of the product becomes the offset in the FAT. The word at this memory address is read. If the product is a whole number, the word read must be combined with a logical AND to 0FFFH to obtain the number of the next cluster. If the product isn't a whole number, the AND is omitted and the word is shifted by 4 bits to the right (divided by 16).

DOS usually doesn't have to load the FAT into memory, because it permanently stores a copy of this data structure in memory to save execution time.

Now DOS knows the next cluster of the file. If you're using a 12-bit FAT, the next cluster lies in the range from FF8H to FFFH. If you're using a 16-bit FAT, the next cluster lies in the range from FFF8H to FFFFH. This process repeats until DOS finds the end of the file.