File Allocation Table

From Wikipedia, the free encyclopedia

File Allocation Table (**FAT**) is the name of a computer file system architecture and a family of industry standard file systems utilizing it.

The FAT file system is a legacy file system which is simple and robust.[4] It offers good performance even in light-weight implementations, but cannot deliver the same performance, reliability and scalability as some modern file systems. It is however supported for compatibility reasons by virtually all existing operating systems for personal computers, and thus is a well-suited format for data exchange between computers and devices of almost any type and age from the early 1980s up to the present.

Originally designed in the late 1970s for use on floppy disks, it was soon adapted and used almost universally on hard disks throughout the DOS and Windows 9x eras for two decades. With the introduction of more powerful computers and operating systems, and the development of more complex filesystems for them, it is no longer the default filesystem for usage on hard drives by most modern desktop operating systems.

Today, FAT file systems are still commonly found on floppy disks, solid-state memory cards, flash memory cards, and on many portable and embedded devices. It is also utilized in the boot stage of EFI-compliant computers.

The name of the file system originates from the file system's prominent usage of an index table, the *FAT*, statically allocated at the time of formatting. The table contains entries for each *cluster*, a contiguous area of disk storage. Each entry contains either the number of the next cluster in the file, or else a marker indicating end of file, unused disk space, or special reserved areas of the disk. The *root directory* of the disk contains the number of the first cluster of each file in that directory; the operating system can then traverse the FAT table, looking up the cluster number of each successive part of the disk file as a *cluster chain* until the end of the file is reached. In much the same way, *sub-directories* are implemented as special files containing the *directory entries* of their respective files.

FAT	
Developer	Microsoft, SCP, IBM, Compaq, Digital Research, Novell
Full name	File Allocation Table: FAT12 (12-bit version), FAT16/FAT16B/FAT16X (16-bit versions), FAT32/FAT32X (32-bit version with 28 bits used)
Introduced	1977 (Stand-alone Disk BASIC-80) FAT12: August 1980 (SCP QDOS) FAT16: August 1984 (IBM PC DOS 3.0) FAT16B: November 1987 (Compaq MS-DOS 3.31) FAT16X: August 1995 (Windows 95) FAT32/FAT32X: August 1996 (Windows 95 OSR2)
Partition identifier	MBR/EBR: FAT12: `0x01` H:`0x11`/`0x8D` S:`0xC1`/`0xD1` FAT16: `0x04` H:`0x14`/`0x90` S:`0xC4`/`0xD4` FAT16B: `0x06` H:`0x16`/`0x92` S:`0xC6`/`0xD6` FAT16X: `0x0E` H:`0x1E`/`0x9A` S:`0xCE` FAT32: `0x0B` H:`0x1B`/`0x97` S:`0xCB` FAT32X: `0x0C` H:`0x1C`/`0x98` S:`0xCC` Logical sectored FAT12/FAT16: `0x08` `0x11` `0x14` `0x24` `0x56` `0xE5` `0xF2` BDP: `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7`
Structures	
Directory contents	Table
File allocation	Linked list
Bad blocks	Cluster tagging
Limits	
Max. file size	4,294,967,295 bytes (4 GiB - 1) with FAT16B and FAT32[1] 274,877,906,943 bytes (256 GiB - 1) only with FAT32+[2]
Max. number of files	FAT12: 4,068 for 8 KiB clusters FAT16: 65,460 for 32 KiB clusters FAT32: 268,173,300 for 32 KiB clusters
Max. filename length	8.3 filename, or 255 UCS-2 characters when using LFN
Max. volume size	FAT12: 32 MiB (256 MiB for 64 KiB clusters) FAT16: 2 GiB (4 GiB for 64 KiB clusters) FAT32: 2 TiB (16 TiB for 4 KiB sectors)
Features	
Dates recorded	Modified date/time, creation date/time (DOS 7.0 and higher only), access date (only available with ACCDATE enabled),[3] deletion date/time (only with DELWATCH 2)
Date range	1980-01-01 to 2099-12-31 (2107-12-31)
Date resolution	2 seconds for last modified time, 10 ms for creation time, 1 day for access date, 2 seconds for deletion time
Forks	Not natively

As disk drives have evolved, the maximum number of clusters has significantly increased, and so the number of bits used to identify each cluster has grown. The successive major versions of the FAT format are named after the number of table element bits: 12 (FAT12), 16 (FAT16), and 32 (FAT32). Each of these variants is still in use. The FAT standard has also been expanded in other ways while generally preserving backward compatibility with existing software.

Attributes	Read-only, Hidden, System, Volume, Directory, Archive
File system permissions	FAT12/FAT16: File, directory and volume access rights for Read, Write, Execute, Delete only with DR-DOS, PalmDOS, Novell DOS, OpenDOS, FlexOS, 4680 OS, 4690 OS, Concurrent DOS, Multiuser DOS, System Manager, REAL/32 (Execute right only with FlexOS, 4680 OS, 4690 OS; individual file / directory passwords not with FlexOS, 4680 OS, 4690 OS; World/Group/Owner permission classes only with multiuser security loaded) FAT32: Partial, only with DR-DOS, REAL/32 and 4690 OS
Transparent compression	FAT12/FAT16: Per-volume, SuperStor, Stacker, DoubleSpace, DriveSpace FAT32: No
Transparent encryption	FAT12/FAT16: Per-volume only with DR-DOS FAT32: No

Contents

- 1 Overview
  - 1.1 Uses
  - 1.2 Nomenclature
- 2 Historical evolution
  - 2.1 File system types
    - 2.1.1 Original 8-bit FAT
    - 2.1.2 FAT12
    - 2.1.3 Initial FAT16
    - 2.1.4 Logical sectored FAT
    - 2.1.5 Final FAT16
    - 2.1.6 FAT32
  - 2.2 Extensions
    - 2.2.1 Extended Attributes
    - 2.2.2 Long file names
    - 2.2.3 Forks and Alternate Data Streams
    - 2.2.4 UMSDOS permissions and filenames
  - 2.3 Derivatives
    - 2.3.1 FATX
    - 2.3.2 exFAT
- 3 Technical design
  - 3.1 Layout
  - 3.2 Boot Sector
    - 3.2.1 BIOS Parameter Block
    - 3.2.2 Extended BIOS Parameter Block
    - 3.2.3 FAT32 Extended BIOS Parameter Block
    - 3.2.4 Exceptions
  - 3.3 FS Information Sector
  - 3.4 File Allocation Table
  - 3.5 Directory table
    - 3.5.1 Directory entry
    - 3.5.2 VFAT long file names
  - 3.6 Size limits
  - 3.7 Fragmentation
- 4 Legal issues
  - 4.1 Licensing
  - 4.2 Challenge
  - 4.3 Patent infringement lawsuits
- 5 See also
- 6 Notes
- 7 References
- 8 External links

Overview

Uses

The FAT file system offers reasonable good performance and robustness, even in very light-weight implementations.[4] It is therefore widely adopted and supported by virtually all existing operating systems for personal computers as well as some home computers and a multitude of embedded systems. This makes it a useful format for solid-state memory cards and a convenient way to share data between operating systems.

FAT file systems are the default file system for removable media (with the exception of CDs and DVDs) and as such are commonly found on floppy disks, super-floppies, memory and flash memory cards or USB flash drives and are supported by most portable devices such as PDAs, digital cameras, camcorders, media players, or mobile phones. While FAT12 is omnipresent on floppy disks, FAT16 and FAT32 are typically found on the larger media.

FAT was also commonly used on hard disks throughout the DOS and Windows 9x eras, but its use on hard drives has declined since the introduction of Windows XP, which primarily uses the newer NTFS. FAT is still used in hard drives expected to be used by multiple operating systems, such as in shared Windows, Linux and DOS environments.

Due to the widespread use of FAT-formatted media, many operating systems provide support for FAT through official or third-party file system handlers. For example, Linux, FreeBSD, BeOS and JNode provide inbuilt support for FAT, even though they also support more sophisticated file systems such as ext4 or btrfs. Mac OS 9 and Mac OS X support FAT file systems on volumes other than the boot disk. AmigaOS supports FAT through the CrossDOS package.

For many purposes, the NTFS file system is superior to FAT in terms of features and reliability; its main drawbacks are the size overhead for small volumes and the very limited support by anything other than the NT-based versions of Windows, since the exact specification is a trade secret of Microsoft. The availability of NTFS-3G since mid-2006 has led to much improved NTFS support in Unix-like operating systems, considerably alleviating this concern. It is still not possible to use NTFS in DOS-like operating systems without third-party drivers, which in turn makes it difficult to use a DOS floppy for recovery purposes. Microsoft provided a recovery console to work around this issue, but for security reasons it severely limited what could be done through the Recovery Console by default. The movement of recovery utilities to boot CDs based on BartPE, Linux (with NTFS-3G), or WinPE is finally eroding this drawback.

FAT has a long history (over three decades) of usage on desktops and portable computers, and it is frequently used in embedded solutions. As such, it continues to be the most widespread file system worldwide.

It is also utilized internally for the EFI system partition (partition type `0xEF`) in the boot stage of EFI-compliant computers.

For floppy disks, FAT has been standardized as ECMA-107[5] and ISO/IEC 9293:1994[6] (superseding ISO 9293:1987[7]). These standards cover FAT12 and FAT16 with only short 8.3 filename support; long filenames with VFAT are partially patented.[8]

Nomenclature

Technically, the term "FAT file system" refers to all three major variants of the file system, FAT12, FAT16 and FAT32, and most parties clearly distinguish between them where necessary. In contrast to this, Microsoft typically no longer distinguishes between all three of them since the introduction of FAT32, and refers to both FAT12 and FAT16 as "FAT", whereas "FAT32" gets treated specially in dialog boxes and documentation. This can sometimes lead to confusion if the actual type of the file system used is not mentioned or cannot be explicitly specified (e.g., "Do you want to format as FAT or FAT32?" instead of "Do you want to format as FAT12, FAT16 or FAT32?").

Another common cause of confusion exists within the group of FAT16 file systems, since the term "FAT16" refers to both, either the whole group of FAT file systems with 16-bit wide cluster entries, or specifically only the original implementation of it with 16-bit sector entries, when it becomes necessary to differentiate between the original and the later implementation. While technically the newer variant with 32-bit sector entries is called "FAT16B", it is commonly referred to under the name "FAT16" as well, in particular since the original variant is rarely seen today and typically only used on small media when backward compatibility with DOS before 3.31 is required.

Further, the term "VFAT" has led to various misconceptions as well, as it is sometimes erroneously used as if it would describe another variant of FAT file system to be distinguished from the FAT12, FAT16 and FAT32 file systems, while in reality it does not specify another file system, but an optional extension, which can work on top of any FAT file system, FAT12, FAT16 or FAT32. Volumes utilizing VFAT long-filenames can be read also by operating systems not supporting the VFAT extension, for as long as they support the underlying file system.

In order to be technically correct and exact, this article uses standard prefixes for the byte unit: 1000 bytes (10^3 bytes) are 1 kB (kilobyte) and 1024 bytes (2^{10} bytes) equal 1 KiB (kibibyte) etc.

Historical evolution

See also: Timeline of DOS operating systems

File system types

Original 8-bit FAT

The original FAT file system (or *FAT structure*, as it was called initially) was designed and coded by Marc McDonald,[9] based on a series of discussions between McDonald and Bill Gates.[9] It was introduced with 8-bit table elements[9] (and valid data cluster numbers `0x02` to `0xBF`) in Microsoft's *Stand-alone Disk BASIC-80* for a 8080-based successor[nb 1] of the NCR 7200 data-entry terminal with 8-inch (200 mm) floppy disks in 1977.[10] In 1978, *Stand-alone Disk BASIC* was ported to the 8086 using an emulator on a DEC PDP-10,[11] since no real 8086 systems were available at this time. The FAT file system was also utilized in Microsoft's MDOS/MIDAS,[9] an operating system for 8080/Z80 platforms written by McDonald since 1979. The *Stand-alone Disk BASIC* version supported three FATs,[12] whereas this was a parameter for MIDAS. Reportedly, MIDAS was also prepared to support 10-bit, 12-bit and 16-bit FAT variants. While the size of directory entries was 16 bytes in *Stand-alone Disk BASIC*, MIDAS instead occupied 32 bytes per entry.

Tim Paterson of Seattle Computer Products (SCP) was first introduced to Microsoft's FAT structure when he helped Bob O'Rear adapting the *Stand-alone Disk BASIC-86* emulator port onto SCP's S-100 bus 8086 CPU board prototype during a guest week at Microsoft in May 1979.[11] The final product was shown at Lifeboat Associates' booth stand at the National Computer Conference in New York[11] on 4–7 June 1979, where Paterson learned about the more sophisticated FAT implementation in MDOS/MIDAS[9] and McDonald talked to him about the design of the file system.[10]

FAT12

Between April and August 1980, while borrowing the FAT concept for SCP's own 8086 operating system QDOS 0.10,[11] Tim Paterson extended the table elements to 12 bits,[13] reduced the number of FATs to two, redefined the semantics of some of the reserved cluster values, and modified the disk layout, so that the root directory was now located between the FAT and the data area for his implementation of **FAT12**, whereas the format used in Microsoft *Stand-alone Disk BASIC's* 8-bit file system precursor was not supported by QDOS. By August 1980, QDOS had been renamed into 86-DOS already.[14] Starting with 86-DOS 0.42, the size and layout of directory entries was changed from 16 bytes to 32 bytes[15] in order to add a file date stamp[15] and increase the theoretical file size limit beyond the previous limit of 16 MiB.[15] 86-DOS 1.00 became available in early 1981. Later in 1981, 86-DOS evolved into Microsoft's MS-DOS and IBM PC DOS.[9][13][16] 86-DOS retained the capability to read previously formatted volumes with 16-byte directory entries[15] up to at least SCP MS-DOS 1.25.

Originally designed as a file system for floppy disks, FAT12 used 12-bit entries for the cluster addresses in the FAT, which not only limited the maximum generally possible count of data clusters to 4078 (for data clusters `0x002` to `0xFEF`)[17][18] or in some controlled scenarios even up to 4084 (for data clusters `0x002` to `0xFF5`),[5][19][20] but made FAT manipulation tricky with the PC's 8-bit and 16-bit registers. (While MS-DOS and PC DOS support up to 4084 data clusters on FAT12 volumes in general, cluster value `0xFF0`[nb 2] is treated as additional end-of-chain marker on any FAT12 volume[21] since MS-DOS/PC DOS 3.3, which also introduced the `0xF0` media descriptor value, therefore restricting the maximum practical number of data clusters to 4078 for compatibility purposes with these operating systems.)

The disk's size was stored and calculated as a 16-bit count of sectors, which limited the size to 32 MiB for a logical sector size of 512 bytes. FAT12 was used by several manufacturers with different physical formats, but a typical floppy disk at the time was 5.25-inch (130 mm), single-sided, 40 tracks, with 8 sectors per track, resulting in a capacity of 160 KiB for both the system areas and files. The FAT12 limitations exceeded this capacity by a factor of ten or more. (NB. The 32 MiB limit was later circumvented using logical sectored FATs with logical sector sizes larger than 512 bytes in some OEM versions of MS-DOS 3.x, but this fell into disuse when FAT16B became available with DOS 3.31, which supported 32-bit sector numbers and thereby further lifted the limits.)

By convention, all the control structures were organized to fit inside the first track, thus avoiding head movement during read and write operations, although this varied depending on the manufacturer and physical format of the disk. A limitation which was not addressed until much later (with FAT32) was that any bad sector in the control structures area, track 0, could prevent the disk from being usable. The DOS formatting tool rejected such disks completely. Bad sectors were allowed only in the file data area and were marked with the reserved value `0xFF7` in the FAT. They made the entire containing cluster unusable.

While 86-DOS supported three disk formats (250.25 KiB, 616 KiB and 1232 KiB with FAT IDs `0xFF` and `0xFE`) on 8-inch (200 mm) floppy drives, IBM PC DOS 1.0, released with the original IBM Personal Computer in 1981, supported only an 8-sector floppy format with a formatted capacity of 160 KiB (FAT ID `0xFE`) for single-sided 5.25-inch floppy drives, and PC DOS 1.1 added support for a double-sided format with 320 KiB (FAT ID `0xFF`). PC DOS 2.0 introduced support for 9-sector floppy formats with 180 KiB (FAT ID `0xFC`) and 360 KiB (FAT ID `0xFD`).

86-DOS 1.00 and PC DOS 1.0 directory entries included only one date, the last modified date. PC DOS 1.1 added the last modified time. PC DOS 1.x file attributes included a hidden bit and system bit, with the remaining six bits undefined. At this time, DOS did not support a hierarchical file system, which was still acceptable, given that the number of files on a disk was typically not more than a few dozen.

The PC XT was the first PC with a hard drive from IBM, and PC DOS 2.0 supported that hard drive with FAT12 (FAT ID `0xF8`). The fixed assumption of 8 sectors per clusters on hard disks practically limited the maximum partition size to 16 MiB for 512 byte sectors and 4 KiB clusters.

The *BIOS Parameter Block* (*BPB*) was introduced with PC DOS 2.0 as well, and this version also added read-only, archive, volume label, and directory attribute bits for hierarchical sub-directories.[22]

MS-DOS 3.0 introduced support for high-density 1.2 MiB 5.25-inch diskettes (media descriptor `0xF9`), which notably had 15 sectors per track, hence more space for the FATs.

FAT12 remains in use on all common floppy disks, including 1.44 MiB and later 2.88 MiB disks (media descriptor byte `0xF0`).

Initial FAT16

On 14 August 1984, IBM released the PC AT, which featured a 20 MiB hard disk and PC DOS 3.0.[23][24] Microsoft introduced MS-DOS 3.0 in parallel. Cluster addresses were increased to 16-bit, allowing for up to 65,524 clusters per volume, and consequently much greater file system sizes, at least in theory. However, the maximum possible number of sectors and the maximum (partition, rather than disk) size of 32 MiB did not change. Therefore, although cluster addresses were 16 bits, this format was not what today is commonly understood as **FAT16**. A partition type `0x04` indicates this form of FAT16 with less than 65536 sectors (less than 32 MiB for sector size 512).

With the initial implementation of FAT16 not actually providing for larger partition sizes than FAT12, the early benefit of FAT16 was to enable the use of smaller clusters, making disk usage more efficient, particularly for large numbers of files only a few hundred bytes in size, which were far more common at the time.

MS-DOS 2.x hard disks larger than 15 MiB are incompatible with later versions of MS-DOS.[25] A 20 MiB hard disk formatted under MS-DOS 3.0 was not accessible by the older MS-DOS 2.0 because MS-DOS 2.0 did not support version 3.0's FAT16. MS-DOS 3.0 could still access MS-DOS 2.0 style 8 KiB-cluster partitions under 15 MiB.

Logical sectored FAT

When hard disks grew larger and the FAT12 and FAT16 file system implementation in MS-DOS / PC DOS did not provide means to take advantage of the extra storage, several manufacturers developed their own FAT variants to address the problem in their MS-DOS OEM issues.

Some vendors (AST and NEC) supported eight, instead of the standard four, primary partition entries in their custom extended *Master Boot Record* (*MBR*), and they adapted MS-DOS to use more than a single primary partition.

Other vendors worked around the volume size limits imposed by the 16-bit sector entries and arithmetics by increasing the *size* of the sectors the file system dealt with, thereby blowing up dimensions.

These so called *logical sectors* were larger (up to 8192 bytes) than the *physical sector* size (still typically 512 bytes) as expected by the ROM-BIOS INT 13h or the disk drive hardware. The DOS-BIOS or System BIOS would then combine multiple physical sectors into logical sectors for the file system to work with. These changes were transparent to the file system implementation in the DOS kernel, since on the file system's abstraction level volumes are seen as a linear series of logically addressable sectors, also known as *absolute sectors* (addressed by their *Logical Sector Number* (*LSN*), starting with LSN 0) independent of the physical location of the volume on the physical medium and its geometry. The underlying DOS-BIOS translated these logical sectors into physical sectors according to partitioning information and the drive's physical geometry.

The drawback of this approach was a less memory-efficient sector buffering and deblocking in the DOS-BIOS, thereby causing an increased memory footprint for the DOS data structures. Since older DOS versions were not flexible enough to work with these logical geometries, the OEMs had to introduce new partition IDs for their FAT variants in order to hide them from off-the-shelf issues of MS-DOS and PC DOS. Known partition IDs for logical sectored FATs include: `0x08` (Commodore MS-DOS 3.x), `0x11` (Leading Edge MS-DOS 3.x), `0x14` (AST MS-DOS 3.x), `0x24` (NEC MS-DOS 3.30), `0x56` (AT&T MS-DOS 3.x), `0xE5` (Tandy MS-DOS), `0xF2` (Sperry IT MS-DOS 3.x, Unisys MS-DOS 3.3 — also used by Digital Research DOS Plus 2.1).[26] OEM versions like Toshiba MS-DOS, Wyse MS-DOS 3.2 and 3.3,[27] as well as Zenith MS-DOS are also known to have utilized logical sectoring.[28]

While non-standard and sub-optimal, these FAT variants are perfectly valid according to the specifications of the file system itself. Therefore, even if default issues of MS-DOS and PC DOS were not able to cope with them, most of these vendor-specific FAT12 and FAT16 variants can be mounted by more flexible file system implementations in operating systems such as DR-DOS, simply by changing the partition ID to one of the recognized types.[nb 3] Also, if they no longer need to be recognized by their original operating systems, existing partitions can be "converted" into FAT12 and FAT16 volumes more compliant with versions of MS-DOS/PC DOS 4.0-6.3, which do not support sector sizes different from 512 bytes,[29] by switching to a BPB with 32-bit entry for the number of sectors, as introduced since DOS 3.31 (see FAT16B below), keeping the cluster size and reducing the logical sector size in the BPB down to 512 bytes, while at the same time increasing the counts of logical sectors per cluster, reserved logical sectors, total logical sectors, and logical sectors per FAT by the same factor.

A parallel development in MS-DOS / PC DOS which allowed an increase in the maximum possible FAT size was the introduction of multiple FAT partitions on a hard disk. To allow the use of more FAT partitions in a compatible way, a new partition type was introduced in PC DOS 3.2 (1986), the *extended partition* (EBR),[9] which is a container for an additional partition called *logical drive*. Since PC DOS 3.3 (April 1987), there is another, optional extended partition containing the next *logical drive*, and so on. The MBR of a hard disk can either define up to four primary partitions, or an extended partition in addition to up to three primary partitions.

See also: Extended boot record

Final FAT16

Finally in November 1987, Compaq MS-DOS 3.31 (a modified OEM version of MS-DOS 3.3 released by Compaq with their machines) introduced what today is simply known as *the FAT16* format, with the expansion of the 16-bit disk sector count to 32 bits in the BPB. Although the on-disk changes were minor, the entire DOS disk driver had to be converted to use 32-bit sector numbers, a task complicated by the fact that it was written in 16-bit assembly language. The result was initially called the *DOS 3.31 Large File System*. Microsoft's DSKPROBE tool refers to type `0x06` as *BigFAT*,[30] whereas some older versions of FDISK described it as *BIGDOS*. Technically, it is known as **FAT16B**.

Since older versions of DOS were not designed to cope with more than 65535 sectors, it was necessary to introduce a new partition type for this format in order to hide it from pre-3.31 issues of DOS. The original form of FAT16 (with less than 65536 sectors) had a partition type `0x04`. To deal with disks larger than this, type `0x06` was introduced to indicate 65536 or more sectors. In addition to this, the disk driver was expanded to cope with more than 65535 sectors as well. The only other difference between the original FAT16 and the newer FAT16B format is the usage of a newer BPB format with 32-bit sector entry. Therefore, newer operating systems supporting the FAT16B format can cope also with the original FAT16 format without any necessary changes.

If partitions to be used by pre-DOS 3.31 issues of DOS need to be created by modern tools, the only criteria theoretically necessary to meet are a sector count of less than 65536, and the usage of the old partition ID (`0x04`). In practice however, type `0x01` and `0x04` primary partitions should not be physically located outside the first 32 MiB of the disk, due to other restrictions in MS-DOS 2.x, which could not cope with them otherwise.

In 1988, the FAT16B improvement became more generally available through DR DOS 3.31, PC DOS 4.0, OS/2 1.1, and MS-DOS 4.0. The limit on partition size was dictated by the 8-bit signed count of sectors per cluster, which originally had a maximum power-of-two value of 64. With the standard hard disk sector size of 512 bytes, this gives a maximum of 32 KiB cluster size, thereby fixing the "definitive" limit for the FAT16 partition size at 2 GiB for sector size 512. On magneto-optical media, which can have 1 or 2 KiB sectors instead of 0.5 KiB, this size limit is proportionally larger.

Much later, Windows NT increased the maximum cluster size to 64 KiB, by considering the sectors-per-cluster count as unsigned. However, the resulting format was not compatible with any other FAT implementation of the time, and it generated greater internal fragmentation. Windows 98, SE and ME also supported reading and writing this variant, but its disk utilities did not work with it and some FCB services are not available for such volumes. This contributes to a confusing compatibility situation.

Prior to 1995, versions of DOS accessed the disk via CHS addressing only. When MS-DOS 7.0 / Windows 95 introduced LBA disk access, partitions could start being physically located outside the first ca. 8 GiB of this disk and thereby out of the reach of the traditional CHS addressing scheme. Partitions partially or fully located beyond the CHS barrier therefore had to be hidden from non-LBA-enabled operating systems by using the new partition type `0x0E` in the partition table instead. FAT16 partitions using this partition type are also named **FAT16X**.[31] The only difference, compared to previous FAT16 partitions, is the fact that some CHS-related geometry entries in the BPB record, namely the number of sectors per track and the number of heads, may contain no or misleading values and should not be used.

The number of root directory entries available for FAT12 and FAT16 is determined when the volume is formatted, and is stored in a 16-bit field. For a given number `RDE` and sector size `SS`, the number `RDS` of root directory sectors is `RDS=ceil((RDE×32)/SS)`, and `RDE` is normally chosen to fill these sectors, i.e., `RDE*32=RDS*SS`. FAT12 and FAT16 media typically use 512 root directory entries on non-floppy media. Some third-party tools, like mkdosfs, allow the user to set this parameter.[32]

FAT32

In order to overcome the volume size limit of FAT16, while at the same time allowing DOS real mode code to handle the format, Microsoft designed a new version of the file system, **FAT32**, which supported an increased number of possible clusters, but could reuse most of the existing code, so that the available conventional memory footprint was reduced by less than 5 KiB under DOS.[33] Cluster values are represented by 32-bit numbers, of which 28 bits are used to hold the cluster number. The boot sector uses a 32-bit field for the sector count, limiting the FAT32 volume size to 2 TiB for a sector size of 512 bytes and 16 TiB for a sector size of 4,096 bytes.[34][35] FAT32 was introduced with MS-DOS 7.1 / Windows 95 OSR2 in 1996, although reformatting was needed to use it, and DriveSpace 3 (the version that came with Windows 95 OSR2 and Windows 98) never supported it. Windows 98 introduced a utility to convert existing hard disks from FAT16 to FAT32 without loss of data. In the Windows NT line, native support for FAT32 arrived in Windows 2000. A free FAT32 driver for Windows NT 4.0 was available from Winternals, a company later acquired by Microsoft. Since the acquisition the driver is no longer officially available. Since 1998, Caldera's dynamically loadable DRFAT32 driver could be used to enable FAT32 support in DR-DOS. The first version of DR-DOS to natively support FAT32 and LBA access was OEM DR-DOS 7.04 in 1999. That same year IMS introduced native FAT32 support with REAL/32 7.90, and IBM 4690 OS added FAT32 support with version 2.[36] Ahead Software provided another dynamically loadable FAT32.EXE driver for DR-DOS 7.03 with Nero Burning ROM in 2004. IBM PC DOS introduced native FAT32 support with OEM PC DOS 7.10 in 2003.

The maximum possible size for a file on a FAT32 volume is 4 GiB minus 1 byte or 4,294,967,295 ($2^{32} - 1$) bytes. This limit is a consequence of the file length entry in the directory table and would also affect huge FAT16 partitions with a sufficient sector size.[1] Video applications, large databases, and some other software easily exceed this limit.

The open FAT+[2] specification proposes how to store larger files up to 256 GiB minus 1 byte or 274,877,906,943 ($2^{38} - 1$) bytes on slightly modified and otherwise backwards compatible FAT32 volumes, but imposes a risk that disk tools or FAT32 implementations not aware of this extension may truncate or delete files exceeding the normal FAT32 file size limit. Also, support for **FAT32+** (and **FAT16+**) is limited to some versions of DR-DOS and not available in mainstream operating systems so far. (This extension is critically incompatible with the `/EAS` option of the FAT32.IFS method to store OS/2 extended attributes on FAT32 volumes.)

As with previous file systems, the design of the FAT32 file system does not include direct built-in support for long filenames, but FAT32 volumes can optionally hold VFAT long filenames in addition to short filenames in exactly the same way as VFAT long filenames have been optionally implemented for FAT12 and FAT16 volumes.

Two partition types have been reserved for FAT32 partitions, `0x0B` and `0x0C`. The latter type is also named **FAT32X** in order to indicate usage of LBA disk access instead of CHS. On such partitions, some CHS-related geometry entries in the EBPB record, namely the number of sectors per track and the number of heads, may contain no or misleading values and should not be used.

Extensions

Extended Attributes

OS/2 heavily depends on extended attributes (EAs) and stores them in a hidden file called "`EA␣DATA.␣SF`" in the root directory of the FAT12 or FAT16 volume. This file is indexed by two previously reserved bytes in the file's (or directory's) directory entry at offset `0x14`.[37] In the FAT32 format, these bytes hold the upper 16 bits of the starting cluster number of the file or directory, hence making it impossible to store OS/2 EAs on FAT32 using this method.

However, the third-party FAT32 installable file system (IFS) driver FAT32.IFS version 0.70 and higher by Henk Kelder & Netlabs for OS/2 and eComStation stores extended attributes in extra files with filenames having the string "`␣EA␣SF`" appended to the regular filename of the file to which they belong. The driver also utilizes the byte at offset `0x0C` in directory entries to store a special mark byte indicating the presence of extended attributes to help speed up things.[38][39] (This extension is critically incompatible with the FAT32+ method to store files larger than 4 GiB minus 1 on FAT32 volumes.[2])

Extended attributes are accessible via the Workplace Shell desktop, through REXX scripts, and many system GUI and command-line utilities (such as 4OS2).[40]

To accommodate its OS/2 subsystem, Windows NT supports the handling of extended attributes in HPFS, NTFS, FAT12 and FAT16. It stores EAs on FAT12, FAT16 and HPFS using exactly the same scheme as OS/2, but does not support any other kind of ADS as held on NTFS volumes. Trying to copy a file with any ADS other than EAs from an NTFS volume to a FAT or HPFS volume gives a warning message with the names of the ADSs that will be lost. It does not support the FAT32.IFS method to store EAs on FAT32 volumes.

Windows 2000 onward acts exactly as Windows NT, except that it ignores EAs when copying to FAT32 without any warning (but shows the warning for other ADSs, like "Macintosh Finder Info" and "Macintosh Resource Fork").

Cygwin uses "`EA␣DATA.␣SF`" files as well.

Long file names

One of the user experience goals for the designers of Windows 95 was the ability to use long filenames (LFNs—up to 255 UCS-2 code units long), in addition to classic 8.3 filenames (SFNs). For backward and forward compatibility LFNs were implemented as an optional extension on top of the existing FAT file system structures using a workaround in the way directory entries are laid out.

This transparent method to store long file names in the existing FAT file systems without altering their data structures is usually known as **VFAT** (for "Virtual FAT") after the Windows 95 virtual device driver.[nb 4]

In Windows NT, support for VFAT long filenames started from version 3.5.

Non VFAT-enabled operating systems can still access the files under their short file name alias without restrictions, however, the associated long file names may get lost, when files with long file names are copied under non VFAT-aware operating systems.

OS/2 added long filename support to FAT using extended attributes (EA) before the introduction of VFAT; thus, VFAT long filenames are invisible to OS/2, and EA long filenames are invisible to Windows, therefore experienced users of both operating systems would have to manually rename the files.

In order to support Java applications, the FlexOS-based IBM 4690 OS version 2 introduced its own virtual file system (VFS) architecture to store long filenames in the FAT file system in a backwards compatible fashion. If enabled, the virtual filenames (VFN) are available under separate logical drive letters, whereas the real filenames (RFN) remain available under the original drive letters.[41]

Forks and Alternate Data Streams

The FAT file system itself is not designed for supporting Alternate Data Streams (ADS), but some operating systems that heavily depend on them have devised various methods for handling them in FAT drives. Such methods either store the additional information in extra files and directories (Mac OS), or give new semantics to previously unused fields of the FAT on-disk data structures (OS/2 and Windows NT).

Mac OS using PC Exchange stores its various dates, file attributes and long filenames in a hidden file called "FINDER.DAT", and resource forks (a common Mac OS ADS) in a subdirectory called "RESOURCE.FRK", in every directory where they are used. From PC Exchange 2.1 onwards, they store the Mac OS long filenames as standard FAT long filenames and convert FAT filenames longer than 31 characters to unique 31-character filenames, which can then be made visible to Macintosh applications.

Mac OS X stores resource forks and metadata (file attributes, other ADS) in a hidden file with a name constructed from the owner filename prefixed with "._", and Finder stores some folder and file metadata in a hidden file called ".DS_Store".

UMSDOS permissions and filenames

Further information: FAT filesystem and Linux

Early Linux distributions also supported a format known as UMSDOS, a FAT variant with Unix file attributes (such as long file name and access permissions) stored in a separate file called "--linux-.---". UMSDOS fell into disuse after VFAT was released and it is not enabled by default in Linux kernels from version 2.5.7 onwards.[42]

Derivatives

FATX

FATX is a family of file systems designed for Microsoft's Xbox video game console hard disk drives and memory cards,[43][44] introduced in 2001.

While resembling the same basic design ideas as FAT16 and FAT32, the **FATX16** and **FATX32** on-disk structures are simplified but fundamentally incompatible with normal FAT16 and FAT32 file systems, making it impossible for normal FAT file system drivers to mount such volumes.

The non-bootable superblock sector is 4 KiB in size and holds a 18 byte large BPB-like structure completely different from normal BPBs. Clusters are typically 16 KiB in size and there is only one copy of the FAT on the Xbox. Directory entries are 64 bytes in size instead of the normal 32 bytes. Files can have filenames up to 42 characters long using the OEM character set and be up to 4 GiB minus 1 byte in size. The on-disk timestamps hold creation, modification and access dates and times but differ from FAT: in FAT, the epoch is 1980; in FATX, the epoch is 2000.[*citation needed*] On the Xbox 360, the epoch is 1980.[45]

exFAT

Main article: exFAT

exFAT is an incompatible file system, that was introduced with Windows Embedded CE 6.0 in November 2006. It is loosely based on the File Allocation Table architecture, but proprietary and protected by patents.

The MBR partition type is 0x07 (the same as used for IFS, HPFS, NTFS, etc.). Logical geometry information located in the VBR is stored in a format not resembling any kind of BPB. exFAT is intended for use on flash drives (such as SDXC and Memory Stick XC), where FAT32 is otherwise used.

It offers several benefits over FAT32 including breaking the 4 GiB file size limit of FAT32 (only when not taking the FAT+[2] extension into account, which allows files larger than 4 GiB also on FAT32 volumes), being more space-efficient for files smaller than 64 KiB on large volumes and, compared to light-weight implementations of FAT32 in DOS and some embedded systems, it can offer faster seeks if more than a few thousand files are stored in a single sub-directory, whereas FAT32 is typically faster than exFAT for larger files as used on digital cameras, camcorders and media players or when flash cards are used mainly for archival purposes.

Storage devices formatted as exFAT cannot exchange data with equipment not supporting the format. Much equipment does not support exFAT, which requires acquisition of a commercial license from Microsoft,[46] which rules out its legal distribution as part of open source operating systems.

Even though the Windows 7 operating system has FAT32 support, the ability to use it as a hard drive formatting option has been artificially limited within the GUI by Microsoft, which only allows the user to choose between exFAT and NTFS. Users attempting a FAT32 format must execute the FORMAT command through a command prompt with administrator authority,[*citation needed*] using the /FS:FAT32 parameter.

Technical design

Layout

Overview of the order of structures in a FAT partition or disk

Contents	Boot Sector	FS Information Sector (FAT32 only)	More reserved sectors (optional)	File Allocation Table #1	File Allocation Table #2 ... (conditional)	Root Directory (FAT12/FAT16 only)	Data Region (for files and directories) ... (to end of partition or disk)
Size in sectors	(number of reserved sectors)			(number of FATs) * (sectors per FAT)		(number of root entries*32) / (bytes per sector)	(number of clusters) * (sectors per cluster)

A FAT file system is composed of four different sections:

- The *Reserved sectors*, located at the very beginning.

  The first reserved sector (logical sector 0) is the Boot Sector (aka *Volume Boot Record* (*VBR*)). It includes an area called the *BIOS Parameter Block* (with some basic file system information, in particular its type, and pointers to the location of the other sections) and usually contains the operating system's boot loader code.
  Important information from the Boot Sector is accessible through an operating system structure called the *Drive Parameter Block* (*DPB*) in DOS and OS/2.
  The total count of reserved sectors is indicated by a field inside the Boot Sector, and is usually 32 on FAT32 filesystems.[19]
  For FAT32 file systems, the reserved sectors include a *File System Information Sector* at logical sector 1 and a *Backup Boot Sector* at logical sector 6.
  While many other vendors have continued to utilize a single-sector setup (logical sector 0 only) for the bootstrap loader, Microsoft's boot sector code has grown to spawn over logical sectors 0 and 2 since the introduction of FAT32, with logical sector 0 depending on sub-routines in logical sector 2. The Backup Boot Sector area consists of three logical sectors 6, 7, and 8 as well. In some cases, Microsoft also uses sector 12 of the reserved sectors area for an extended boot loader.

- The *FAT Region*.

  This typically contains two copies (may vary) of the *File Allocation Table* for the sake of redundancy checking, although rarely used, even by disk repair utilities.
  These are maps of the Data Region, indicating which clusters are used by files and directories. In FAT12 and FAT16 they immediately follow the reserved sectors.
  Typically the extra copies are kept in tight synchronization on writes, and on reads they are only used when errors occur in the first FAT. In FAT32, it is possible to switch from the default behaviour and select a single FAT out of the available ones to be used for diagnosis purposes.
  The first two clusters (cluster 0 and 1) in the map contain special values.

- The *Root Directory Region*.

  This is a *Directory Table* that stores information about the files and directories located in the root directory. It is only used with FAT12 and FAT16, and imposes on the root directory a fixed maximum size which is pre-allocated at creation of this volume. FAT32 stores the root directory in the Data Region, along with files and other directories, allowing it to grow without such a constraint. Thus, for FAT32, the Data Region starts here.

- The *Data Region*.

  This is where the actual file and directory data is stored and takes up most of the partition. Traditionally, the unused parts of the data region are initialized with a filler value of `0xF6` as per the INT 1Eh's Disk Parameter Table (DPT) during format on IBM compatible machines, but also used on the Atari Portfolio. 8-inch CP/M floppies typically came pre-formatted with a value of `0xE5`;[12] by way of Digital Research this value was also used on Atari ST formatted floppies.[nb 5] Amstrad used `0xF4` instead. Some modern formatters wipe hard disks with a value of `0x00`, whereas a value of `0xFF`, the default value of a non-programmed flash block, is used on flash disks to reduce wear. The latter value is typically also used on ROM disks. (Some advanced formating tools allow to configure the format filler byte.[nb 6])

  The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT. Note however, that files are allocated in units of clusters, so if a 1 KiB file resides in a 32 KiB cluster, 31 KiB are wasted.

  FAT32 typically commences the Root Directory Table in cluster number 2: the first cluster of the Data Region.

FAT uses little-endian format for all entries in the header (except for, where explicitly mentioned, for some entries on Atari ST boot sectors) and the FAT(s). It is possible to allocate more FAT sectors than necessary for the number of clusters. The end of the last FAT sector can be unused if there are no corresponding clusters. The total number of sectors (as noted in the boot record) can be larger than the number of sectors used by data (clusters × sectors per cluster), FATs (number of FATs × sectors per FAT), and hidden sectors including the boot sector: this would result in unused sectors at the end of the volume. If a partition contains more sectors than the total number of sectors occupied by the file system it would also result in unused sectors at the end of the volume.

Boot Sector

On non-partitioned devices, such as floppy disks, the Boot Sector (VBR) is the first sector (logical sector 0 with physical CHS address 0/0/1 or LBA address 0). For partitioned devices such as hard drives, the first sector is the Master Boot Record defining partitions, while the first sector of partitions formatted with a FAT file system is again the Boot Sector.

Common structure of the first 11 bytes used by most FAT versions for IBM compatible x86-machines since DOS 2.0 are:

Byte Offset	Length (bytes)	Description
0x000	3	Jump instruction. If the boot sector has a valid signature residing in the last two bytes of the boot sector (tested by most boot loaders residing in the System BIOS or the MBR) and this volume is booted from, the prior boot loader will pass execution to this entry point with certain register values, and the jump instruction will then skip past the rest of the (non-executable) header. See Volume Boot Record. Since DOS 2.0, valid x86-bootable disks must start with either a short jump followed by a NOP (opstring sequence `0xEB 0x?? 0x90`[21][47] as seen since DOS 3.0[nb 7]—and on DOS 1.1[48][49]) or a near jump (`0xE9 0x?? 0x??`[21][47] as seen on DOS 2.x formatted disks). For backward compatibility MS-DOS, PC DOS and DR-DOS also accept a jump (`0x69 0x?? 0x??`)[21][47][50] on removable disks. On hard disks, DR DOS additionally accepts the swapped JMPS sequence starting with a NOP (`0x90 0xEB 0x??`), whereas MS-DOS/PC DOS do not. (See below for Atari ST compatibility.) The presence of one of these opstring patterns (in combination with a test for a valid media descriptor value at offset `0x015`) serves as indicator to DOS 3.3 and higher that some kind of BPB is present (although the exact size should not be determined from the jump target since some boot sectors contain private boot loader data following the BPB), while for DOS 1.x (and some DOS 3.0) volumes, they will have to fall back to the DOS 1.x method to detect the format via the media byte in the FAT (in logical sector 1).
0x003	8	OEM Name (padded with spaces `0x20`). This value determines in which system the disk was formatted. Although officially documented as free for OEM use, MS-DOS/PC DOS (since 3.1), Windows 95/98/SE/ME and OS/2 check this field to determine which other parts of the boot record can be relied upon and how to interpret them. Therefore, setting the OEM label to arbitrary or bogus values may cause MS-DOS, PC DOS and OS/2 to not recognize the volume properly and cause data corruption on writes.[51][52][53] Common examples are "`IBM␣␣3.3`", "`MSDOS5.0`", "`MSWIN4.1`", "`IBM␣␣7.1`", "`mkdosfs␣`", and "`FreeDOS␣`". Some vendors store licensing info or access keys in this entry. The Volume Tracker in Windows 95/98/SE/ME will overwrite the OEM label with "`?????IHC`" signatures (a left-over from "`␣OGACIHC`" for "Chicago") even on a seemingly read-only disk access (such as a `DIR A:`) if the medium is not write-protected. Given the dependency on certain values explained above, this may, depending on the actual BPB format and contents, cause MS-DOS/PC DOS, Windows 95/98/SE/ME and OS/2 to no longer recognize a medium and throw error messages despite the fact that the medium is not defective and can still be read without problems under other operating systems.[54] Some boot loaders make adjustments or refuse to pass control to a boot sector depending on certain values detected here (e.g., NEWLDR offset `0x018`).
0x00B	varies	*BIOS Parameter Block* (13, 19, 21 or 25 bytes), *Extended BIOS Parameter Block* (32 or 51 bytes) or *FAT32 Extended BIOS Parameter Block* (79 bytes); size and contents varies between operating systems and versions, see below
varies	varies	File system and operating system specific boot code; often starts immediately behind [E]BPB, but sometimes additional "private" boot loader data is stored between the end of the [E]BPB and the start of the boot code.
0x1FD	1	Physical drive number (only in DOS 3.2 to 3.31 boot sectors). With OS/2 1.0 and DOS 4.0, this entry moved to sector offset `0x024` (at offset `0x19` in the EBPB). Most Microsoft and IBM boot sectors maintain values of `0x00` at offset `0x1FC` and `0x1FD` ever since, although they are not part of the signature at `0x1FE`. If this belongs to a boot volume, the DR-DOS 7.07 enhanced MBR can be configured (see NEWLDR offset `0x014`) to dynamically update this entry to the DL value provided at boot time or the value stored in the partition table. This enables booting off alternative drives, even when the VBR code ignores the DL value.
0x1FE	2	Boot sector signature (`0x55 0xAA`).[19][nb 8] This signature indicates an IBM PC compatible boot code and is tested by most boot loaders residing in the System BIOS or the MBR before passing execution to the boot sector's boot code (but, e.g., not by the original IBM PC ROM-BIOS[55]). This signature does not indicate a particular file system or operating system. Since this signature is not present on all FAT-formatted disks (e.g., not on DOS 1.x[48][49] or non-x86-bootable FAT volumes), operating systems must not rely on this signature to be present when logging in volumes (old issues of MS-DOS/PC DOS prior to 3.3 checked this signature, but newer issues as well as DR-DOS do not). Formatting tools must not write this signature if the written boot sector does not contain at least an x86-compatible dummy boot loader stub; at minimum, it must halt the CPU in an endless loop (`0xF4 0xEB 0xFD`) or issue an INT 19h and RETF (`0xCD 0x19 0xCB`). These opstrings should not be used at sector offset `0x000`, however, because DOS tests for other opcodes as signatures. Many MSX-DOS 2 floppies use `0xEB 0xFE 0x90` at sector offset `0x000` to catch the CPU in a tight loop while maintaining an opcode pattern recognized by MS-DOS/PC DOS. This signature must be located at fixed sector offset `0x1FE` for sector sizes 512 or higher. If the physical sector size is larger, it may be repeated at the end of the physical sector. Atari STs will assume a disk to be Atari 68000 bootable if the checksum over the 256 big-endian words of the boot sector equals `0x1234`.[56][nb 9] If the boot loader code is IBM compatible, it is important to ensure that the checksum over the boot sector does not match this checksum by accident. If this would happen to be the case, changing an unused bit (e.g., before or after the boot code area) can be used to ensure this condition is not met. In rare cases, a reversed signature `0xAA 0x55` has been observed on disk images. This can be the result of a faulty implementation in the formating tool based on faulty documentation,[nb 8] but it may also indicate a swapped byte order of the disk image, which might have occurred in transfer between platforms using a different endianness. BPB values and FAT12, FAT16 and FAT32 file systems are meant to use little-endian representation only and there are no known implementations of variants using big-endian values instead.

FAT-formatted Atari ST floppies have a very similar boot sector layout:

Byte Offset	Length (bytes)	Description
0x000	2	Jump instruction. Original Atari ST boot sectors start with a 68000 BRA.S instruction (`0x60 0x??`). For compatibility with PC operating systems, Atari ST formatted disks since TOS 1.4 start with `0xE9 0x??` instead.
0x002	6	OEM Name (padded with spaces `0x20`), e.g., "`Loader`" (`0x4C 0x6F 0x61 0x64 0x65 0x72`) on volumes containing an Atari ST boot loader. See OEM Name precautions for PC formatted disks above. Note the different offset and length of this entry compared to the entry on PC formatted disks.
0x008	3	Disk serial number (default: `0x00 0x00 0x00`), used by Atari ST to detect a disk change. (Windows 9x Volume Tracker will always store "`IHC`" here on non-write-protected floppy disks; see above.) This value must be changed if the disk content is externally changed, otherwise Atari STs may not recognize the change on re-insertion. This entry overlaps the OEM Name field on PC formatted disks. For maximum compatibility, it may be necessary to match certain patterns here; see above.
0x00B	19	*DOS 3.0 BIOS Parameter Block* (little-endian format)
0x01E	varies	Private boot sector data (mixed big-endian and little-endian format)
varies	varies	File system and operating system specific Atari ST boot code. No assumptions must be made in regard to the load position of the code, which must be relocatable. If loading an operating system fails, the code can return to the Atari ST BIOS with a 68000 RTS (opcode `0x4E75` with big-endian byte sequence `0x4E 0x75`[nb 8]) instruction and all registers unaltered.
0x1FE	2	Checksum. The 16-bit checksum over the 256 big-endian words of the 512 bytes boot sector including this word must match the magic value `0x1234` in order to indicate an Atari ST 68000 executable boot sector code.[56] This checksum entry can be used to align the checksum accordingly.[nb 9] If the logical sector size is larger than 512 bytes, the remainder is not included in the checksum and is typically zero-filled.[56] Since some PC operating systems erroneously do not accept FAT formatted floppies if the `0x55 0xAA`[nb 8] signature is not present here, it is advisable to place the `0x55 0xAA` in this place (and add an IBM compatible boot loader or stub) and use an unused word in the private data or the boot code area or the serial number in order to ensure that the checksum `0x1234`[nb 9] is not matched (unless the shared fat code overlay would be both IBM PC and Atari ST executable at the same time).

FAT12-formatted MSX-DOS volumes have a very similar boot sector layout:

Byte Offset	Length (bytes)	Description
0x000	3	Dummy jump instruction (e.g., `0xEB 0xFE 0x90`).
0x003	8	OEM Name (padded with spaces `0x20`).
0x00B	19	*DOS 3.0 BPB*
0x01E	varies (2)	MSX-DOS 1 code entry point for Z80 processors into MSX boot code. This is where MSX-DOS 1 machines jump to when passing control to the boot sector. This location overlaps with BPB formats since DOS 3.2 or the x86 compatible boot sector code of IBM PC compatible boot sectors and will lead to a crash on the MSX machine unless special precautions have been taken such as catching the CPU in a tight loop here (opstring `0x18 0xFE` for JR `0x01E`).
0x020	6	MSX-DOS 2 volume signature "`VOL_ID`".
0x026	1	MSX-DOS 2 undelete flag (default: `0x00`. If the "`VOL_ID`" signature is present at sector offset `0x020`, this flag indicates, if the volume holds deleted files which can be undeleted (see offset `0x0C` in directory entries).
0x027	4	MSX-DOS 2 disk serial number (default: `0x00000000`). If the "`VOL_ID`" signature is present at sector offset `0x020`, MSX-DOS 2 stores a volume serial number here for media change detection.
0x02B	5	reserved
0x030	varies (2)	MSX-DOS 2 code entry point for Z80 processors into MSX boot code. This is where MSX-DOS 2 machines jump to when passing control to the boot sector. This location overlaps with EBPB formats since DOS 4.0 / OS/2 1.2 or the x86 compatible boot sector code of IBM PC compatible boot sectors and will lead to a crash on the MSX machine unless special precautions have been taken such as catching the CPU in a tight loop here (opstring `0x18 0xFE` for JR `0x030`).
0x1FE	2	Signature

BIOS Parameter Block

Common structure of the first 25 bytes of the BIOS Parameter Block (BPB) used by FAT versions since DOS 2.0 (bytes at sector offset `0x00B` to `0x017` are stored since DOS 2.0, but not always used before DOS 3.2, values at `0x018` to `0x01B` are used since DOS 3.0):

Sector Offset	BPB Offset	Length (bytes)	Description
0x00B	0x00	2	Bytes per logical sector in powers of two; the most common value is 512. Some operating systems don't support other sector sizes. For simplicity and maximum performance, the logical sector size is often identical to a disk's physical sector size, but can be larger or smaller in some scenarios. The minimum allowed value for non-bootable FAT12/FAT16 volumes with up to 65535 logical sectors is 32 bytes, or 64 bytes for more than 65535 logical sectors. The minimum practical value is 128. Some pre-DOS 3.31 OEM versions of DOS used logical sector sizes up to 8192 bytes for logical sectored FATs. Atari ST GEMDOS supports logical sector sizes between 512 and 4096.[56] DR-DOS supports booting off FAT12/FAT16 volumes with logical sector sizes up to 32 KiB and INT 13h implementations supporting physical sectors up to 1024 bytes/sector. The minimum logical sector size for standard FAT32 volumes is 512 bytes, which can be reduced downto 128 bytes without support for the FS Information Sector. Floppy drives and controllers use physical sector sizes of 128, 256, 512 and 1024 bytes (e.g., PC/AX). The Atari Portfolio supports a sector size of 512 for volumes larger than 64 KiB, 256 bytes for volumes larger 32 KiB and 128 bytes for smaller volumes. Magneto-optical drives used sector sizes of 512, 1024 and 2048 bytes. In 2005 some Seagate custom hard disks used sector sizes of 1024 bytes instead of the default 512 bytes.[57] Advanced Format hard disks use 4096 bytes per sector (*4Kn*) since 2010, but will also be able to emulate 512 byte sectors (*512e*) for a transitional period.
0x00D	0x02	1	Logical sectors per cluster. Allowed values are 1, 2, 4, 8, 16, 32, 64, and 128. Some MS-DOS 3.x versions supported a maximum cluster size of 4 KiB only, whereas modern MS-DOS/PC DOS and Windows 95 support a maximum cluster size of 32 KiB. Windows 98/SE/ME partially support a cluster size of 64 KiB as well, but some FCB services are not available on such disks and various applications fail to work. The Windows NT family and some alternative DOS versions such as PTS-DOS fully support 64 KiB clusters. For DOS-based operating systems, the maximum cluster size remains at 32 KiB or 64 KiB even for sector sizes larger than 512 bytes. MS-DOS/PC DOS will hang on startup if this value is erroneously specified as 0.[58]
0x00E	0x03	2	Count of reserved logical sectors. The number of logical sectors before the first FAT in the file system image. At least 1 for this sector, usually 32 for FAT32 (to hold the extended boot sector, FS info sector and backup boot sectors). Since DR-DOS 7.0x FAT32 formatted volumes use a single-sector boot sector, FS info sector and backup sector, some volumes formatted under DR-DOS use a value of 4 here.
			Number of File Allocation Tables. Almost always 2; RAM disks might use 1. Most versions of MS-DOS/PC DOS do not support more than 2 FATs. Some DOS operating systems support only two FATs in their built-in disk driver, but support other FAT counts for block device drivers loaded later on.

0x010	0x05	1	Volumes declaring 2 FATs in this entry will never be treated as TFAT volumes. If the value differs from 2, some Microsoft operating systems may attempt to mount the volume as a TFAT volume and use the second cluster (cluster 1) of the first FAT to determine the TFAT status.
0x011	0x06	2	Maximum number of FAT12 or FAT16 root directory entries. 0 for FAT32, where the root directory is stored in ordinary data clusters; see offset `0x02C` in FAT32 EBPBs. This value must be adjusted so that directory entries always consume full logical sectors, whereby each directory entry takes up 32 bytes. MS-DOS/PC DOS require this value to be a multiple of 16. The maximum value supported on floppy disks is 240, the maximum value supported by MS-DOS/PC DOS on hard disks is 512. DR-DOS supports booting off FAT12/FAT16 volumes, if the boot file is located in the first 2048 root directory entries.
0x013	0x08	2	Total logical sectors (if zero, use 4 byte value at offset `0x020`)
0x015	0x0A	1	Media descriptor (compare: FAT ID):[59][60][61][nb 7] 0xE5 - 8-inch (200 mm) Single sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (250.25 KiB) (DR-DOS only) 0xED - 5.25-inch (130 mm) Double sided, 80 tracks per side, 9 sector, 720 KiB (Tandy 2000 only)[53] 0xF0[5][6][7] - 3.5-inch (90 mm) Double Sided, 80 tracks per side, 18 or 36 sectors per track (1440 KiB, known as “1.44 MB”; or 2880 KiB, known as “2.88 MB”). - Designated for use with custom floppy and superfloppy formats where the geometry is defined in the BPB. - Used also for other media types such as tapes.[62] 0xF8 - Fixed disk (i.e., typically a partition on a hard disk). (since DOS 2.0)[9][63] - Designated to be used for any partitioned fixed or removable media, where the geometry is defined in the BPB. - 3.5-inch Single sided, 80 tracks per side, 9 sectors per track (360 KiB) (MS-DOS 3.1[47] and MSX-DOS) - 5.25-inch Double sided, 80 tracks per side, 9 sectors per track (720 KiB) (Sanyo 55x DS-DOS 2.11 only)[53] 0xF9[5][6][7] - 3.5-inch Double sided, 80 tracks per side, 9 sectors per track (720 KiB) (since DOS 3.2)[9] - 3.5-inch Double sided, 80 tracks per side, 18 sectors per track (1440 KiB) (DOS 3.2 only)[9] - 5.25-inch Double sided, 80 tracks per side, 15 sectors per track (1200 KiB, known as “1.2 MB”) (since DOS 3.0)[9] 0xFA - 3.5-inch and 5.25-inch Single sided, 80 tracks per side, 8 sectors per track (320 KiB) - Used also for RAM disks and ROM disks (e.g., on Columbia Data Products[64] and on HP 200LX) - Hard disk (Tandy MS-DOS only) 0xFB - 3.5-inch and 5.25-inch Double sided, 80 tracks per side, 8 sectors per track (640 KiB) 0xFC - 5.25-inch Single sided, 40 tracks per side, 9 sectors per track (180 KiB) (since DOS 2.0)[9] 0xFD - 5.25-inch Double sided, 40 tracks per side, 9 sectors per track (360 KiB) (since DOS 2.0)[9] - 8-inch Double sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (500.5 KiB) - (8-inch Double sided, (single and) double density (DOS 1)[9]) 0xFE - 5.25-inch Single sided, 40 tracks per side, 8 sectors per track (160 KiB) (since DOS 1.0)[9] - 8-inch Single sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (250.25 KiB) - 8-inch Double sided, 77 tracks per side, 8 sectors per track, 1024 bytes per sector (1232 KiB) - (8-inch Single sided, (single and) double density (DOS 1)[9]) 0xFF - 5.25-inch Double sided, 40 tracks per side, 8 sectors per track (320 KiB) (since DOS 1.1)[9] - Hard disk (Sanyo 55x DS-DOS 2.11 only)[53] This value must reflect the media descriptor stored (in the entry for cluster 0) in the first byte of each copy of the FAT. Certain operating systems before DOS 3.2 (86-DOS, MS-DOS/PC DOS 1.x and MSX-DOS version 1.0) ignore the boot sector parameters altogether and use the media descriptor value from the first byte of the FAT to choose among internally pre-defined parameter templates. Must be greater or equal to `0xF0` since DOS 4.0. On removable drives, DR-DOS will assume the presence of a BPB if this value is greater or equal to `0xF0`, whereas for fixed disks, it must be `0xF8` to assume the presence of a BPB. Initially, these values were meant to be used as bit flags; for any removable media without a recognized BPB format and a media descriptor of either `0xF8` or `0xFA` to `0xFF` MS-DOS/PC DOS treats bit 1 as a flag to choose a 9-sectors per track format rather than an 8-sectors format, and bit 0 as a flag to indiciate double-sided media.[47] Values `0x00` to `0xEF` and `0xF1` to `0xF7` are reserved and must not be used.
0x016	0x0B	2	Logical sectors per File Allocation Table for FAT12/FAT16, 0 for FAT32 (cf. offset `0x024` below)

DOS 3.0 BPB:

The following extensions were documented since DOS 3.0, however, they were already supported by some issues of DOS 2.13. MS-DOS 3.10 still supported the DOS 2.0 format, but could use the DOS 3.0 format as well.

Sector Offset	BPB Offset	Length (bytes)	Description
0x00B	0x00	13	*DOS 2.0 BPB*
0x018	0x0D	2	Physical sectors per track for disks with INT 13h CHS geometry,[19] e.g., 15 for a “1.20 MB” (1200 KiB) floppy. A zero entry indicates that this entry is reserved, but not used.
0x01A	0x0F	2	Number of heads for disks with INT 13h CHS geometry,[19] e.g., 2 for a double sided floppy. A bug in all versions of MS-DOS/PC DOS up to including 7.10 causes these operating systems to crash for CHS geometries with 256 heads, therefore almost all BIOSes choose a maximum of 255 heads only. A zero entry indicates that this entry is reserved, but not used.
0x01C	0x11	2	Count of hidden sectors preceding the partition that contains this FAT volume. This field should always be zero on media that are not partitioned. This DOS 3.0 entry is incompatible with a similar entry at offset `0x01C` in BPBs since DOS 3.31. It must not be used if the logical sectors entry at offset `0x013` is zero.

DOS 3.2 BPB:

Officially, MS-DOS 3.20 still used the DOS 3.0 format, but SYS and FORMAT were adapted to support a 6 bytes longer format already (of which not all entries were used).

Sector Offset	BPB Offset	Length (bytes)	Description
0x00B	0x00	19	*DOS 3.0 BPB*
0x01E	0x13	2	Total logical sectors including hidden sectors. This DOS 3.2 entry is incompatible with a similar entry at offset `0x020` in BPBs since DOS 3.31. It must not be used if the logical sectors entry at offset `0x013` is zero.

DOS 3.31 BPB:

Officially introduced with DOS 3.31 and not used by DOS 3.2, some DOS 3.2 utilities were designed to be aware of this new format already. Official documentation recommends to trust these values only if the logical sectors entry at offset `0x013` is zero.

Sector Offset	BPB Offset	Length (bytes)	Description
0x00B	0x00	13	*DOS 2.0 BPB*
0x018	0x0D	2	Physical sectors per track for disks with INT 13h CHS geometry,[19] e.g., 18 for a “1.44 MB” (1440 KiB) floppy. Unused for drives, which don't support CHS access any more. Identical to an entry available since DOS 3.0. A zero entry indicates that this entry is reserved, but not used. A value of 0 may indicate LBA-only access, but may cause a divide-by-zero exception in some boot loaders, which can be avoided by storing a neutral value of 1 here, if no CHS geometry can be reasonably emulated.
0x01A	0x0F	2	Number of heads for disks with INT 13h CHS geometry,[19] e.g., 2 for a double sided floppy. Unused for drives, which don't support CHS access any more. Identical to an entry available since DOS 3.0. A bug in all versions of MS-DOS/PC DOS up to including 7.10 causes these operating systems to crash for CHS geometries with 256 heads, therefore almost all BIOSes choose a maximum of 255 heads only. A zero entry indicates that this entry is reserved, but not used. A value of 0 may indicate LBA-only access, but may cause a divide-by-zero exception in some boot loaders, which can be avoided by storing a neutral value of 1 here, if no CHS geometry can be reasonably emulated.
0x01C	0x11	4	Count of hidden sectors preceding the partition that contains this FAT volume. This field should always be zero on media that are not partitioned.[5][6][7] This DOS 3.31 entry is incompatible with a similar entry at offset `0x01C` in DOS 3.0-3.3 BPBs. At least, it can be trusted if it holds zero, or if the logical sectors entry at offset `0x013` is zero. If this belongs to an Advanced Active Partition (AAP) selected at boot time, the BPB entry will be dynamically updated by the enhanced MBR to reflect the "relative sectors" value in the partition table, stored at offset `0x1B6` in the AAP or NEWLDR MBR, so that it becomes possible to boot the operating system from EBRs.
0x020	0x15	4	Total logical sectors (if greater than 65535; otherwise, see offset `0x013`). This DOS 3.31 entry is incompatible with a similar entry at offset `0x01E` in DOS 3.2-3.3 BPBs. Officially, it must be used only if the logical sectors entry at offset `0x013` is zero, but some operating systems (some old versions of DR DOS) use this entry also for smaller disks.

A simple formula translates a volume's given cluster number `CN` to a logical sector number `LSN`:[5][6][7]

1. Determine (once) `SSA=RSC+FN×SF+ceil((32×RDE)/SS)`, where the reserved sector count `RSC` is stored at offset `0x00E`, the FAT number `FN` at offset `0x010`, the sectors per FAT `SF` at offset `0x016` (FAT12/FAT16) or `0x024` (FAT32), the root directory entries `RDE` at offset `0x011`, the sector size `SS` at offset `0x00B`, and `ceil(x)` rounds up to a whole number.
2. Determine `LSN=SSA+(CN-2)×SC`, where the sectors per cluster `SC` are stored at offset `0x00D`.

On unpartitioned media the volume's number of hidden sectors is zero and therefore `LSN` and `LBA` addresses become the same for as long as a volume's logical sector size is identical to the underlying medium's physical sector size. Under these conditions, it is also simple to translate between `CHS` addresses and `LSN`s as well:

`LSN=SPT×(HN+(NOS×TN))+SN-1`, where the sectors per track `SPT` are stored at offset `0x018`, and the number of sides `NOS` at offset `0x01A`. Track number `TN`, head number `HN`, and sector number `SN` correspond to Cylinder-head-sector: the formula gives the known CHS to LBA translation.

Extended BIOS Parameter Block

Further structure used by FAT12 and FAT16 since OS/2 1.0 and DOS 4.0, also known as Extended BIOS Parameter Block (EBPB) (bytes below sector offset `0x024` are the same as for the DOS 3.31 BPB):

Sector Offset	EBPB Offset	Length (bytes)	Description
0x00B	0x00	25	*DOS 3.31 BPB*
0x024	0x19	1	Physical drive number (`0x00` for (first) removable media, `0x80` for (first) fixed disk as per INT 13h). Allowed values for possible physical drives depending on BIOS are `0x00-0x7E` and `0x80-0xFE`. Values `0x7F` and `0xFF` are reserved for internal purposes such as remote or ROM boot and should never occur on disk. Some boot loaders such as the MS-DOS/PC DOS boot loader use this value when loading the operating system, others ignore it altogether or use the drive number provided in the DL register by the underlying boot loader (e.g., with many BIOSes and MBRs). The entry is sometimes changed by SYS tools or it can be dynamically fixed up by the prior bootstrap loader in order to force the boot sector code to load the operating system from alternative physical disks than the default. A similar entry existed (only) in DOS 3.2 to 3.31 boot sectors at sector offset `0x1FD`. If this belongs to a boot volume, the DR-DOS 7.07 enhanced MBR can be configured (see NEWLDR offset `0x014`) to dynamically update this EBPB entry to the DL value provided at boot time or the value stored in the partition table. This enables booting off alternative drives, even when the VBR code ignores the DL value.
0x025	0x1A	1	Reserved; ▪ In some MS-DOS/PC DOS boot code used as a scratchpad for the INT 13h current head high byte for the assumed 16-bit word at offset `0x024`. Some DR-DOS FAT12/FAT16 boot sectors use this entry as a scratchpad as well, but for different purposes. ▪ VGACOPY stores a CRC over the system's ROM-BIOS in this location. ▪ Some boot managers use this entry to communicate the desired drive letter under which the volume should occur to operating systems such as OS/2 by setting bit 7 and specifying the drive number in bits 6-0 (C: = value 0, D: = value 1, ...). Since this normally affects the in-memory image of the boot sector only, this does not cause compatibility problems with other uses; ▪ In Windows NT used for CHKDSK flags (bits 7-2 always cleared, bit 1: disk I/O errors encountered, possible bad sectors, run surface scan on next boot, bit 0: volume is "dirty" and was not properly unmounted before shutdown, run CHKDSK on next boot).[63] Should be set to 0 by formating tools.[5][6][7] See also: Bitflags in the second cluster entry in the FAT.
0x026	0x1B	1	Extended boot signature. (Should be `0x29`[5][6][7][59] to indicate that an EBPB with the following 3 entries exists (since OS/2 1.2 and DOS 4.0). Can be `0x28` on some OS/2 1.0-1.1 and PC DOS 3.4 disks indicating an earlier form of the EBPB format with only the serial number following. MS-DOS/PC DOS 4.0 and higher, OS/2 1.2 and higher as well as the Windows NT family recognize both signatures accordingly.)
0x027	0x1C	4	Volume ID (serial number) Typically the serial number "xxxx-xxxx" is created by a 16-bit addition of both DX values returned by INT 21h/AH=2Ah (get system date)[nb 10] and INT 21h/AH=2Ch (get system time)[nb 10] for the high word and another 16-bit addition of both CX values for the low word of the serial number. Alternatively, some DR-DOS disk utilities provide a `/#` option to generate a human-readable time stamp "mmdd-hhmm" build from BCD-encoded 8-bit values for the month, day, hour and minute instead of a serial number.
0x02B	0x20	11	Partition Volume Label, padded with blanks (`0x20`), e.g., "`NO␣NAME␣␣␣␣`" Software changing the directory volume label in the file system should also update this entry, but not all software does. The partition volume label is typically displayed in partitioning tools since it is accessible without mounting the volume. Supported since OS/2 1.2 and MS-DOS 4.0 and higher. This area was used by boot sectors of DOS 3.2 to 3.3 to store a private copy of the Disk Parameter Table (DPT) instead of using the INT 1Eh pointer to retrieve the ROM table as in later issues of the boot sector. The re-usage of this location for the mostly cosmetical partition volume label minimized problems if some older system utilities would still attempt to patch the former DPT.
0x036	0x2B	8	File system type, padded with blanks (`0x20`), e.g., "`FAT12␣␣␣`", "`FAT16␣␣␣`", "`FAT␣␣␣␣␣`" This entry is meant for display purposes only and must not be used by the operating system to identify the type of the file system. Nevertheless, it is sometimes used for identification purposes by third-party software and therefore the values should not differ from those officially used. Supported since OS/2 1.2 and MS-DOS 4.0 and higher.

FAT32 Extended BIOS Parameter Block

In essence FAT32 inserts 28 bytes into the EBPB, followed by the remaining 26 EBPB bytes as shown above for FAT12 and FAT16. Microsoft and IBM operating systems determine the type of FAT filesystem used on a volume solely by the number of clusters, not by the used BPB format or the indicated file system type, that is, it is technically possibly to use a "FAT32 EBPB" also for FAT12 and FAT16 volumes as well as a DOS 4.0 EBPB for small FAT32 volumes. Since such volumes were found to be created by Windows operating systems under some odd conditions, operating systems should be prepared to cope with these hybrid forms.

Sector Offset	FAT32 EBPB Offset	Length (bytes)	Description
0x00B	0x00	25	*DOS 3.31 BPB*
0x024	0x19	4	Logical sectors per file allocation table (corresponds with this old entry). The byte at offset 0x026 in this entry should never become 0x28 or 0x29 in order to avoid any misinterpretation with the EBPB format under non-FAT32 aware operating systems.
0x028	0x1D	2	Drive description / mirroring flags (bits 3-0: zero-based number of active FAT, if bit 7 set.[19] If bit 7 is clear, all FATs are mirrored as usual. Other bits reserved and should be 0.) DR-DOS 7.07 FAT32 boot sectors with dual LBA and CHS support utilize bits 15-8 to store an access flag and part of a message. These bits contain either bit pattern 0110:1111b (low-case letter 'o', bit 13 set for CHS access) or 0100:1111b (upper-case letter 'O', bit 13 cleared for LBA access). The byte is also used for the second character in a potential "No␣IBMBIO␣␣COM" error message (see offset 0x034), displayed either in mixed or upper case, thereby indicating which access type failed). Formating tools or non-DR SYS-type tools may clear these bits, but other disk tools should leave bits 15-8 unchanged.
0x02A	0x1F	2	Version (defined as 0.0). The high byte of the version number is stored at offset 0x02B, and the low byte at offset 0x02A.[19] FAT32 implementations should refuse to mount volumes with version numbers unknown by them. The FAT+ specification proposes to change the version number of FAT32+ volumes with file exceeding the standard FAT32 file size limit of 4 GiB - 1 to a version number 0.1 to keep implementations unaware of this extension from mounting the volume. Aware implementations should still mount and expect FAT+ large file entries on volumes with a version number 0.0 for maximum compatibility.[2]
0x02C	0x21	4	Cluster number of root directory start, typically 2 (first cluster[22]) if it contains no bad sector. Microsoft's FAT32 implementation imposes an artificial limit of 65,535 entries per directories, whilst many third-party implementations do not. A cluster value of 0 is not officially allowed and can never indicate a valid root directory start cluster. Some non-standard FAT32 implementations may treat it as an indicator to search for a fixed-sized root directory where it would be expected on FAT16 volumes; see offset 0x011.
0x030	0x25	2	Logical sector number of FS Information Sector, typically 1, i.e., the second of the three FAT32 boot sectors. Some FAT32 implementations support a slight variation of Microsoft's specification in making the FS Information Sector optional by specifying a value of 0xFFFF[58] (or 0x0000) in this entry. Since logical sector 0 can never be a valid FS Information Sector, but FS Information Sectors use the same signature as found on many boot sectors, file system implementations should never attempt to use logical sector 0 as FS Information sector and instead assume that the feature is unsupported on that particular volume. Without a FS Information Sector, the minimum allowed logical sector size of FAT32 volumes can be reduced downto 128 bytes for special purposes.
0x032	0x27	2	First logical sector number of a copy of the three FAT32 boot sectors, typically 6.[19] Since DR-DOS 7.0x FAT32 formatted volumes use a single-sector boot sector, some volumes formatted under DR-DOS use a value of 2 here. Values of 0x0000[19] (and/or 0xFFFF[58]) are reserved and indicate that no backup sector is available.
0x034	0x29	12	Reserved (may be changed to format filler byte 0xF6[nb 6] as an artefact by MS-DOS FDISK, must be initialized to 0 by formatting tools, but must not be changed by file system implementations or disk tools later on.) DR-DOS 7.07 FAT32 boot sectors use these 12 bytes to store the filename of the "IBMBIO␣␣COM"[nb 11] file to be loaded (up to the first 29,696 bytes or the actual file size, whatever is smaller) and executed by the boot sector, followed by a terminating NUL (0x00) character. This is also part of an error message, indicating the actual boot file name and access method (see offset 0x028).
0x040	0x35	1	Cf. 0x024 for FAT12/FAT16 (Physical Drive Number)
0x041	0x36	1	Cf. 0x025 for FAT12/FAT16 (Used for various purposes; see FAT12/FAT16) May hold format filler byte 0xF6[nb 6] artefacts after partitioning with MS-DOS FDISK, but not yet formatted.
0x042	0x37	1	Cf. 0x026 for FAT12/FAT16 (Extended boot signature, 0x29) Most FAT32 file system implementations do not support an alternative signature of 0x28 to indicate a shortened form of the FAT32 EBPB with only the serial number following (and no Volume Label and File system type entries), but since these 19 mostly unused bytes might serve different purposes in some scenarios, implementations should accept 0x28 as an alternative signature and then fall back to use the directory volume label in the file system instead of in the EBPB for compatibility with potential extensions.
0x043	0x38	4	Cf. 0x027 for FAT12/FAT16 (Volume ID)
0x047	0x3C	11	Cf. 0x02B for FAT12/FAT16 (Volume Label)
0x052	0x47	8	Cf. 0x036 for FAT12/FAT16 (File system type, padded with blanks (0x20), e.g., "FAT32␣␣␣")

Exceptions

Versions of DOS before 3.2 totally or partially relied on the media descriptor byte in the BPB or the FAT ID byte in cluster 0 of the first FAT in order to determine FAT12 diskette formats even if a BPB is present. Depending on the FAT ID found and the drive type detected they default to use one of the following BPB prototypes instead of using the values actually stored in the BPB.[nb 7]

Originally, the FAT ID was meant to be a bit flag with all bits set except for bit 2 cleared to indicate a 80 track (vs. 40 track) format, bit 1 cleared to indicate a 9 sector (vs. 8 sector) format, and bit 0 cleared to indicate a single-sided (vs. double-sided) format,[47] but this scheme was not followed by all OEMs and became obsolete with the introduction of hard disks and high-density formats. Also, the various 8-inch formats supported by 86-DOS and MS-DOS do not fit this scheme.

FAT ID (compare with media ID at BPB offset 0x0A)[60][61]	**0xFF**		**0xFE**			**0xFD**			**0xFC**	0xFB	0xFA	**0xF9**			0xF8			**0xF0**		
Size	8"	5.25"	8"	8"	5.25"	8"	8"	5.25"	5.25"	5.25" / 3.5"	5.25" / 3.5"	5.25"	3.5"	3.5"	5.25"	5.25" / 3.5"	3.5"	3.5"	3.5"	
Density	?	DD 48tpi	SD	DD	DD 48tpi	?	SD	DD 48tpi	DD 48tpi	?	?	HD 96tpi	DD 135tpi	HD 135tpi	QD 96tpi	?	DD	HD 135tpi	ED	
Modulation	?	MFM	FM	MFM	MFM	?	FM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	
Formatted capacity (KiB)	?	320	250	1200	160	?	500	360	180	640	320	1200	720	1440	720	360	360	1440	2880	
Cylinders (CHS)	77	40	77	77	40	77	77	40	40	80	80	80	80	80	80	80	80	80	80	
Physical sectors / track (BPB offset 0x0D)	?	8	26	8	8	?	26	9	9	8	8	15	9	18	9 (8[64])	9	9	18	36	
Number of heads (BPB offset 0x0F)	?	2	1	2[47][60] (1)	1	?	2[60] (1?[47])	2	1	2	1	2	2	2	2	1	1	2	2	
Byte payload / physical sector	?	512	128	1024	512	?	128	512	512	512	512	512	512	512	512	512	512	512	512	
Bytes / logical sector (BPB offset 0x00)	?	512	128	1024	512	?	128	512	512	512	512	512	512	512	512	512	512	512	512	
Logical sectors / cluster (BPB offset 0x02)	?	2	4	1	1	?	4	2	1	2	1[60] (2?[47])	1	2	1	?	2	?	1	2	
Reserved logical sectors (BPB offset 0x03)	?	1	1	1	1	?	4	1	1	1	1	1	1 (2)	1	1	1	1	1	1	
Number of FATs (BPB offset 0x05)	?	2	2	2	2	?	2	2	2	2	2	2	2	2	2	2	2	2	2	
Root directory entries (BPB offset 0x06)	?	112 (7 sectors)	68 (17 sectors)	192 (6 sectors)	64 (4 sectors)	?	68 (17 sectors)	112 (7 sectors)	64 (4 sectors)	112 (7 sectors)	112 (7 sectors)	224 (14 sectors)	112 (7 sectors)	224 (14 sectors)	?	112 (7 sectors)	?	224 (14 sectors)	240 (15 sectors)	
Total logical sectors (BPB offset 0x08)	?	640	2002	1232[60] (616[47])	320	?	4004[60] (2002?[47])	720	360	1280	640	2400	1440	2880	?	720	?	2880	5760	
Logical sectors / FAT (BPB offset 0x0B)	?	1	6	2	1	?	6?[60]	2	2	2	2[60] (1?[47])	7	3	9 (7)	?	2	?	9	9	
Hidden sectors (BPB offset 0x11)	?	0	3[60] (0[47])	0	0	?	0	0	0	0	0	0	0	0	0	0	0	0	0	
Total number of clusters	?	315	497	1227	313	?	997?[60]	354	351	?	?	2371	713	2847?	?	?	?	2847	2863	
Logical sector order	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
Sector mapping	?	sector+/ head+/ track+	sector+/ track+	sector+/ head+/ track+	sector+/ track+	?	sector+/ head+/ track+	sector+/ head+/ track+	sector+/ track+	sector+/ head+/ track+	sector+/ track+	sector+/ head+/ track+	sector+/ head+/ track+	sector+/ head+/ track+	?	sector+/ track+	sector+/ track+	sector+/ head+/ track+	sector+/ head+/ track+	
First physical sector (CHS)	?	1	1	1	1	?	1	1	1	?	?	1	1	1	?	1	?	1	1	
DRIVER.SYS /F:n	?	0	3	4	0	?	3	0	0	?	?	1	2	7	?	?	?	7	9	
BPB Presence	?	?	?	?	?	?	?	?	?	?	?	Yes	Yes	Yes	?	?	?	Yes	Yes	
Support	?	DOS 1.1	DOS 1.0	DOS 2.0	DOS 1.0	?	DOS 2.0	DOS 2.0	DOS 2.0	?	?	DOS 3.0	DOS 3.2	DOS 3.2 only; (DR-DOS)	Sanyo 55x DS-DOS 2.11 only	MS-DOS 3.1[47]	MSX-DOS	DOS 3.3	DOS 5.0	

Microsoft recommends to distinguish between the two 8-inch formats for FAT ID 0xFE by trying to read of a single-density address mark. If this results in an error, the medium must be double-density.[61]

The table does not list a number of incompatible 8-inch and 5.25-inch FAT12 floppy formats supported by 86-DOS, which differ either in the size of the directory entries (16 bytes vs. 32 bytes) or in the extent of the reserved sectors area (several whole tracks vs. one logical sector only).

The implementation of a single-sided 315 KiB FAT12 format used in MS-DOS for the Apricot PC and F1e[65] had a different boot sector layout, to accommodate that computer's non-IBM compatible BIOS. The jump instruction and OEM name were omitted, and the MS-DOS BPB parameters (offsets 0x00B-0x017 in the standard boot sector) were located at offset 0x050. The Portable, F1, PC duo and Xi FD supported a non-standard double-sided 720 KiB FAT12 format instead.[65] The differences in the boot sector layout and media IDs made these formats incompatible with many other operating systems. The geometry parameters for these formats are:

- 315 KiB: Bytes per logical sector: 512 bytes, logical sectors per cluster: 1, reserved logical sectors: 1, number of FATs: 2, root directory entries: 128, total logical sectors: 630, FAT ID: `0xFC`, logical sectors per FAT: 2, physical sectors per track: 9, number of heads: 1.[65][66]
- 720 KiB: Bytes per logical sector: 512 bytes, logical sectors per cluster: 2, reserved logical sectors: 1, number of FATs: 2, root directory entries: 176, total logical sectors: 1440, FAT ID: `0xFE`, logical sectors per FAT: 3, physical sectors per track: 9, number of heads: 2.[65]

Later versions of Apricot MS-DOS gained the ability to read and write disks with the standard boot sector in addition to those with the Apricot one. These formats were also supported by DOS Plus 2.1e/g for the Apricot ACT series.

The DOS Plus adaptation for the BBC Master 512 supported two FAT12 formats on 80-track, double-sided, double-density 5.25" drives, which did not use conventional boot sectors at all. 800 KiB data disks omitted a boot sector and began with a single copy of the FAT.[66] The first byte of the relocated FAT in logical sector 0 was used to determine the disk's capacity. 640 KiB boot disks began with a miniature ADFS file system containing the boot loader, followed by a single FAT.[66][67] Also, the 640 KiB format differed by using physical CHS sector numbers starting with 0 (not 1, as common) and incrementing sectors in the order sector-track-head (not sector-head-track, as common).[67] The FAT started at the beginning of the next track. These differences make these formats unrecognizable by other operating systems. The geometry parameters for these formats are:

- 800 KiB: Bytes per logical sector: 1024 bytes, logical sectors per cluster: 1, reserved logical sectors: 0, number of FATs: 1, root directory entries: 192, total logical sectors: 800, FAT ID: `0xFD`, logical sectors per FAT: 2, physical sectors per track: 5, number of heads: 2.[66][67]
- 640 KiB: Bytes per logical sector: 256 bytes, logical sectors per cluster: 8, reserved logical sectors: 16, number of FATs: 1, root directory entries: 112, total logical sectors: 2560, FAT ID: `0xFF`, logical sectors per FAT: 2, physical sectors per track: 16, number of heads: 2.[66][67]

DOS Plus for the Master 512 could also access standard PC disks formatted to 180 KiB or 360 KiB, using the first byte of the FAT in logical sector 1 to determine the capacity.

FS Information Sector

The "FS Information Sector" was introduced in FAT32[68] for speeding up access times of certain operations (in particular, getting the amount of free space). It is located at a logical sector number specified in the FAT32 EBPB boot record at position `0x030` (usually logical sector 1, immediately after the boot record itself).

Byte Offset	Length (bytes)	Description
0x000	4	FS information sector signature (`0x52 0x52 0x61 0x41` = "`RRaA`") For as long as the FS Information Sector is located in logical sector 1, the location, where the FAT typically started in FAT12 and FAT16 filesystems (with only one reserved sectors), the presence of this signature ensures that early versions of DOS will never attempt to mount a FAT32 volume, as they expect the values in cluster 0 and cluster 1 to follow certain bit patterns, which are not met by this signature.
0x004	480	Reserved (byte values should be set to `0x00` during format, but not be relied upon and never changed later on)
0x1E4	4	FS information sector signature (`0x72 0x72 0x41 0x61` = "`rrAa`")
0x1E8	4	Last known number of free data clusters on the volume, or `0xFFFFFFFF` if unknown. Should be set to `0xFFFFFFFF` during format and updated by the operating system later on. Must not be absolutely relied upon to be correct in all scenarios. Before using this value, the operating system should sanity check this value to be at least smaller or equal to the volume's count of clusters.
0x1EC	4	Number of the most recently known to be allocated data cluster. Should be set to `0xFFFFFFFF` during format and updated by the operating system later on. With `0xFFFFFFFF` the system should start at cluster `0x00000002`. Must not be absolutely relied upon to be correct in all scenarios. Before using this value, the operating system should sanity check this value to be a valid cluster number on the volume.
0x1F0	12	Reserved (byte values should be set to `0x00` during format, but not be relied upon and never changed later on)
0x1FC	4	FS information sector signature (`0x00 0x00 0x55 0xAA`)[19][nb 8] (All four bytes should match before the contents of this sector should be assumed to be in valid format.)

The sector's data may be outdated and not reflect the current media contents, because not all operating systems update or use this sector, and even if they do, the contents is not valid when the medium has been ejected without properly unmounting the volume or after a power-failure. Therefore, operating systems should first inspect a volume's optional shutdown status bitflags residing in the FAT entry of cluster 1 or the FAT32 EBPB at offset `0x041` and ignore the data stored in the FS information sector, if these bitflags indicate that the volume was not properly unmounted before. This does not cause any problems other than a possible speed penalty for the first free space query or data cluster allocation; see fragmentation.

If this sector is present on a FAT32 volume, the minimum allowed logical sector size is 512 bytes, whereas otherwise it would be 128 bytes. Some FAT32 implementations support a slight variation of Microsoft's specification by making the FS information sector optional by specifying a value of `0xFFFF`[58] (or `0x0000`) in the entry at offset `0x030`.

File Allocation Table

A volume's data area is divided up into identically sized *clusters*, small blocks of contiguous space. Cluster sizes vary depending on the type of FAT file system being used and the size of the partition; typically cluster sizes lie somewhere between 2 KiB and 32 KiB.

Each file may occupy one or more of these clusters depending on its size; thus, a file is represented by a chain of these clusters (referred to as a singly linked list). However these clusters are not necessarily stored adjacent to one another on the disk's surface but are often instead *fragmented* throughout the Data Region.

The *File Allocation Table* (*FAT*) is contiguous number of sectors immediately following the area of reserved sectors. It represents a list of entries that map to each cluster on the volume. Each entry records one of five things:

- the cluster number of the next cluster in a chain
- a special *end of cluster-chain* (*EOC*) entry that indicates the end of a chain
- a special entry to mark a bad cluster
- a zero to note that the cluster is unused

Each version of the FAT file system uses a different size for FAT entries. Smaller numbers result in a smaller FAT, but waste space in large partitions by needing to allocate in large clusters. The FAT12 file system uses 12 bits per FAT entry, thus two entries span 3 bytes. It is consistently little-endian: if those three bytes are considered as one little-endian 24-bit number, the 12 least significant bits represent the first entry (cluster 0) and the 12 most significant bits the second (cluster 1).

For very early versions of DOS to recognize the file system, the FAT must start with the volume's second sector (logical sector 1 with physical CHS address 0/0/2 or LBA address 1), that is, immediately following the boot sector. Operating systems assume this hard-wired location of the FAT in order to find the FAT ID in the FAT's cluster 0 entry on DOS 1.0-1.1 FAT diskettes, where no valid BPB is found.

The first two entries in a FAT store special values:

The first entry (cluster 0 in the FAT) holds the FAT ID (allowed values `0xF0-0xFF` with `0xF1-0xF7` reserved) in bits 7-0, which is also copied into the BPB of the boot sector, offset `0x015` since DOS 2.0. The remaining 4 bits (if FAT12), 8 bits (if FAT16) or 20 bits (if FAT32) of this entry are always 1. For FAT IDs other than `0xFF` (and `0x00`) it is possible to determine the correct nibble and byte order (to be) used by the file system driver, however, the FAT file system officially uses a little-endian representation only and there are no

known implementations of variants using big-endian values instead.

The second entry (cluster 1 in the FAT) nominally stores the end-of-cluster-chain marker as used by the formater, but typically always holds `0xFFF` / `0xFFFF` / `0x0FFFFFFF`, that is, with the exception of bits 31-28 on FAT32 volumes these bits are normally always set. Some Microsoft operating systems, however, set these bits if the volume is not the volume holding the running operating system (that is, use `0xFFFFFFFF` instead of `0x0FFFFFFF` here).[69] (In conjunction with alternative end-of-chain markers the lowest bits 2-0 can become zero for the lowest allowed end-of-chain marker `0xFF8` / `0xFFF8` / `0x?FFFFFF8`; bit 3 should be reserved as well given that clusters `0xFF0` / `0xFFF0` / `0x?FFFFFF0` and higher are officially reserved. Some operating systems may not be able to mount some volumes if any of these bits are not set, therefore the default end-of-chain marker should not be changed.)

Since DOS 7.1 the two most-significant bits of this cluster entry may hold two optional bitflags representing the current volume status on FAT16 and FAT32, but not on FAT12 volumes. These bitflags are not supported by all operating systems, but operating systems supporting this feature would set these bits on shutdown and clear the most significant bit on startup:
If bit 15 (on FAT16) or bit 27 (on FAT32)[33] is not set when mounting the volume, the volume was not properly unmounted before shutdown or ejection and thus is in an unknown and possibly "dirty" state.[62] On FAT32 volumes, the FS Information Sector may hold outdated data and thus should not be used. The operating system would then typically run SCANDISK or CHKDSK on the next startup[nb 12][33] (but not on insertion of removable media) to ensure and possibly reestablish the volume's integrity.
If bit 14 (on FAT16) or bit 26 (on FAT32)[33] is cleared, the operating system has encountered disk I/O errors on startup,[33] a possible indication for bad sectors. Operating systems aware of this extension will interpret this as a recommendation to carry out a surface scan (SCANDISK) on the next boot.[33][62] (A similar set of bitflags exists in the FAT12/FAT16 EBPB at offset `0x1A` or the FAT32 EBPB at offset `0x36`. While the cluster 1 entry can be accessed by file system drivers once they have mounted the volume, the EBPB entry is available even when the volume is not mounted and thus easier to use by disk block device drivers or partitioning tools.)

If the number of FATs in the BPB is not set to 2, the second cluster entry in the first FAT (cluster 1) may also reflect the status of a TFAT volume for TFAT-aware operating systems. If the cluster 1 entry in that FAT holds the value 0, this may indicate that the second FAT represents the last known valid transaction state and should be copied over the first FAT, whereas the first FAT should be copied over the second FAT if all bits are set.

Because these first two FAT entries store special values, there are no data clusters 0 or 1. The first data cluster (after the root directory if FAT12/FAT16) is cluster 2,[22] and cluster 2 is by definition the first cluster in the data area.

FAT entry values:

FAT12	FAT16	FAT32	Description
0x000	0x0000	0x?0000000	Free Cluster; also used by DOS to refer to the parent directory starting cluster in ".." entries of subdirectories of the root directory on FAT12/FAT16 volumes.[12][21] Otherwise, if this value occurs in cluster chains (e.g., in directory entries of zero length or deleted files), file system implementations should treat this like an end-of-chain marker.[47]
0x001	0x0001	0x?0000001	Reserved for internal purposes; MS-DOS/PC DOS use this cluster value as a temporary non-free cluster indicator while constructing cluster chains during file allocation (only seen on disk if there is a crash or power failure in the middle of this process).[12][21] If this value occurs in on-disk cluster chains, file system implementations should treat this like an end-of-chain marker.
0x002 - 0xFEF	0x0002 - 0xFFEF (0x0002 - 0x7FFF)	0x?0000002 - 0x?FFFFFEF	Used as data clusters; value points to next cluster. MS-DOS/PC DOS accept values up to `0xFEF` / `0xFFEF` / `0x0FFFFFEF` (sometimes more; see below), whereas for Atari GEMDOS only values up to `0x7FFF` are allowed on FAT16 volumes.
0xFF0[nb 2] - 0xFF5 (0xFF1 - 0xFF5)	0xFFF0 - 0xFFF5	0x?FFFFFF0 - 0x?FFFFFF5	Reserved in some contexts,[17] or also used[5][6][7][19][20] as data clusters in some non-standard systems. Volume sizes, which would utilize these values as data clusters, should be avoided, but if these values occur in existing volumes, the file system must treat them as normal data clusters in cluster-chains (ideally applying additional sanity checks), similar to what MS-DOS, PC DOS and DR-DOS do, and should avoid to allocate them for files otherwise. MS-DOS/PC DOS 3.3 and higher treats a value of `0xFF0`[nb 2] on FAT12 (but not on FAT16 or FAT32) volumes as additional end-of-chain marker similar to `0xFF8-0xFFF`. For compatibility with MS-DOS/PC DOS, file systems should avoid to use data cluster `0xFF0` in cluster chains on FAT12 volumes (that is, treat it as a reserved cluster similar to `0xFF7`). (NB. The correspondence of the low byte of the cluster number with the FAT ID and media descriptor values is the reason, why these cluster values are reserved.)
0xFF6	0xFFF6	0x?FFFFFF6	Reserved; do not use.[5][6][7][19][20][59] (NB. Corresponds with the default format filler value `0xF6` on IBM compatible machines.) Volumes should not be created which would utilize this value as data cluster, but if this value occurs in existing volumes, the file system must treat it as normal data cluster in cluster-chains (ideally applying additional sanity checks), and should avoid to allocate it for files otherwise.[47]
0xFF7	0xFFF7	0x?FFFFFF7	Bad sector in cluster or reserved cluster. The cutover values for the maximum number of clusters for FAT12 and FAT16 file systems are defined as such that the highest possible data cluster values (`0xFF5` and `0xFFF5`, respectively) will always be smaller than this value. Therefore, this value cannot normally occur in cluster-chains, but if it does, it may be treated as a normal data cluster, since `0xFF7` could have been a non-standard data cluster on FAT12 volumes before the introduction of FAT16 with DOS 3.0,[47] and `0xFFF7` could have been a non-standard data cluster on FAT16 volumes before the introduction of FAT32 with DOS 7.10. Theoretically, `0x0FFFFFF7` can be part of a valid cluster chain on FAT32 volumes, but disk utilities should avoid creating FAT32 volumes, where this condition could occur. The file system should avoid to allocate this cluster for files.[47] Disk utilities must not attempt to restore "lost clusters" holding this value in the FAT, but count them as bad clusters.
0xFF8 - 0xFFF (and optionally 0xFF0;[nb 2] see note)	0xFFF8 - 0xFFFF	0x?FFFFFF8 - 0x?FFFFFFF	Last cluster in file (EOC). File system implementations must treat all these values as end-of-chain marker at the same time.[47] Most file system implementations use `0xFFF`[47] / `0xFFFF`[47] / `0x0FFFFFFF` as end-of-file marker when allocating files, but versions of Linux before 2.5.40 used `0xFF8` / `0xFFF8` / `0x0FFFFFF8`,[70] whereas some disk repair and defragment tools utilize other values in the set (e.g., SCANDISK may use `0xFF8` / `0xFFF8` / `0x0FFFFFF8` instead). While in the original 8-bit FAT implementation in Microsoft's Stand-alone Disk BASIC different end markers (`0xC0..0xCD`) were used to indicate the number of sectors (0 to 13) used up in the last cluster occupied by a file, different end markers were repurposed under DOS to indicate different types of media,[47] with the currently used end marker indicated in the cluster 1 entry, however, this concept does not seem to have been broadly utilized in practice—and to the extent that in some scenarios volumes may not be recognized by some operating systems, if some of the low-order bits of the value stored in cluster 1 are not set. Also, some faulty file system implementations only accept `0xFFF` / `0xFFFF` / `0x?FFFFFFF` as valid end-of-chain marker. File system implementations should check cluster values in cluster-chains against the maximum allowed cluster value calculated by the actual size of the volume and treat higher values as if they were end-of-chain markers as well. (The low byte of the cluster number conceptually corresponds with the FAT ID and media descriptor values;[47] see note above for MS-DOS/PC DOS special usage of `0xFF0`[nb 2] on FAT12 volumes.)

Despite its name FAT32 uses only 28 bits of the 32 possible bits. The upper 4 bits are usually zero, but are reserved and should be left untouched. A standard conformant FAT32 file system driver or maintenance tool must not rely on the upper 4 bits to be zero and it must strip them off before evaluating the cluster number in order to cope with possible future expansions where these bits may be used for other purposes. They must not be cleared by the file system driver when allocating new clusters, but should be cleared during a reformat.

Directory table

A *directory table* is a special type of file that represents a directory (also known as a folder). Each file or directory stored within it is represented by a 32-byte entry in the table. Each entry records the name, extension, attributes (archive, directory, hidden, read-only, system and volume), the date and time of last modification, the address of the first cluster of the file/directory's data and finally the size of the file/directory. Aside from the Root Directory Table in FAT12 and FAT16 file systems, which occupies the special *Root Directory Region* location, all Directory Tables are stored in the Data Region. The actual number of entries in a directory stored in the Data Region can grow by adding another cluster to the chain in the FAT.

The FAT file system itself does not impose any limits on the depth of a subdirectory tree for as long as there are free clusters available to allocate the subdirectories, however, the Current Directory Structure under MS-DOS/PC DOS limits the absolute path of a directory to 66 characters (including the drive letter, but excluding the zero delimiter),[5][6][7] thereby limiting the maximum supported depth of subdirectories to 32, whatever occurs earlier. Concurrent DOS, Multiuser DOS and DR DOS 3.31 to 6.0 (up to including the 1992-11 updates) do not store absolute paths to working directories internally and therefore do not show this limitation.[71] The same applies to Atari GEMDOS, but the Atari Desktop does not support more than 8 sub-directory levels. Most applications aware of this extension support paths up to at least 127 bytes. FlexOS, 4680 OS and 4690 OS support a length of up to 127 bytes as well, allowing depths down to 60 levels.[36] PalmDOS, DR DOS 6.0 (since BDOS 7.1) and higher, Novell DOS, and OpenDOS sport a MS-DOS-compatible CDS and therefore have the same length limits as MS-DOS/PC DOS.

Note that before each entry there can be "fake entries" to support a long filename (LFN); see further below.

Legal characters for DOS short filenames include the following:

- Upper case letters `A–Z`
- Numbers `0–9`
- Space (though trailing spaces in either the base name or the extension are considered to be padding and not a part of the file name, also filenames with space in them could not easily be used on the DOS command line prior to Windows 95 because of the lack of a suitable escaping system). Another exception are the internal commands `MKDIR`/`MD` and `RMDIR`/`RD` under DR-DOS which accept single arguments and therefore allow spaces to be entered.

- `! # $ % & ' ( ) - @ ^ _ ` { } ~`
- Values 128–255

This excludes the following ASCII characters:

- `" * / : < > ? \ |`
  Windows/MS-DOS has no shell escape character
- `+ , . ; = [ ]`
  They are allowed in long file names only.
- Lower case letters `a–z`
  Stored as `A–Z`. Allowed in long file names.
- Control characters 0–31
- Value 127 (DEL)

The following additional characters are allowed on Atari's GEMDOS, but should be avoided for compatibility with MS-DOS/PC DOS:

- `" + , ; < = > [ ] |`

The semicolon (;) should be avoided in filenames under DR DOS 3.31 and higher, PalmDOS, Novell DOS, OpenDOS, Concurrent DOS, Multiuser DOS, System Manager and REAL/32, because it may conflict with the syntax to specify file and directory passwords: "`...\DIRSPEC.EXT;DIRPWD\FILESPEC.EXT;FILEPWD`". The operating system will strip off one[71] (and also two—since DR-DOS 7.02) semicolons and pending passwords from the filenames before storing them on disk. (The command processor 4DOS uses semicolons for include lists and requires the semicolon to be doubled for password protected files with any commands supporting wildcards.[71])

The @-character is used for filelists by many DR-DOS, PalmDOS, Novell DOS, OpenDOS and Multiuser DOS, System Manager and REAL/32 commands as well as by 4DOS and may therefore sometimes be difficult to use in filenames.[71]

Under Multiuser DOS and REAL/32, the exclamation mark (!) is not a valid filename character since it is used to separate multiple commands in a single command line.[71]

Under IBM 4680 OS and 4690 OS, the following characters are not allowed in filenames:

- `? * : . ; , [ ] ! + = < > " - / \ |`

Additionally, the following special characters are not allowed in the first, fourth, fifth and eight character of a filename, as they conflict with the host command processor (HCP) and input sequence table build file names:

- `@ # ( ) { } $ &`

The DOS file names are in the current OEM character set: this can have surprising effects if characters handled in one way for a given code page are interpreted differently for another code page (DOS command CHCP) with respect to lower and upper case, sorting, or validity as file name character.

Directory entry

Before Microsoft added support for long filenames and creation/access time stamps, bytes `0x0C–0x15` of the directory entry were used by other operating systems to store additional metadata, most notably the operating systems of the Digital Research family stored file passwords, access rights, owner IDs, and file deletion data there. While Microsoft's newer extensions are not fully compatible with these extensions by default, most of them can coexist in third-party FAT implementations (at least on FAT12 and FAT16 volumes).

Directory entries, both in the Root Directory Region and in subdirectories, are of the following format (see also 8.3 filename):

<table>
<tr><th>Byte Offset</th><th>Length (bytes)</th><th>Description</th></tr>
<tr><td>0x00</td><td>8</td><td>Short file name (padded with spaces)

The first byte can have the following special values:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0x00</td><td>Entry is available and no subsequent entry is in use. Also serves as an end marker when DOS scans a directory table. (Since MS-DOS/PC DOS 2.0, but not in 86-DOS).</td></tr>
<tr><td>0x05</td><td>Initial character is actually <code>0xE5</code>. (since DOS 3.0)

Under DR DOS 6.0 and higher, including PalmDOS, Novell DOS and OpenDOS, <code>0x05</code> is also used for pending delete files under DELWATCH. Once they are removed from the deletion tracking queue, the first character of an erased file is replaced by <code>0xE5</code>.</td></tr>
<tr><td>0x2E</td><td>'Dot' entry; either "<code>.</code>" or "<code>..</code>" (since DOS 2.0)</td></tr>
<tr><td>0xE5</td><td>Entry has been previously erased and/or is available. File undelete utilities must replace this character with a regular character as part of the undeletion process. See also: <code>0x05</code>.

(The reason, why <code>0xE5</code> was chosen for this purpose in 86-DOS is down to the fact, that 8-inch CP/M floppies came pre-formatted with this value filled and so could be used to store files out-of-the box.[12][nb 5])</td></tr>
</table>
Versions of DOS prior to 5.0 start scanning directory tables from the top of the directory table to the bottom. In order to increase chances for successful file undeletion, DOS 5.0 and higher will remember the position of the last written directory entry and use this as a starting point for directory table scans.</td></tr>
<tr><td>0x08</td><td>3</td><td>Short file extension (padded with spaces)</td></tr>
<tr><td></td><td></td><td>File Attributes</td></tr>
</table>

0x0B 1

Bit	Mask	Description
0	0x01	Read Only. (Since DOS 2.0) If this bit is set, the operating system will not allow a file to be opened for modification. Deliberately setting this bit for files which will not be written to (executables, shared libraries and data files) may help avoid problems with concurrent file access in multi-tasking, multi-user or network environments with applications not specifically designed to work in such environments (i.e. non-SHARE-enabled programs). Files larger than 4 GiB following the FAT+[2] proposal may have the Read-only attribute set to keep unaware operating systems from modifying the file. If FAT+ large files are opened for write via FAT+ APIs, enabled implemenations may ignore the Read-only attribute of FAT+ files when the System attribute is not set at the same time, and otherwise treat the System attribute as alternative Read-only attribute for FAT+ large files.
1	0x02	Hidden. Hides files or directories from normal directory views. Under DR DOS 3.31 and higher, under PalmDOS, Novell DOS, OpenDOS, Concurrent DOS, Multiuser DOS, REAL/32, password protected files and directories also have the hidden attribute set.[71] Password-aware operating systems should not hide password-protected files from directory views, even if this bit may be set. The password protection mechanism does not depend on the hidden attribute being set up to including DR-DOS 7.03, but if the hidden attribute is set, it should not be cleared for any password-protected files. Files larger than 4 GiB following the FAT+[2] proposal may have the hidden attribute set to hide them from normal directory scans on unaware operating systems; in this special case, they may ignore the Hidden attribute for FAT+ files when the System attribute is not set at the same time, and otherwise treat the System attribute as alternative Hidden attribute for FAT+ large files.
2	0x04	System. Indicates that the file belongs to the system and must not be physically moved (e.g., during defragmentation), because there may be references into the file using absolute addressing bypassing the file system (boot loaders, kernel images, swap files, extended attributes, etc.).
3	0x08	Volume Label. (Since DOS 2.0) Indicates an optional directory volume label, normally only residing in a volume's root directory. Ideally, the volume label should be the first entry in the directory (after reserved entries) in order to avoid problems with VFAT LFNs. If this volume label is not present, some systems may fall back to display the partition volume label instead, if a EBPB is present in the boot sector (not present with some non-bootable block device drivers, and possibly not writeable with boot sector write protection). Even if this volume label is present, partitioning tools like FDISK may display the partition volume label instead. The entry occupies a directory entry but has no file associated with it. Volume labels have a filesize entry of zero. Pending delete files and directories under DELWATCH have the volume attribute set until they are purged or undeleted.[71]
4	0x10	Subdirectory. (Since DOS 2.0) Indicates that the cluster-chain associated with this entry gets interpreted as subdirectory instead of as a file. Subdirectories have a filesize entry of zero.
5	0x20	Archive. (Since DOS 2.0) Typically set by the operating system as soon as the file is created or modified to mark the file as "dirty", and reset by backup software once the file has been backed up to indicate "pure" state.
6	0x40	Device (internally set for character device names found in filespecs, never found on disk), must not be changed by disk tools.
7	0x80	Reserved, must not be changed by disk tools.

Under DR DOS 6.0 and higher, including PalmDOS, Novell DOS and OpenDOS, the volume attribute is set for pending delete files and directories under DELWATCH.

An attribute combination of `0x0F` is used to designate a VFAT long file name entry since MS-DOS 7.0. Older versions of DOS can mistake this for a directory volume label, as they take the first entry with volume attribute set as volume label. This problem can be avoided if a directory volume label is enforced as part of the format process; for this reason some disk tools explicitly write dummy "`NO␣NAME␣␣␣␣`" directory volume labels when the user does not specify a volume label.[nb 13] Since volume labels normally don't have the system attribute set at the same time, it is possible to distinguish between volume labels and VFAT LFN entries. The attribute combination `0x0F` could occasionally also occur as part of a valid pending delete file under DELWATCH, however on FAT12 and FAT16 volumes, VFAT LFN entries always have the cluster value at `0x1A` set to `0x0000` and the length entry at `0x1C` is never `0x00000000`, whereas the entry at `0x1A` is always non-zero for pending delete files under DELWATCH (with the possible exception of zero-length files or deleted files following the FAT16+[2] proposal indicated by `0x00000000` at `0x1C`). This check does not work on FAT32 volumes.

0x0C 1

- CP/M-86 and DOS Plus store user attributes F1'—F4' here.[72] (DOS Plus 1.2 with BDOS 4.1 supports passwords only on CP/M media, not on FAT12 or FAT16 media.[73] While DOS Plus 2.1 supported logical sectored FATs with a partition type `0xF2`, FAT16B and FAT32 volumes were not supported by these operation systems. Even if a partition would have been converted to FAT16B it would still not be larger than 32 MiB. Therefore, this usage is not conflictive with FAT32.IFS, FAT16+ or FAT32+ as they can never occur on the same type of volume.):

Bit	Mask	Description
7	0x80	F1': Modify default open rules[71]
6	0x40	F2': Partial close default[71]
5	0x20	F3': Ignore Close Checksum Error[71]
4	0x10	F4': Disable checksums[71]
3	0x08	Reserved
2	0x04	Delete requires password
1	0x02	Write requires password
0	0x01	Read requires password

- MSX-DOS 2: For a deleted file, the original first character of the filename. For the same feature in various other operating systems, see offset `0x0D` if enabled in MSX boot sectors at sector offset `0x026`. MSX-DOS supported FAT12 volumes only, but third-party extensions for FAT16 volumes exist. Therefore, this usage is not conflictive with FAT32.IFS and FAT32+ below. It does not conflict with the usage for user attributes under CP/M-86 and DOS Plus as well, since they are no longer important for deleted files.
- Windows NT and later versions uses bits 3 and 4 to encode case information (see below); otherwise 0.[74]
- DR-DOS 7.0x reserved bits other than 3 and 4 for internal purposes since 1997. The value should be set to 0 by formating tools and must not be changed by disk tools.[71]
- On FAT32 volumes under OS/2 and eComStation the third-party FAT32.IFS driver utilizes this entry as a mark byte to indicate the presence of extra "`␣EA.␣SF`" files holding extended attributes with parameter `/EAS`. Version 0.70 to 0.96 used the magic values `0x00` (no EAs), `0xEA` (normal EAs) and `0xEC` (critical EAs),[39] whereas version 0.97 and higher since 2003-09 use `0x00`, `0x40` (normal EAs) and `0x80` (critical EAs) as bitflags for compatibility with

Windows NT.[38][75]

- Bits other than 3 and 4 are utilized by FAT+,[2] a proposal how to store files larger than 4 GiB on FAT32 (and FAT16B) volumes, currently implemented in some versions of EDR-DOS. The value should be set to 0 by formating tools and must not be changed by disk tools. If some of these bits are set, non-enabled implementations should refuse to open the file. To avoid problems with non-aware operating systems, partitions containing files larger than 4 GiB could use non-standard partition IDs to hide the partition from these operating systems. Under DR-DOS, partition IDs of secured partition types can be utilized for this purpose. Files larger than 4 GiB should have the Hidden, Read-only and System attributes set to hide them from normal directory searches on non-aware operating systems, similar to password protected files under DR-DOS. While FAT+ implementations do not rely on these attributes being set, for FAT+ large files they may ignore these attributes in file searches and when opening large files for modification and instead treat the System attribute as an alternative combined Read-only+Hidden attribute in this scenario.

FAT32.IFS is critically conflictive with FAT32+ revision 2.

0x0D 1

- First character of a deleted file under Novell DOS, OpenDOS and DR-DOS 7.02 and higher. A value of `0xE5` (229), as set by DELPURGE, will prohibit undeletion by UNDELETE, a value of `0x00` will allow conventional undeletion asking the user for the missing first filename character.[71] S/DOS 1 and PTS-DOS 6.51 and higher also support this feature if enabled with `SAVENAME=ON` in CONFIG.SYS. For the same feature in MSX-DOS, see offset `0x0C`.
- Create time, fine resolution: 10 ms units, values from 0 to 199 (since DOS 7.0 with VFAT).

Double usage for create time ms and file char is not conflictive, since the creation time is no longer important for deleted files.

0x0E 2

- Under DR DOS 3.31 and higher including PalmDOS, Novell DOS and OpenDOS[72] as well as under Concurrent DOS, Multiuser DOS, System Manager, and REAL/32 and possibly also under FlexOS, 4680 OS, 4690 OS any non-zero value indicates the password hash of a protected file, directory or volume label.[71] The hash is calculated from the first eight characters of a password. If the file operation to be carried out requires a password as per the access rights bitmap stored at offset `0x14`, the system tries to match the hash against the hash code of the currently set global password (by `PASSWORD /G`) or, if this fails, tries to extract a semicolon-appended password from the filespec passed to the operating system and checks it against the hash code stored here. A set password will be preserved even if a file is deleted and later undeleted.[71]
- Create time (since DOS 7.0 with VFAT). The hour, minute and second are encoded according to the following bitmap:

Bits	Description
15-11	Hours (0-23)
10-5	Minutes (0-59)
4-0	Seconds/2 (0-29)

The *seconds* is recorded only to a 2 second resolution. Finer resolution for file creation is found at offset `0x0D`.

If bits 15-11 > 23 or bits 10-5 > 59 or bits 4-0 > 29 here, or when bits 12-0 at offset `0x14` hold an access bitmap and this is not a FAT32 volume or a volume using OS/2 Extended Attributes, then this entry actually holds a password hash, otherwise it can be assumed to be a file creation time.

0x10 2

- FlexOS, 4680 OS and 4690 OS store a record size in the word at entry `0x10`.[72] This is mainly used for their special database-like file types *random file*, *direct file*, *keyed file*, and *sequential file*. If the record size is set to 0 (default) or 1, the operating systems assume a record granularity of 1 byte for the file, for which it will not perform record boundary checks in read/write operations.[41]
- With DELWATCH 2.00 and higher under Novell DOS 7, OpenDOS 7.01 and DR-DOS 7.02 and higher, this entry is used to store the last modified time stamp for pending delete files and directories.[71][72] Cleared when file is undeleted or purged. See offset `0x0E` for a format description.
- Create date (since DOS 7.0 with VFAT). The year, month and day are encoded according to the following bitmap:

Bits	Description
15-9	Year (0 = 1980, 119 = 2099 supported under DOS/Windows, theoretically up to 127 = 2107)
8-5	Month (1–12)
4-0	Day (1–31)

The usage for creation date for existing files and last modified time for deleted files is not conflictive because they are never used at the same time. For the same reason, the usage for the record size of existing files and last modified time of deleted files is not conflictive as well. Creation dates and record sizes cannot be used at the same time, however, both are stored only on file creation and never changed later on, thereby limiting the conflict to FlexOS, 4680 OS and 4690 OS systems accessing files created under foreign operating systems as well as potential display or file sorting problems on systems trying to interpret a record size as creation time. To avoid the conflict, the storage of creation dates should be an optional feature of operating systems supporting it.

0x12 2

- FlexOS, 4680 OS, 4690 OS, Multiuser DOS, System Manager, REAL/32 and DR DOS 6.0 and higher with multi-user security enabled use this field to store owner IDs.[71] Offset `0x12` holds the user ID, `0x13` the group ID of a file's creator.[72]

  In multi-user versions, system access requires a logon with account name and password, and the system assigns group and user IDs to running applications according to the previously set up and stored authorization info and inheritance rules. For 4680 OS and 4690 OS, group ID 1 is reserved for the system, group ID 2 for vendor, group ID 3 for the default user group. Background applications started by users have a group ID 2 and user ID 1, whereas operating system background tasks have group IDs 1 or 0 and user IDs 1 or 0. IBM 4680 BASIC and applications started as primary or secondary always get group ID 2 and user ID 1. When applications create files, the system will store their user ID and group ID and the required permissions with the file.[41]
- With DELWATCH 2.00 and higher under Novell DOS 7, OpenDOS 7.01 and DR-DOS 7.02 and higher, this entry is used to store the last modified date stamp for pending delete files and directories.[71][72] Cleared when file is undeleted or purged. See offset `0x10` for a format description.
- Last access date (since DOS 7.0 if ACCDATE enabled in CONFIG.SYS for the corresponding drive);[3][71] see offset `0x10` for a format description.

The usage for the owner IDs of existing files and last modified date stamp for deleted files is not conflictive because they are never used at the same time.[71] The usage of the last modified date stamp for deleted files and access date is also not conflictive since access dates are no longer important for deleted files, however, owner IDs and access dates cannot be used at the same time.

0x14 2

- Access rights bitmap for world/group/owner read/write/execute/delete protection for password protected files, directories (or volume labels) under DR DOS 3.31 and higher, including PalmDOS, Novell DOS and OpenDOS,[72] and under FlexOS,[72] 4680 OS, 4690 OS, Concurrent DOS, Multiuser DOS, System Manager, and REAL/32.

  Typical values stored on a single-user system are `0x0000` (`PASSWORD /N` for all access rights "RWED"), `0x0111` (`PASSWORD /D` for access rights "RW?-"), `0x0555` (`PASSWORD /W` for access rights "R-?-") and `0x0DDD` (`PASSWORD /R` for files or `PASSWORD /P` for directories for access rights "--?-").[71] Bits 1, 5, 9, 12-15 will be preserved when changing access rights. If execute bits are set on systems other than FlexOS, 4680 OS or 4690 OS, they will be treated similar to read bits. (Some versions of PASSWORD allow to set passwords on volume labels (`PASSWORD /V`) as well.)
  Single-user systems calculate the most restrictive rights of the three sets (DR DOS up to 5.0 used bits 0-3 only) and check if any of the requested file access types requires a permission and if a file password is stored.[71] If not, file access is granted. Otherwise the stored password is checked against an optional global password provided by the operating system and an optional file password provided as part of the filename separated by a semicolon (not under FlexOS, 4680 OS, 4690 OS). If neither of them is provided, the request will fail. If one of them matches, the system will grant access (within the limits of the normal file attributes, that is, a read-only file can still not be opened for write this way), otherwise fail the request.[71]
  Under FlexOS, 4680 OS and 4690 OS the system assigns group and user IDs to applications when launched. When they request file access, their group and user IDs are compared with the group and user IDs of the file to be opened. If both IDs match, the application will be treated as file owner. If only the group ID matches, the operating system will grant group access to the application, and if the group ID does not match as well, it will grant world access. If an application's group ID and user ID are both 0, the operating system will bypass security checking. Once the permission class has been determined, the operating system will check if any of the access types of the requested file operation requires a permission according to the stored bitflags of the selected class owner, group or world in the file's directory entry. Owner, group and world access rights are independent and do not need to have diminishing access levels. Only, if none of the requested access types require a permission, the operating system will grant access, otherwise it fails.
  If multiuser file / directory password security is enabled the system will not fail at this stage but perform the password checking mechanism for the selected permission class similar to the procedure described above. With multi-user security loaded many utilities since DR DOS 6.0 will provide an additional `/U:name` parameter.[71]

  File access rights bitmap:[76]

Bit	Mask	Description
0	0x0001	Owner delete/rename/attribute change requires permission[71][72][76]
1	0x0002	Owner execute requires permission (FlexOS, 4680 OS, 4690 OS only)[76]
2	0x0004	Owner write/modify requires permission[71][72][76]
3	0x0008	Owner read/copy requires permission[71][72][76]
4	0x0010	Group delete/rename/attribute change requires permission[71][72][76]
5	0x0020	Group execute requires permission (FlexOS, 4680 OS, 4690 OS only)[76]
6	0x0040	Group write/modify requires permission[71][72][76]
7	0x0080	Group read/copy requires permission[71][72][76]
8	0x0100	World delete/rename/attribute change requires permission[71][72][76]
9	0x0200	World execute requires permission (FlexOS, 4680 OS, 4690 OS only)[76]
10	0x0400	World write/modify requires permission[71][72][76]
11	0x0800	World read/copy requires permission[71][72][76]
12-15		bits must be set to 0 during format and must not be modified by disk tools later on;[71] bit 15 is used internally,[76] but not on disk

  File renames require either write or delete rights, IBM 4680 BASIC CHAIN requires execute rights.

- Extended Attributes handle (used by OS/2 1.2 and higher as well as by Windows NT) in FAT12 and FAT16; first cluster of EA file or 0, if not used.[37][71] A different method to store extended attributes has been devised for FAT32 volumes, see FAT32.IFS under offset `0x0C`.
- High two bytes of first cluster number in FAT32; with the low two bytes stored at offset `0x1A`.

The storage of the high two bytes of the first cluster in a file on FAT32 is partially conflictive with access rights bitmaps.

0x16 2

- Last modified time (since DOS 1.1); see offset `0x0E` for a format description.
- Under Novell DOS, OpenDOS and DR-DOS 7.02 and higher, this entry holds the deletion time of pending delete files or directories under DELWATCH 2.00 or higher. The last modified time stamp is copied to `0x10` for possible later restoration.[71] See offset `0x0E` for a format description.

0x18 2

- Last modified date; see offset `0x10` for a format description.
- Under Novell DOS, OpenDOS and DR-DOS 7.02 and higher, this entry holds the deletion date of pending delete files or directories under DELWATCH 2.00 or higher. The last modified date stamp is copied to `0x12` for possible later restoration.[71] See offset `0x10` for a format description.

0x1A 2

Start of file in clusters in FAT12 and FAT16. Low two bytes of first cluster in FAT32; with the high two bytes stored at offset `0x14`.

Entries with the Volume Label flag, subdirectory ".." pointing to FAT12/FAT16 root, and empty files with size 0 should have first cluster 0.

VFAT LFN entries also have this entry set to 0; on FAT12 and FAT16 volumes this can be used as part of a detection mechanism to distinguish between pending delete files under DELWATCH and VFAT LFNs; see above.

0x1C 4

File size in bytes. Entries with the Volume Label or Subdirectory flag set should have a size of 0.

VFAT LFN entries never store the value `0x00000000` here. This can be used as part of a detection mechanism to distinguish between pending delete files under DELWATCH and VFAT LFNs; see above.

		For files larger than 4 GiB following the FAT+ proposal, this entry only holds the size of the last chunk of the file (that is bits 31-0). The most significant bits 37-32 are stored in the entry at offset `0x0C`.[2]

The FlexOS-based operating systems IBM 4680 OS and IBM 4690 OS support unique distribution attributes stored in some bits of the previously reserved areas in the directory entries:[77]

1. Local: Don't distributed file but keep on local controller only.[nb 14]
2. Mirror file on update: Distribute file to server only when file is updated.
3. Mirror file on close: Distribute file to server only when file is closed.
4. Compound file on update: Distribute file to all controllers when file is updated.
5. Compound file on close: Distribute file to all controllers when file is closed.[78]

Some incompatible extensions found in some operating systems include:

Byte Offset	Length (bytes)	System	Description
0x0C	2	RISC OS	File type, `0x0000–0x0FFF`
0x0C	4	Petrov DOSFS (http://mdfs.net/Apps/Filing/DOSFS)	File load address
0x0E	2	ANDOS	File address in the memory
0x10	4	Petrov DOSFS (http://mdfs.net/Apps/Filing/DOSFS)	File execution address

VFAT long file names

VFAT Long File Names (LFN) are stored on a FAT file system using a trick—adding (possibly multiple) additional entries into the directory before the normal file entry. The additional entries are marked with the Volume Label, System, Hidden, and Read Only attributes (yielding `0x0F`), which is a combination that is not expected in the MS-DOS environment, and therefore ignored by MS-DOS programs and third-party utilities. Notably, a directory containing only volume labels is considered as empty and is allowed to be deleted; such a situation appears if files created with long names are deleted from plain DOS. This method is very similar to the DELWATCH method to utilize the volume attribute to hide pending delete files for possible future undeletion since DR DOS 6.0 (1991) and higher.

Because older versions of DOS could mistake LFN names in the root directory for the volume label, VFAT was designed to create a blank volume label in the root directory before adding any LFN name entires (if a volume label did not already exist).[nb 13]

Each phony entry can contain up to 13 UCS-2 characters (26 bytes) by using fields in the record which contain file size or time stamps (but not the starting cluster field, for compatibility with disk utilities, the starting cluster field is set to a value of 0. See 8.3 filename for additional explanations). Up to 20 of these 13-character entries may be chained, supporting a maximum length of 255 UCS-2 characters.[74]

After the last UCS-2 character, a `0x0000` is added. The remaining unused characters are filled with `0xFFFF`.

LFN entries use the following format:

Byte Offset	Length (bytes)	Description
0x00	1	Sequence Number (bit 6: last logical, first physical LFN entry, bit 5: 0; bits 4-0: number `0x01..0x14` (`0x1F`), deleted entry: `0xE5`)
0x01	10	Name characters (five UCS-2 characters)
0x0B	1	Attributes (always `0x0F`)
0x0C	1	Type (always `0x00` for VFAT LFN, other values reserved for future use; for special usage of bits 4 and 3 in SFNs see below)
0x0D	1	Checksum of DOS file name
0x0E	12	Name characters (six UCS-2 characters)
0x1A	2	First cluster (always `0x0000`)
0x1C	4	Name characters (two UCS-2 characters)

If there are multiple LFN entries, required to represent a file name, firstly comes the *last* LFN entry (the last part of the filename). The sequence number also has bit 6 (`0x40`) set (this means the last LFN entry, however it's the first entry seen when reading the directory file). The last LFN entry has the largest sequence number which decreases in following entries. The *first* LFN entry has sequence number 1. A value of `0xE5` is used to indicate that the entry is deleted.

On FAT12 and FAT16 volumes, testing for the values at `0x1A` to be zero and at `0x1C` to be non-zero can be used to distinguish between VFAT LFNs and pending delete files under DELWATCH.

For example if we have filename "File with very long filename.ext" it would be formatted like this:

Sequence number	Entry data
0x43	"me.ext"
0x02	"y long filena"
0x01	"File with ver"
???	Normal 8.3 entry

A checksum also allows verification of whether a long file name matches the 8.3 name; such a mismatch could occur if a file was deleted and re-created using DOS in the same directory position. The checksum is calculated using the algorithm below. (Note that pFCBName is a pointer to the name as it appears in a regular directory entry, i.e. the first eight characters are the filename, and the last three are the extension. The dot is implicit. Any unused space in the filename is padded with space characters (ASCII `0x20`). For example, "Readme.txt" would be "`README␣␣TXT`".)

```
unsigned char lfn_checksum(const unsigned char *pFCBName)
{
   int i;
   unsigned char sum = 0;

   for (i = 11; i; i--)
      sum = ((sum & 1) << 7) + (sum >> 1) + *pFCBName++;

   return sum;
}
```

If a filename contains only lowercase letters, or is a combination of a lowercase *basename* with an uppercase *extension*, or vice-versa; and has no special characters, and fits within the 8.3 limits, a VFAT entry is not created on Windows NT and later versions of Windows such as XP. Instead, two bits in byte `0x0C` of the directory entry are used to indicate that the filename should be considered as entirely or partially lowercase. Specifically, bit 4 means lowercase *extension* and bit 3 lowercase *basename*, which allows for combinations such as "`example.TXT`" or "`HELLO.txt`" but not "`Mixed.txt`". Few other operating systems support it. This creates a backwards-compatibility problem with older Windows versions (Windows 95 / 98 / 98 SE / ME) that see all-uppercase filenames if this extension has been used, and therefore can change the name of a file when it is transported between operating systems, such as on a USB flash drive. Current 2.6.x versions of Linux will recognize this extension when reading (source: kernel 2.6.18 `/fs/fat/dir.c` and `fs/vfat/namei.c`); the mount option `shortname` determines whether this feature is used when writing.[79]

Size limits

The FAT12, FAT16, FAT16B, and FAT32 variants of the FAT file systems have clear limits based on the number of clusters and the number of sectors per cluster (1, 2, 4, ..., 128). For the typical value of 512 bytes per sector:

```
FAT12 requirements : 3 sectors on each copy of FAT for every 1,024 clusters
FAT16 requirements : 1 sector on each copy of FAT for every 256 clusters
FAT32 requirements : 1 sector on each copy of FAT for every 128 clusters

FAT12 range : 1 to 4,084 clusters : 1 to 12 sectors per copy of FAT
FAT16 range : 4,085 to 65,524 clusters : 16 to 256 sectors per copy of FAT
FAT32 range : 65,525 to 268,435,444 clusters : 512 to 2,097,152 sectors per copy of FAT

FAT12 minimum : 1 sector per cluster × 1 clusters = 512 bytes (0.5 KiB)
FAT16 minimum : 1 sector per cluster × 4,085 clusters = 2,091,520 bytes (2,042.5 KiB)
FAT32 minimum : 1 sector per cluster × 65,525 clusters = 33,548,800 bytes (32,762.5 KiB)

FAT12 maximum : 64 sectors per cluster × 4,084 clusters = 133,824,512 bytes (≈ 127 MiB)
[FAT12 maximum : 128 sectors per cluster × 4,084 clusters = 267,694,024 bytes (≈ 255 MiB)]

FAT16 maximum : 64 sectors per cluster × 65,524 clusters = 2,147,090,432 bytes (≈2,047 MiB)
[FAT16 maximum : 128 sectors per cluster × 65,524 clusters = 4,294,180,864 bytes (≈4,095 MiB)]

FAT32 maximum : 8 sectors per cluster × 268,435,444 clusters = 1,099,511,578,624 bytes (≈1,024 GiB)
FAT32 maximum : 16 sectors per cluster × 268,173,557 clusters = 2,196,877,778,944 bytes (≈2,046 GiB)
[FAT32 maximum : 32 sectors per cluster × 134,152,181 clusters = 2,197,949,333,504 bytes (≈2,047 GiB)]
[FAT32 maximum : 64 sectors per cluster × 67,092,469 clusters = 2,198,486,024,192 bytes (≈2,047 GiB)]
[FAT32 maximum : 128 sectors per cluster × 33,550,325 clusters = 2,198,754,099,200 bytes (≈2,047 GiB)]
```

> **Legend:** 268435444+3 is `0x0FFFFFF7`, because FAT32 version 0 uses only 28 bits in the 32-bit cluster numbers, cluster numbers `0x0FFFFFF7` up to `0x0FFFFFFF` flag bad clusters or the end of a file, cluster number 0 flags a free cluster, and cluster number 1 is not used.[22] Likewise 65524+3 is `0xFFF7` for FAT16, and 4084+3 is `0xF7` for FAT12. The number of sectors per cluster is a power of 2 fitting in a single byte, the smallest value is 1 (`0x01`), the biggest value is 128 (`0x80`). Lines in square brackets indicate the unusual cluster size 128, and for FAT32 the bigger than necessary cluster sizes 32 or 64.[34]

Because each FAT32 entry occupies 32 bits (4 bytes) the maximal number of clusters (268435444) requires 2097152 FAT sectors for a sector size of 512 bytes. 2097152 is `0x200000`, and storing this value needs more than two bytes. Therefore FAT32 introduced a new 32-bit value in the FAT32 boot sector immediately following the 32-bit value for the total number of sectors introduced in the FAT16B variant.

The boot record extensions introduced with DOS 4.0 start with a magic 40 (`0x28`) or 41 (`0x29`). Typically FAT drivers look only at the number of clusters to distinguish FAT12, FAT16, and FAT32: the human readable strings identifying the FAT variant in the boot record are ignored, because they exist only for media formatted with DOS 4.0 or later.

Determining the number of directory entries per cluster is straight forward, each entry occupies 32 bytes, this results in 16 entries per sector for a sector size of 512 bytes. The DOS 5 `RMDIR`/`RD` command removes the initial "`.`" (this directory) and "`..`" (parent directory) entries in subdirectories directly, therefore sector size 32 on a RAM disk is possible for FAT12, but requires 2 or more sectors per cluster. A FAT12 boot sector without the DOS 4 extensions needs 29 bytes before the first unnecessary FAT16B 32-bit number of hidden sectors, this leaves three bytes for the (on a RAM disk unused) boot code and the magic `0x55 0xAA` at the end of all boot sectors. On Windows NT the smallest supported sector size is 128.

On Windows NT operating systems the FORMAT command options `/A:128K` and `/A:256K` correspond to the maximal cluster size `0x80` (128) with a sector size 1024 and 2048, respectively. For the common sector size 512 `/A:64K` yields 128 sectors per cluster.

Both editions of each ECMA-107[5] and ISO/IEC 9293[6][7] specify a *Max Cluster Number* `MAX` determined by the formula `MAX=1+trunc((TS-SSA)/SC)`, and reserve cluster numbers `MAX+1` up to 4086 (`0xFF6`, FAT12) and later 65526 (`0xFFF6`, FAT16) for future standardization.

Microsoft's EFI FAT32 specification[19] states that any FAT file system with less than 4085 clusters is FAT12, else any FAT file system with less than 65525 clusters is FAT16, and otherwise it is FAT32. The entry for cluster 0 at the beginning of the FAT must be identical to the media descriptor byte found in the BPB, whereas the entry for cluster 1 reflects the end-of-chain value used by the formatter for cluster chains (`0xFFF`, `0xFFFF` or `0x0FFFFFFF`). The entries for cluster numbers 0 and 1 end at a byte boundary even for FAT12, e.g., `0xF9FFFF` for media descriptor `0xF9`.

The first data cluster is 2,[22] and consequently the last cluster `MAX` gets number `MAX+1`. This results in data cluster numbers 2...4085 (`0xFF5`) for FAT12, 2...65525 (`0xFFF5`) for FAT16, and 2...268435445 (`0x0FFFFFF5`) for FAT32.

The only available values reserved for future standardization are therefore `0xFF6` (FAT12) and `0xFFF6` (FAT16). As noted below "less than 4085" is also used for Linux implementations,[20] or as Microsoft's FAT specification puts it:[19]

> *when it says <, it does not mean <=. Note also that the numbers are correct. The first number for FAT12 is 4085; the second number for FAT16 is 65525. These numbers and the '<' signs are not wrong.*

Fragmentation

The FAT file system does not contain built-in mechanisms which prevent newly written files from becoming scattered across the partition.[80] On volumes where files are created and deleted frequently or their lengths often changed, the medium will become increasingly fragmented over time.

While the design of the FAT file system does not cause any organizational overhead in disk structures or reduce the amount of free storage space with increased amounts of fragmentation, as it occurs with external fragmentation, the time required to read and write fragmented files will increase as the operating system will have to follow the cluster chains in the FAT (with parts having to be loaded into memory first in particular on large volumes) and read the corresponding data physically scattered over the whole medium reducing chances for the low-level block device driver to perform multi-sector disk I/O or initiate larger DMA transfers, thereby effectively increasing I/O protocol overhead as well as arm movement and head settle times inside the disk drive. Also, file operations will become slower with growing fragmentation as it takes increasingly longer for the operating system to find files or free clusters.

Other file systems, e.g., HPFS or exFAT, use free space bitmaps that indicate used and available clusters, which could then be quickly looked up in order to find free contiguous areas. Another solution is the linkage of all free clusters into one or more lists (as is done in Unix file systems). Instead, the FAT has to be scanned as an array to find free clusters, which can lead to performance penalties with large disks.

In fact, seeking for files in large subdirectories or computing the free disk space on FAT volumes is one of the most resource intensive operations, as it requires reading the directory tables or even the entire FAT linearly. Since the total amount of clusters and the size of their entries in the FAT was still small on FAT12 and FAT16 volumes, this could still be tolerated on FAT12 and FAT16 volumes most of the time, considering that the introduction of more sophisticated disk structures would have also increased the complexity and memory footprint of real-mode operating systems with their minimum total memory requirements of 128 KiB or less (such as with DOS) for which FAT has been designed and optimized originally.

With the introduction of FAT32, long seek and scan times became more apparent, particularly on very large volumes. A possible justification suggested by Microsoft's Raymond Chen for limiting the maximum size of FAT32 partitions created on Windows was the time required to perform a "`DIR`" operation, which always displays the free disk space as the last line.[81] Displaying this line took longer and longer as the number of clusters increased. FAT32 therefore introduced a special file system information sector where the previously computed amount of free space is preserved over power cycles, so that the free space counter needs to be recalculated only when a removable FAT32 formatted medium gets ejected without first unmounting it or if the system is switched off without properly shutting down the operating system, a problem mostly visible with pre-ATX-style PCs, on plain DOS systems and some battery-powered consumer products.

With the huge cluster sizes (16 KiB, 32 KiB, 64 KiB) forced by larger FAT partitions, internal fragmentation in form of disk space waste by file slack due to cluster overhang (as files are rarely exact multiples of cluster size) starts to be a problem as well, especially when there are a great many small files.

Various optimizations and tweaks to the implementation of FAT file system drivers, block device drivers and disk tools have been devised to overcome most of the performance bottlenecks in the file system's inherit design without having to change the layout of the on-disk structures. They can be divided into on-line and off-line methods and work by trying to avoid fragmentation in the file system in the first place, deploying methods to better cope with existing fragmentation, and by reordering and optimizing the on-disk structures. With optimizations in place, the performance on FAT volumes can often reach that of more sophisticated file systems in practical scenarios, while at the same time retaining the advantage of being accessible even on very small or old systems.

DOS 3.0 and higher will not immediately reuse disk space of deleted files for new allocations but instead seek for previously unused space before starting to use disk space of previously deleted files as well. This not only helps to maintain the integrity of deleted files for as long as possible but also speeds up file allocations and avoids fragmentation, since never before allocated disk space is always unfragmented. DOS accomplishes this by keeping a pointer to the last allocated cluster on each mounted volume in memory and starts searching for free space from this location upwards instead of at the beginning of the FAT, as it was still done by DOS 2.x.[53] If the end of the FAT is reached, it would wrap around to continue the search at the beginning of the FAT until either free space has been found or the original position has been reached again without having found free space.[53] These pointers are initialized to point to the start of the FATs after bootup,[53] but on FAT32 volumes, DOS 7.1 and higher will attempt to retrieve the last position from the FS Information Sector. This mechanism is defeated, however, if an application often deletes and recreates temporary files as the operating system would then try to maintain the integrity of void data effectively causing more fragmentation in the end.[53] In some DOS versions, the usage of a special API function to create temporary files can be used to avoid this problem.

Additionally, directory entries of deleted files will be marked `0xE5` since DOS 3.0.[12] DOS 5.0 and higher will start to reuse these entries only when previously unused directory entries have been used up in the table and the system would otherwise have to expand the table itself.[21]

Since DOS 3.3 the operating system provides means to improve the performance of file operations with FASTOPEN by keeping track of the position of recently opened files or directories in various forms of lists (MS-DOS/PC DOS) or hash tables (DR-DOS), which can reduce file seek and open times significantly. Before DOS 5.0 special care must be taken when using such mechanisms in conjunction with disk defragmentation software bypassing the file system or disk drivers.

Windows NT will allocate disk space to files on FAT in advance, selecting large contiguous areas, but in case of a failure, files which were being appended will appear larger than they were ever written into, with a lot of random data at the end.

Other high-level mechanisms may read in and process larger parts or the complete FAT on startup or on demand when needed and dynamically build up in-memory tree representations of the volume's file structures different from the on-disk structures. This may, on volumes with many free clusters, occupy even less memory than an image of the FAT itself. In particular on highly fragmented or filled volumes, seeks become much faster than with linear scans over the actual FAT, even if an image of the FAT would be stored in memory. Also, operating on the logically high level of files and cluster-chains instead of on sector or track level, it becomes possible to avoid some degree of file fragmentation in the first place or to carry out local file defragmentation and reordering of directory entries based on their names or access patterns in the background.

Some of the perceived problems with fragmentation of FAT file systems also result from performance limitations of the underlying block device drivers, which becomes more visible the lesser memory is available for sector buffering and track blocking/deblocking:

While the single-tasking DOS had provisions for multi-sector reads and track blocking/deblocking, the operating system and the traditional PC hard disk architecture (only one outstanding input/output request at a time and no DMA transfers) originally did not contain mechanisms which could alleviate fragmentation by asynchronously prefetching next data while the application was processing the previous chunks. Such features became available later. Later DOS versions also provided built-in support for look-ahead sector buffering and came with dynamically loadable disk caching programs working on physical or logical sector level, often utilizing EMS or XMS memory and sometimes providing adaptive caching strategies or even run in protected mode through DPMS or Cloaking to increase performance by gaining direct access to the cached data in linear memory rather than through conventional DOS APIs.

Write-behind caching was often not enabled by default with Microsoft software (if present) given the problem of data loss in case of a power failure or crash, made easier by the lack of hardware protection between applications and the system.

Legal issues

Licensing

Microsoft applied for, and was granted, a series of patents for key parts of the FAT file system in the mid-1990s. Being almost universally compatible and well-understood, FAT is frequently chosen as an interchange format for flash media used in digital cameras and PDAs.

On December 3, 2003 Microsoft announced[82] it would be offering licenses for use of its FAT specification and "associated intellectual property", at the cost of a US$0.25 royalty per unit sold, with a $250,000 maximum royalty per license agreement.[83] To this end, Microsoft cited four patents on the FAT file system as the basis of its intellectual property claims. All four pertain to long-filename extensions to FAT first seen in Windows 95:

- U.S. Patent 5,745,902 (http://www.google.com/patents/US5745902): Method and system for accessing a file using file names having different file name formats. Filed July 6, 1992. This covered a means of generating and associating a short, 8.3 filename with long one (for example, "Microsoft.txt" with "MICROS~1.TXT") and a means of enumerating conflicting short filenames (for example, "MICROS~2.TXT" and "MICROS~3.TXT").
- U.S. Patent 5,579,517 (http://www.google.com/patents/US5579517): Common name space for long and short filenames. Filed for on 1995-04-24. This covers the method of chaining together multiple consecutive 8.3 named directory entries to hold long filenames, with some of the entries specially marked to prevent their confusing older, long filename-unaware FAT implementations.
  - The Public Patent Foundation successfully challenged this patent; the claims were rejected[84] on 2004-09-14, due to prior disclosure[85] of the claimed techniques in patents U.S. Patent 5,307,494 (http://www.google.com/patents/US5307494) and U.S. Patent 5,367,671 (http://www.google.com/patents/US5367671). This decision was later overturned by the Patent Office on 2006-01-10.
  - The patent fails to declare prior art in the form of detailed knowledge of the method to utilize the volume attribute to hide pending delete files for possible future undeletion as used by DELWATCH in DR DOS 6.0 (1991) and higher in the same way as LFN entries get hidden on VFAT volumes.
- U.S. Patent 5,758,352 (http://www.google.com/patents/US5758352): Common name space for long and short filenames. Filed on 1996-09-05. This is very similar to 5,579,517.
  - The Public Patent Foundation successfully challenged this patent (USPTO); The USPTO rejected this patent on 2005-10-05, on the grounds that "the six assignees

names were incorrect".[86][87] This decision was also later overturned by the Patent Office on 2006-01-10.

- U.S. Patent 6,286,013 (http://www.google.com/patents/US6286013): Method and system for providing a common name space for long and short file names in an operating system. Filed on 1997-01-28. This makes claims on the methods used when Windows 95, Windows 98 and Windows ME expose long filenames to their MS-DOS compatibility layer. It does not appear to affect any non-Microsoft FAT implementations.[*citation needed*]

In the EFI FAT32 specification[19] Microsoft specifically grants a number of rights, which many readers have interpreted as permitting operating system vendors to implement FAT.[88]

Microsoft is not the only company to have applied for patents for parts of the FAT file system. Other patents affecting FAT include:

- U.S. Patent 5,367,671 (http://www.google.com/patents/US5367671): System for accessing extended object attribute (EA) data through file name or EA handle linkages in path tables. Filed on 1990-09-25 by Barry A. Feigenbaum and Felix Miro of IBM, this makes claims on the methods used by OS/2, Windows NT, and Linux for storing extended attribute data in the "EA DATA. SF" file. As a USA patent filed before 1995, it would have expired after the later of the filing date plus 20 years, or the issue date plus 17 years. The patent was issued on 1994-11-24, so it expired in 2011.

Challenge

The Public Patent Foundation (PUBPAT) submitted evidence to the US Patent and Trade Office (USPTO) disputing the validity of these patents, including prior art references from Xerox and IBM. The USPTO acknowledged that the evidence raised "substantial new question[s] of patentability," and opened an investigation into the validity of Microsoft's FAT patents.[89] On 2004-09-30 the USPTO rejected all claims of U.S. Patent 5,579,517 (http://www.google.com/patents/US5579517), based primarily on evidence provided by PUBPAT.

On 2005-10-05 the USPTO announced that, following the re-examination process, it had again rejected all claims of patent 5,579,517, and it additionally found U.S. Patent 5,758,352 (http://www.google.com/patents/US5758352) invalid on the grounds that the patent had incorrect assignees.

However, on 2006-01-10 the USPTO ruled that features of Microsoft's implementation of the FAT system were "novel and non-obvious", reversing both earlier non-final decisions.[90]

Patent infringement lawsuits

In February 2009, Microsoft filed a patent infringement lawsuit against TomTom alleging that the device maker's products infringe on patents related to VFAT long filenames. As some TomTom products are based on Linux, this marked the first time that Microsoft tried to enforce its patents against the Linux platform.[91] The lawsuit was settled out of court the following month with an agreement that Microsoft be given access to four of TomTom's patents, that TomTom will drop support for the VFAT long filenames from its products, and that in return Microsoft not seek legal action against TomTom for the five year duration of the settlement agreement.[92]

In October 2010, Microsoft filed a patent infringement lawsuit against Motorola alleging several patents (including two of the VFAT patents) were not licensed for use in the Android operating system.[93] They also submitted a complaint to the ITC.[94] Developers of open source software have designed methods intended to circumvent Microsoft's patents.[95]

See also

- Comparison of file systems
- Drive letter assignment
- List of file systems
- Transaction-Safe FAT File System

Notes

1. **^** Sources differ in regard to the first NCR data entry terminal integrating support for the FAT filesystem. According to Stephen Manes and Paul Andrews, "Gates", development was for a NCR 8200 in late 1977, according to them a floppy-based upgrade to the NCR 7200, which was cassette-based (and is mentioned in other sources). Marc McDonald remembers a NCR 8500. Another possible candidate is the 8080-based NCR 7500, which is known to have used 8"-floppies and was released in 1978.
2. **^** ***a*** ***b*** ***c*** ***d*** ***e*** See other links for special precautions in regard to occurrences of a cluster value of `0xFF0` on FAT12 volumes under MS-DOS/PC DOS 3.3 and higher.
3. **^** DR-DOS is able to boot off FAT12/FAT16 logical sectored media with logical sector sizes up to 1024 bytes.
4. **^** A driver named VFAT appeared before Windows 95, in Windows for Workgroups 3.11, but this older version was only used for implementing 32-bit file access and did not support long file names.
5. **^** ***a*** ***b*** This is the reason, why `0xE5` had a special meaning in directory entries.
6. **^** ***a*** ***b*** ***c*** One utility providing an option to specify the desired format filler value for hard disks is DR-DOS' FDISK R2.31 with its optional wipe parameter `/W:246`. In contrast to other FDISK utilities, DR-DOS FDISK is not only a partitioning tool, but can also format freshly created partitions as FAT12, FAT16 or FAT32. This reduces the risk to accidentally format wrong volumes.
7. **^** ***a*** ***b*** ***c*** For maximum compatibility with MS-DOS/PC DOS and DR-DOS, operating systems trying to determine a floppy disk's format should test on all mentioned opcode sequences at sector offset `0x000` in *addition* to looking for a valid media descriptor byte at sector offset `0x015` before assuming the presence of a BPB. Although PC DOS 1.0 floppy disks do not contain a BPB, they start with `0xEB` as well, but do not show a `0x90` at offset `0x002`. PC DOS 1.10 floppy disks even start with `0xEB 0x?? 0x90`, although they still do not feature a BPB. In both cases, a test for a valid media descriptor at offset `0x015` would fail (value `0x00` instead of valid media descriptors `0xF0` and higher). If these tests fail, DOS checks for the presence of a media descriptor byte in the first byte of the first FAT in the sector following the boot sector (logical sector 1 on FAT12/FAT16 floppies).
8. **^** ***a*** ***b*** ***c*** ***d*** ***e*** The signature at offset `0x1FE` in boot sectors is `0x55 0xAA`, that is `0x55` at offset `0x1FE` and `0xAA` at offset `0x1FF`. Since little-endian representation must be assumed in the context of IBM PC compatible machines, this can be written as 16-bit word `0xAA55` in programs for x86 processors (note the swapped order), whereas it would have to be written as `0x55AA` in programs for other CPU architectures using a big-endian representation. Since this has been mixed up numerous times in books and even in original Microsoft reference documents, this article uses the offset-based byte-wise on-disk representation to avoid any possible misinterpretation.
9. **^** ***a*** ***b*** ***c*** The checksum entry in Atari boot sectors holds the alignment value, not the magic value itself. The magic value `0x1234` is not stored anywhere on disk. In contrast to Intel x86 processors, the Motorola 680x0 processors as used in Atari machines use a big-endian memory representation and therefore a big-endian representation must be assumed when calculating the checksum. As a consequence of this, for checksum verification code running on x86 machines, pairs of bytes must be swapped before the 16-bit addition.
10. **^** ***a*** ***b*** The following DOS functions return these register values: INT 21h/AH=2Ah "Get system date" returned values: CX = year (1980..2099), DH = month (1..12), DL = day (1..31). INT 21h/AH=2Ch "Get system time" returned values: CH = hour (0..23), CL = minute (0..59), DH = second (0..59), DL = 1/100 seconds (0..99).
11. **^** In order to support the coexistance of DR-DOS with PC DOS and multiple parallel installations of DR-DOS, the extension of the default "`IBMBIO␣␣COM`" boot file name can be changed using the `SYS /DR:ext` option, where ext represents the new extension. Other potential DR-DOS boot file names to be expected in special scenarios are "`DRBIOS␣␣␣SYS`", "`DRDOS␣␣␣SYS`", "`IO␣␣␣␣␣␣SYS`", "`JO␣␣␣␣␣␣SYS`".
12. **^** If a volume's dirty shutdown flag is still cleared on startup, the volume was not properly unmounted. This would, for example, cause Windows 98 WIN.COM to start SCANDISK in order to check for and repair potential logical file system errors. If the bad sector flag is cleared, it will force a surface scan to be carried out as well. This can be disabled by setting AUTOSCAN=0 in the [OPTIONS] section in MSDOS.SYS file.
13. **^** ***a*** ***b*** To avoid potential misinterpretation of directory volume labels with VFAT LFN entries by non-VFAT aware operating systems, the DR-DOS 7.07 FDISK and FORMAT tools are known to explicitly write dummy "`NO␣NAME␣␣␣␣`" directory volume labels if the user skips entering a volume label. The operating system would internally default to return the same string if no directory volume label could be found in the root of a volume, but without a real volume label stored as the first entry (after the directory entries), older operating systems could erroneously pick up VFAT LFN entries instead.
14. **^** This IBM 4680 OS and 4690 OS distribution attribute type must have an on-disk bit value of 0 as files fall back to this type when attributes get lost accidentally.

References

1. **^** ***a*** ***b*** "File Systems" (http://technet.microsoft.com/en-us/library/cc938937.aspx). Microsoft TechNet. 2001. Retrieved 2011-07-31.

1. ^ **a b** "File Systems" (http://technet.microsoft.com/en-us/library/cc938937.aspx). Microsoft TechNet. 2001. Retrieved 2011-07-31.
2. ^ **a b c d e f g h i j** Udo Kuhnt, Luchezar Georgiev, Jeremy Davis (2007). *FAT+*. FATPLUS.TXT, draft revision 2 ([1] (http://www.unet.univie.ac.at/~a0503736/php/drdoswiki/index.php?n=Main.FATplus), [2] (http://www.fdos.org/kernel/fatplus.txt)).
3. ^ **a b** Microsoft (2006-11-15). Windows 95 CD-ROM CONFIG.TXT File (http://support.microsoft.com/kb/135481/EN-US) Article 135481, Revision: 1.1, retrieved 2011-12-22: "For each hard disk, specifies whether to record the date that files are last accessed. Last access dates are turned off for all drives when your computer is started in safe mode, and are not maintained for floppy disks by default. Syntax: `ACCDATE=drive1+|- [drive2+|-]...`"
4. ^ **a b** "FAT File System (Windows Embedded CE 6.0)" (http://msdn.microsoft.com/de-de/library/ee489982%28v=winembedded.60%29.aspx). Microsoft. 2010-01-06. Retrieved 2013-07-07.
5. ^ **a b c d e f g h i j k l** "Volume and File Structure of Disk Cartridges for Information Interchange" (http://www.ecma-international.org/publications/standards/Ecma-107.htm). *Standard ECMA-107 (2nd ed., June 1995)*. ECMA. 1995. Retrieved 2011-07-30.
6. ^ **a b c d e f g h i j k** "Information technology -- Volume and file structure of disk cartridges for information interchange" (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21273). *ISO/IEC 9293:1994*. ISO catalogue. 1994. Retrieved 2012-01-06.
7. ^ **a b c d e f g h i j k** "Information processing -- Volume and file structure of flexible disk cartridges for information interchange" (http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16948). *ISO 9293:1987*. ISO catalogue. 1987. Retrieved 2012-01-06.
8. ^ Aaron R. Reynolds, Dennis R. Adler, Ralph A. Lipe, Ray D. Pedrizetti, Jeffrey T. Parsons, Rasipuram V. Arun (1998-05-26). "Common name space for long and short filenames" (http://www.google.de/patents?id=bUohAAAAEBAJ). *US Patent 5758352*. Retrieved 2012-01-19.
9. ^ **a b c d e f g h i j k l m n o p q** Ray Duncan (1988). *The MS-DOS Encyclopedia - version 1.0 through 3.2*. Microsoft Press. ISBN 1-55615-049-0.
10. ^ **a b** Manes, Stephen; Paul Andrews (1993). *Gates: How Microsoft's Mogul Reinvented an Industry—and Made Himself the Richest Man in America*. Doubleday. ISBN 0-385-42075-7.
11. ^ **a b c d** David Hunter (1983). *Tim Paterson - The roots of DOS*. Softalk, *IBM Personal Computer*, March 1983 ([3] (http://www.patersontech.com/Dos/Softalk/Softalk.html)).
12. ^ **a b c d e f** Andrew Schulman, Ralf Brown, David Maxey, Raymond J. Michels, Jim Kyle (1994). *Undocumented DOS*. Addison Wesley, second edition. ISBN 0-201-63287-X, ISBN 978-0-201-63287-3.
13. ^ **a b** Tim Paterson (2007-09-30). "Design of DOS" (http://dosmandrivel.blogspot.com/2007/09/design-of-dos.html). *DosMan Drivel*. Retrieved 2011-07-04.
14. ^ Seattle Computer Products (1980-08). "86-DOS - 8086 OPERATING SYSTEM - $95" (http://archive.org/stream/byte-magazine-1980-08/1980_08_BYTE_05-08_The_Forth_Language#page/n173/mode/2up). *Byte*. 8 **5**. p. 174. Retrieved 2013-08-18. (NB. The SCP advertisment already calls the product *86-DOS*, but does not mention a specific version number. Version 0.3 is known to be called 86-DOS already, so the name change must have taken place either for version 0.2 or immediately afterwards in August 1980.)
15. ^ **a b c d** Seattle Computer Products (1981). "SCP 86-DOS 1.0 Addendum" (http://bitsavers.informatik.uni-stuttgart.de/pdf/seattleComputer/86-DOS_1.0_Addendum.pdf). Retrieved 2013-03-10.
16. ^ Wallace & Erickson, 1992. *Hard Drive*. John Wiley & Sons. ISBN 0-471-56886-4.
17. ^ **a b** Peter Norton (1986). *Inside the IBM PC, Revised and Enlarged*, Brady. ISBN 0-89303-583-1, p. 157.
18. ^ Brian Jenkinson, Sammes, A. J. (2000). *Forensic Computing: A Practitioner's Guide (Practitioner Series)*. Berlin: Springer. p. 157. ISBN 1-85233-299-9. "... only 2^12 (that is, 4096) allocation units or clusters can be addressed. In fact, the number is less than this, since 000h and 001h are not used and FF0h to FFFh are reserved or used for other purposes, leaving 002h to FEFh (2 to 4079) as the range of possible clusters."
19. ^ **a b c d e f g h i j k l m n o p q** "Microsoft Extensible Firmware Initiative FAT32 File System Specification, FAT: General Overview of On-Disk Format" (http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc). Microsoft. 2000-12-06. Retrieved 2011-07-03.
20. ^ **a b c d** Andries Brouwer. "FAT under Linux" (http://www.win.tue.nl/~aeb/linux/fs/fat/fat-2.html).
21. ^ **a b c d e f g** Geoff Chappell (1994). *DOS Internals*. Addison Wesley. ISBN 0-201-60835-9, ISBN 978-0-201-60835-9.
22. ^ **a b c d e** Tim Paterson (1983). "An Inside Look at MS-DOS" (http://www.patersontech.com/dos/Byte/InsideDos.htm#InsideDos_44). Byte. Retrieved 2011-07-18. "The numbering starts with 2; the first two numbers, 0 and 1, are reserved."
23. ^ IBM (1984). *IBM PC DOS 3.0 announcement letter*.
24. ^ IBM (1985). *IBM PC DOS Technical Reference*. First Edition, P/N 6024181, dated February 1985.
25. ^ Microsoft Knowledge Base article: "MS-DOS Partitioning Summary" (http://support.microsoft.com/kb/q69912/)
26. ^ Andries Brouwer. "List of partition identifiers for PCs" (http://www.win.tue.nl/~aeb/partitions/partition_types-1.html).
27. ^ Microsoft (2000-12-17). *Wyse DOS 3.3 Partitions Incompatible with MS-DOS 5.x and 6.x*. Document Q78407 ([4] (ftp://ftp.microsoft.com/misc1/PEROPSYS/MSDOS/KB/Q78/4/07.TXT)).
28. ^ Microsoft (2000-12-17). *Upgrading Pre-4.0 Systems with Logical Drive(s) > 32 MB*. Document Q68176 ([5] (ftp://ftp.microsoft.com/misc1/PEROPSYS/MSDOS/KB/Q68/1/76.TXT)).
29. ^ Andries Brouwer. "Properties of partition tables" (http://www.win.tue.nl/~aeb/partitions/partition_types-2.html#ss2.6).
30. ^ "Dskprobe Overview: Data Recovery" (http://technet.microsoft.com/en-us/library/cc736327(v=ws.10).aspx). Microsoft TechNet. 2003-03-28. Retrieved 2011-08-03.
31. ^ "Errors Creating Files or Folders in the Root Directory" (http://support.microsoft.com/kb/120138). Microsoft Help and Support. December 16, 2004. Retrieved 2006-10-14.
32. ^ "mkdosfs man page" (http://www.die.net/doc/linux/man/man8/mkdosfs.8.html).
33. ^ **a b c d e f** "Windows 98 Resource Kit - Chapter 10 - Disks and File Systems" (http://technet.microsoft.com/en-us/library/cc768180.aspx). Microsoft TechNet. 1998. Retrieved 2012-07-16.
34. ^ **a b** "Limitations of FAT32 File System" (http://support.microsoft.com/kb/184006). Microsoft Knowledge Base. 2007-03-26. Retrieved 2011-08-21. "Clusters cannot be 64 kilobytes (KB) or larger"
35. ^ "Limitations of the FAT32 File System in Windows XP" (http://support.microsoft.com/kb/314463). Microsoft Knowledge Base. 2007-12-01. Retrieved 2011-08-21.
36. ^ **a b** IBM. *4690 OS User's Guide Version 5.2*, IBM document SC30-4134-01, 2008-01-10 ([6] (ftp://ftp.software.ibm.com/software/retail/pubs/sw/opsys/4690/ver5r2/bsf1_UG_mst.pdf)).
37. ^ **a b** Bob Eager, Tavi Systems (2000-10-28). *Implementation of extended attributes on the FAT file system*. ([7] (http://www.tavi.co.uk/os2pages/eadata.html)).
38. ^ **a b** Henk Kelder (2003). *FAT32.TXT for FAT32.IFS version 0.9.13*." *([8] (http://svn.netlabs.org/repos/fat32/branches/fat32-0.9/src/fat32.txt)): "This byte […] is not modified while running Windows 95 and neighter by SCANDISK or DEFRAG. […] If another program sets the value to `0x00` for a file that has EAs these EAs will no longer be found using DosFindFirst/Next calls only. The other OS/2 calls for retrieving EAs (DosQueryPathInfo, DosQueryFileInfo and DosEnumAttribute) do not rely on this byte. Also the opposite could […] occur. […] In this situation only the performance of directory scans will be decreased. Both situations […] are corrected by CHKDSK*".
39. ^ **a b** Henk Kelder. *FAT32.TXT for FAT32.IFS version 0.74*. ([9] (http://macarlo.com/fat32v074.htm)). Comment: This older version of the README file still discusses the old `0xEA` and `0xEC` magic values.
40. ^ Bob Eager (2000-10-28). "Implementation of extended attributes on the FAT file system" (http://www.tavi.co.uk/os2pages/eadata.html). *Tavi OS/2 pages*. Retrieved 2006-10-14.
41. ^ **a b c** IBM. *4690 OS Programming Guide Version 5.2*, IBM document SC30-4137-01, 2007-12-06 ([10] (ftp://ftp.software.ibm.com/software/retail/pubs/sw/opsys/4690/ver5r2/bsi1_PG_mst.pdf)).
42. ^ "Release notes for v2.5.7" (http://www.kernel.org/pub/linux/kernel/v2.5/ChangeLog-2.5.7). The Linux Kernel archives. 2002-03-12. Retrieved 2006-10-14.
43. ^ "FATX Specification" (http://www.free60.org/FATX). free60 wiki. Retrieved 2011-08-16.
44. ^ "Xbox partitioning and file system details" (http://www.xbox-linux.org/wiki/Xbox_Partitioning_and_Filesystem_Details). Xbox-Linux project.
45. ^ "Differences between Xbox FATX and MS-DOS FAT" (http://www.xbox-linux.org/wiki/Differences_between_Xbox_FATX_and_MS-DOS_FAT). Xbox-Linux project.
46. ^ Microsoft. "exFAT File System Intellectual Property licensing program" (http://www.microsoft.com/en-us/legal/intellectualproperty/IPLicensing/Programs/exFATFileSystem.aspx). Retrieved 2013-04-23.
47. ^ **a b c d e f g h i j k l m n o p q r s t u v w** *Microsoft MS-DOS 3.1 Programmierhandbuch in englischer Sprache* [*Microsoft MS-DOS Operating System Programmer's Reference Manual*] (in English). München: Markt & Technik Verlag (published 1986). 1984. ISBN 3-89090-368-1. 8411-310-02, 036-014-012. "In regard to the jump instruction at the start of a boot sector: "Determine if the first byte of the boot sector is an E9H or EBIT (the first byte of a 3-byte NEAR or 2-byte short jump) or an EBH (the first byte of a 2-byte jump followed by a NOP). If so, a BPB is located beginning at offset 3."" (NB. This book contains many errors.)
48. ^ **a b** Daniel B. Sedory. *The Boot Sector of IBM® Personal Computer™ DOS Version 1.00 (1981)*. 2005-08-02 ([11] (http://thestarman.narod.ru/DOS/ibm100/Boot.htm#AuthID)).
49. ^ **a b** Daniel B. Sedory. *The Boot Sector of IBM® Personal Computer™ DOS Version 1.10 (1982)*. 2005-07-29 ([12] (http://thestarman.narod.ru/DOS/ibm110/Boot.htm#AuthID)).
50. ^ Caldera (1997). *Caldera OpenDOS Machine Readable Source Kit 7.01*. The DISK.ASM file in the machine readable source kit shows that DR-DOS tests on value `0x69` as well.
51. ^ Matthias Paul (2002-02-20). "Need DOS 6.22 (Not OEM)" (http://groups.google.com/group/alt.msdos.programmer/msg/6b10a1ea602e61e). *alt.msdos.programmer*. Retrieved 2006-10-14.
52. ^ Wally Bass (1994-02-14). "Cluster Size" (http://groups.google.co.uk/group/comp.os.msdos.programmer/msg/79de2d76832cfbd6). *comp.os.msdos.programmer*. Retrieved 2006-10-14.
53. ^ **a b c d e f g h** Dave Williams (1992). *Programmer's Technical Reference for MSDOS and the IBM PC*. DOSREF, Shareware version 01/12/1992. ISBN 1-878830-02-3. ([13] (http://www.o3one.org/hwdocs/bios_doc/dosref22.html), accessed on 2012-01-08). Comment: The author mentions that DOS 4.0 checks the OEM label, but denies that DOS 3.2 checks it as well (although it does).
54. ^ Matthias Paul (2004-08-25). "NOVOLTRK.REG" (http://www.ibiblio.org/pub/micro/pc-stuff/freedos/win9x/NOVOLTRK.ZIP). *www.drdos.org*. Retrieved 2011-12-17.
55. ^ IBM (1983). *IBM PC Technical Reference Handbook*. Comment: Includes a complete listing of the ROM BIOS source code of the original IBM PC.
56. ^ **a b c d** Hans-Dieter Jankowski, Dietmar Rabich, Julian F. Reschke (1992). *Atari Profibuch ST-STE-TT*. Sybex, 4th edition, 12th batch. ISBN 3-88745-888-5, ISBN 978-3-88745-888-1.
57. ^ Seagate Technologies, "The Transition to Advanced Format 4K Sector Hard Drives", 2010 ([14] (http://www.seagate.com/docs/pdf/whitepaper/tp613_transition_to_4k_sectors.pdf)).
58. ^ **a b c d** Ralf Brown (2002-12-29). "The x86 Interrupt List" (http://www.cs.cmu.edu/~ralf/files.html). Retrieved 2011-10-14.
59. ^ **a b c** *Microsoft MS-DOS Programmer's Reference: version 5.0*. Microsoft press. 1991. ISBN 1-55615-329-5.
60. ^ **a b c d e f g h i j k** "Standard Floppy Disk Formats Supported by MS-DOS" (http://support.microsoft.com/kb/75131). 2.0. Microsoft Help and Support. 2003-05-12. Retrieved 2012-09-11.
61. ^ **a b c** Microsoft (1987-07). *MS-DOS 3.3 Programmer's Reference*.
62. ^ **a b c** Andries Brouwer (2002-09-20). "The FAT file system" (http://www.win.tue.nl/~aeb/linux/fs/fat/fat.html). Retrieved 2011-10-16.
63. ^ **a b** "Detailed Explanation of FAT Boot Sector" (http://support.microsoft.com/kb/140418). *Microsoft Knowledge Base*. 2003-12-06. Retrieved 2011-10-16.
64. ^ **a b c** Lai, Robert S.; The Waite Group (1987). *Writing MS-DOS Device Drivers* (2nd ed.). Addison Wesley. ISBN 0-201-60837-5.
65. ^ **a b c d** *PORT-DOS - Userprompt Guide for Apricot Portable*. User-Prompt Guides, UK ([15] (http://actapricot.org/support/apricot_user_prompt_guide_portable.pdf)).
66. ^ **a b c d e** John Elliott (1998). *DOSPLUS disc formats*. ([16] (http://www.seasip.demon.co.uk/Cpm/fat_formats.html)).
67. ^ **a b c d** *The BBC Master 512*. Yellow Pig's BBC Computer Pages ([17] (http://www.cowsarenotpurple.co.uk/bbccomputer/master512/format.html)).

68. ^ "Detailed Explanation of FAT Boot Sector" (http://www.dewassoc.com/kbase/hard_drives/boot_sector.htm). DEW Associates Corporation. 2002. Retrieved 2011-10-16.
69. ^ Daniel B. Sedory. *Detailed Notes on the "Dirty Shutdown Flag" under MS-Windows*. 2001-12-04. ([18] (http://thestarman.narod.ru/DOS/DirtyShutdownFlag.html)).
70. ^ Andries Brouwer (2002-09-20). "FAT" (http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html#ss1.3). Retrieved 2012-01-11.
71. ^ ***a b c d e f g h i j k l m n o p q r s t u v w x y z aa ab ac ad ae af ag ah ai aj ak*** Matthias Paul (1997-07-30). "NWDOS-TIPs — Tips & Tricks rund um Novell DOS 7, mit Blick auf undokumentierte Details, Bugs und Workarounds" (http://www.antonis.de/dos/dos-tuts/mpdostip/html/nwdostip.htm) (e-book). *MPDOSTIP* (in German) (edition 3, release 157 ed.). Retrieved 2012-01-11. NWDOSTIP.TXT is a comprehensive work on Novell DOS 7 and OpenDOS 7.01, including the description of many undocumented features and internals. It is part of the author's yet larger MPDOSTIP.ZIP collection maintained up to 2001 and distributed on many sites at the time. The provided link points to a HTML-converted older version of the NWDOSTIP.TXT file.
72. ^ ***a b c d e f g h i j k l m n o p q*** Caldera (1997). *Caldera OpenDOS Machine Readable Source Kit 7.01*. The FDOS.EQU file in the machine readable source kit has equates for the corresponding directory entries.
73. ^ John Elliott (1998). *CP/M 4.1 disc formats*. ([19] (http://www.seasip.demon.co.uk/Cpm/format41.html)): "CP/M 4.1 (DOS Plus [1.2]) allows the use of two filesystems - CP/M and DOS. The version [...] supplied with the Amstrad PC1512 cannot handle larger floppies than 360k (CP/M) / 1.2Mb (DOS), or larger hard drive partitions than 32Mb. [...] The DOS filesystem can be either FAT12 or FAT16. The format is exactly as in PCDOS 2.11, except: Byte 0Ch of the directory entry [...] holds the four "user attributes" F1'-F4' [...] DRDOS-style passwords are not supported."
74. ^ ***a b*** vinDaci (1998-01-06). "Long Filename Specification" (http://www.teleport.com/~brainy/lfn.htm). Retrieved 2007-03-13.
75. ^ Netlabs. *FAT32.IFS Wiki and Sources*. ([20] (http://svn.netlabs.org/fat32/wiki/WikiStart)).
76. ^ ***a b c d e f g h i j k l m n*** Caldera, Inc. (1997). *OpenDOS Developer's Reference Series — System and Programmer's Guide — Programmer's Guide*. Printed in the UK, August 1997. Caldera Part No. 200-DODG-003 ([21] (http://www.drdos.net/documentation/sysprog/httoc.htm)).
77. ^ IBM (2003). *Information about 4690 OS unique file distribution attributes*, IBM document R1001487, 2003-07-30. ([22] (http://www-01.ibm.com/support/docview.wss?uid=pos1R1001487)): "[...] file types are stored in the "Reserved bits" portion of the PC-DOS file directory structure [...] only 4690 respects and preserves these attributes. Various non-4690 operating systems take different actions if these bits are turned on [...] when copying from a diskette created on a 4690 system. [...] PC-DOS and Windows 2000 Professional will copy the file without error and zero the bits. OS/2 [...] 1.2 [...] will refuse to copy the file unless [...] first run CHKDSK /F on the file. After [...] CHKDSK, it will copy the file and zero the bits. [...] when [...] copy [...] back to the 4690 system, [...] file will copy as a local file."
78. ^ IBM. *4690 save and restore file distribution attributes*. IBM document R1000622, 2010-08-31 ([23] (http://www-01.ibm.com/support/docview.wss?uid=pos1R1000622)).
79. ^ "mount(8): mount file system" (http://linux.die.net/man/8/mount). *Linux man page*.
80. ^ Duncan, Ray (1989). "Design goals and implementation of the new High Performance File System" (http://cd.textfiles.com/megademo2/INFO/OS2_HPFS.TXT) **4** (5). Microsoft Systems Journal. [NB. This particular text file has a number of OCR errors; e.g., "Ray" is the author's correct name; not 'Roy' as the text shows.]
81. ^ Chen, Raymond (2006-07). "Microsoft TechNet: A Brief and Incomplete History of FAT32" (http://www.microsoft.com/technet/technetmag/issues/2006/07/WindowsConfidential/). Microsoft TechNet Magazine.
82. ^ Microsoft.com (http://www.microsoft.com/presspass/press/2003/dec03/12-03ExpandIPPR.mspx)
83. ^ title = FAT%20File%20System "FAT File System" (http://www.microsoft.com/iplicensing/productDetail.aspx?product). *Intellectual Property Licensing*. Microsoft.
84. ^ "At PUBPAT's request, patent office rejects Microsoft's FAT patent: Government Relies Heavily on Evidence Submitted by PUBPAT" (http://www.pubpat.org/Microsoft_517_Rejected.htm). Public Patent Foundation. 2004-09-30. Retrieved 2006-10-14.
85. ^ Ina Fried (2004-09-30). "Microsoft FAT patent falls flat" (http://archive.is/TPQ9b). CNET News. Archived from the original (http://news.com.com/Microsoft+FAT+patent+falls+flat/2100-1014_3-5390138.html) on 2013-01-02. Retrieved 2006-10-14.
86. ^ Andrew Orlowski (2005-10-05). "Microsoft FAT patent rejected — again" (http://www.regdeveloper.co.uk/2005/10/05/microsoft_patent/). The Register. Retrieved 2006-10-14.
87. ^ "Patent Office rejects two Microsoft FAT patents" (http://www.out-law.com/default.aspx?page=6202). out-law.com. 2005-06-10. Retrieved 2006-10-14.
88. ^ Matthew Garrett (Jan 19, 2012). "EFI and Linux: the future is here, and it's awful - Matthew Garrett" (https://www.youtube.com/watch?v=V2aq5M3Q76U). *linux.conf.au 2012*. Retrieved 2 April 2012.
89. ^ Andrew Orlowski (2004-06-14). "Microsoft's war on GPL dealt patent setback" (http://www.theregister.co.uk/2004/06/14/ms_fat_patent_reexamined/). The Register. Retrieved 2006-10-14.
90. ^ Anne Broache (2006-01-10). "Microsoft's file system patent upheld" (http://archive.is/TeF5C). CNET News. Archived from the original (http://news.com.com/Microsofts+file+system+patent+upheld/2100-1012_3-6025447.html) on 2013-01-19. Retrieved 2006-10-14.
91. ^ Paul, Ryan (2009-02-25). "Microsoft suit over FAT patents could open OSS Pandora's Box" (http://arstechnica.com/microsoft/news/2009/02/microsoft-sues-tomtom-over-fat-patents-in-linux-based-device.ars). arstechnica.com. Retrieved 2009-02-28.
92. ^ Fried, Ina (2009-03-30). "Microsoft, TomTom settle patent dispute" (http://news.cnet.com/8301-13860_3-10206988-56.html). cnet.com. Retrieved 2009-08-22.
93. ^ "Microsoft Motorola Patent Suit" (http://www.scribd.com/doc/38550703/Micrsoft-Motorola-Patent-Suit). 2010-10-01. Retrieved 2010-10-02.
94. ^ Protalinski, Emil (2010-10-01). "Microsoft sues Motorola, citing Android patent infringement" (http://arstechnica.com/microsoft/news/2010/10/microsoft-sues-motorola-citing-android-patent-infringement.ars). arstechnica.com. Retrieved 2010-10-02.
95. ^ Brown, Eric (2009-07-02). "Can FAT patch avoid Microsoft lawsuits?" (http://www.desktoplinux.com/news/NS4980952387.html?kc=rss). DesktopLinux.Com. Retrieved 2009-08-23.

External links

- ECMA-107 Volume and File Structure of Disk Cartridges for Information Interchange (http://www.ecma-international.org/publications/standards/Ecma-107.htm), identical to ISO/IEC 9293.
- Microsoft Extensible Firmware Initiative FAT32 File System Specification, FAT: General Overview of On-Disk Format (http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx)
- Understanding FAT32 file systems (explained for embedded firmware developers) (http://www.pjrc.com/tech/8051/ide/fat32.html)
- Understanding FAT (http://users.iafrica.com/c/cq/cquirke/fat.htm) including lots of info about LFNs
- Detailed Explanation of FAT Boot Sector (http://support.microsoft.com/kb/140418/): Microsoft Knowledge Base Article 140418
- Description of the FAT32 File System (http://support.microsoft.com/kb/154997/): Microsoft Knowledge Base Article 154997
- FAT12/FAT16/FAT32 file system implementation for *nix (http://sourceforge.net/projects/libfat/): Includes libfat libraries and fusefat, a FUSE file system driver
- MS-DOS: Directory and Subdirectory Limitations (http://support.microsoft.com/kb/39927/): Microsoft Knowledge Base Article 39927
- Overview of FAT, HPFS, and NTFS File Systems (http://support.microsoft.com/kb/100108/): Microsoft Knowledge Base Article 100108
- Volume and file size limits of FAT file systems (http://www.microsoft.com/technet/prodtechnol/winxppro/reskit/c13621675.mspx): Microsoft Technet
- Microsoft TechNet: A Brief and Incomplete History of FAT32 (http://www.microsoft.com/technet/technetmag/issues/2006/07/WindowsConfidential/) by Raymond Chen
- FAT32 Formatter (http://www.ridgecrop.demon.co.uk/index.htm?fat32format.htm): allows formatting volumes larger than 32 GiB with FAT32 under Windows 2000, Windows XP and Windows Vista
- Fdisk does not recognize full size of hard disks larger than 64 GB (http://support.microsoft.com/kb/263044): Microsoft Knowledge Base Article 263044.
- Microsoft Windows XP: FAT32 File System (http://web.archive.org/web/20050319235548/www.microsoft.com/resources/documentation/Windows/XP/all/reskit/en-us/prkc_fil_cycz.asp). Copy made by Internet Archive Wayback Machine (http://www.archive.org/) of an article with summary of limits in FAT32 which is no longer available on Microsoft website.
- Visual Layout of a FAT16 drive (http://www.beginningtoseethelight.org/fat16/)