File Allocation Table

From Wikipedia, the free encyclopedia

File Allocation Table or **FAT** is a computer file system architecture originally developed by Bill Gates and Marc McDonald in 1976/1977.[1][2] It is the primary file system for various operating systems including DR-DOS, OpenDOS, freeDOS, MS-DOS, OS/2 (v1.1), and Microsoft Windows (up to Windows Me). For floppy disks (FAT12 and FAT16 without long filename support) it has been standardized as ECMA-107[3] and ISO/IEC 9293.[4][5] The use of long filenames with FAT is patented in part.

The FAT file system is relatively straightforward and is supported by virtually all existing operating systems for personal computers. This makes it an ideal format for solid-state memory cards and a convenient way to share data between operating systems.

Common implementations have a serious drawback in that when files are deleted and new files written to the media, directory fragments tend to become scattered over the entire disk, making reading and writing slower on storage devices which have higher seek time for random data access (Such as mechanical hard drives). Manually-invoked periodic defragmentation is one solution to this problem, but is often a lengthy process, and unwise in some instances.

Defragmentation is not required for solid-state memory cards, because there is no penalty associated with seeking from one location on the disk to another. Furthermore, because flash devices have a significantly limited number of lifetime writes (compared

FAT			
Developer	Microsoft		
Full Name	File Allocation Table		
	(12-bit version)	(16-bit version)	(32-bit version)
Introduced	1980 (Seattle QDOS)	November 1987, (Compaq DOS 3.31)	August 1996 (Windows 95 OSR2)
Partition identifier	0x01 (MBR)	0x04, 0x06, 0x0E (MBR)	0x0B, 0x0C (MBR) EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 (GPT)
Structures			
Directory contents	Table		
File allocation	Linked List		
Bad blocks	Cluster tagging		
Limits			
Max file size	4 GB − 1 byte (or volume size if smaller)		
Max cluster count	4,077 (2^{12}-19)	65,517 (2^{16}-19)	268,435,437 (2^{28}-19)
Max filename size	8.3 filename, or 255 UTF-16 characters when using LFN		
Max volume size	32 MB	2 GB 4 GB with 64k clusters (not widely supported)	2 TB 8 TB (2-KB sector)
Features			
Dates recorded	Creation, modified, access (accuracy to day only) (Creation time and access date are only available when LFN support is enabled)		
Date range	January 1, 1980 - December 31, 2107		
Forks	Not natively		
Attributes	Read-only, hidden, system, volume label, subdirectory, archive		
Permissions	No		
Transparent compression	Per-volume, Stacker, DoubleSpace, DriveSpace		No

with magnetic storage media), these devices should not be defragmented.

Transparent encryption	Per-volume only with DR-DOS	No

Contents

- 1 History
  - 1.1 FAT12
  - 1.2 Initial FAT16
  - 1.3 Extended partition and logical drives
  - 1.4 Final FAT16
  - 1.5 Long file names
  - 1.6 FAT32
  - 1.7 Fragmentation
  - 1.8 Third party support
  - 1.9 FAT and Alternate Data Streams
  - 1.10 Future
  - 1.11 exFAT
- 2 Design
  - 2.1 Boot Sector
    - 2.1.1 Exceptions
  - 2.2 File Allocation Table
  - 2.3 Directory table
    - 2.3.1 Long file names
    - 2.3.2 Third-party extensions
- 3 FAT licensing
  - 3.1 Appeal
- 4 See also
- 5 Notes and references
- 6 External links

History

The FAT file system was created for managing disks in Microsoft Standalone Disk BASIC. In August 1980 Tim Paterson incorporated FAT into his 86-DOS operating system for the S-100 8086 CPU boards;[6] the file system was the main difference between 86-DOS and its predecessor, CP/M.

The name originates from the usage of a table which centralizes the information about which areas belong to files, are free or possibly unusable, and where each file is stored on the disk. To limit the size of the table, disk space is allocated to files in contiguous groups of hardware sectors called *clusters*. As disk drives have evolved,

the maximum number of clusters has dramatically increased, and so the number of bits to identify a cluster has grown. The successive major versions of the FAT format are named after the number of table element bits: 12, 16, and 32. The FAT standard has also been expanded in other ways while preserving backward compatibility with existing software.

FAT12

The initial version of FAT is now referred to as **FAT12**. Designed as a file system for floppy diskettes, it limited cluster addresses to 12-bit values, which not only limited the cluster count to 4078,[7] but made FAT manipulation tricky with the PC's 8-bit and 16-bit registers. (Under Linux, FAT12 is limited to 4084 clusters.[8]) The disk's size is stored as a 16-bit count of sectors, which limited the size to 32 MB[9]. FAT12 was used by several manufacturers with different physical formats, but a typical floppy diskette at the time was 5.25-inch, single-sided, 40 tracks, with 8 sectors per track, resulting in a capacity of 160 KB for both the system areas and files. The FAT12 limitations exceeded this capacity by a factor of ten or more.

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. At the time FAT12 was introduced, DOS did not support hierarchical directories, and the maximum number of files was typically limited to a few dozen. Hierarchical directories were introduced in MS-DOS version 2.0.[10]

A limitation which was not addressed until much later was that any bad sector in the control structures area, track 0, could prevent the diskette from being usable. The DOS formatting tool rejected such diskettes completely. Bad sectors were allowed only in the file area, where they made the entire holding cluster unusable as well. FAT12 remains in use on 1.44MB floppy disks.

Initial FAT16

In 1984, IBM released the PC AT, which featured a 20 MB hard disk. Microsoft introduced MS-DOS 3.0 in parallel. (The earlier PC XT was the first PC with a hard drive from IBM, and MS-DOS 2.0 supported that hard drive with FAT12.) Cluster addresses were increased to 16-bit, allowing for up to 65,517 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 MB did not change. Therefore, although technically already "FAT16", this format was not what today is commonly understood as FAT16. 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 a smaller clusters, making disk usage more efficient, particularly for small files (which were more common at the time than they are today, after the 1990s). Also, the introduction of FAT16 actually did bring an increase in the maximum partition size under MS-DOS, since the implementation of FAT12 for hard disks in MS-DOS 2.0 was limited to 15 MB. (That is, the initial FAT16 did not support larger drives than FAT12, but MS-DOS 3.0 using FAT16 did support larger drives than MS-DOS 2.0 using FAT12, by a factor of two.)[11]

A 20 MB hard disk formatted under MS-DOS 3.0 was not accessible by the older MS-DOS 2.0. (This was because MS-DOS 2.0 did not support version 3.0's FAT-16 *and* because it did not support hard disk partitions over 15 MB in size.) Of course, MS-DOS 3.0 could still access MS-DOS 2.0 style 8 KB-cluster partitions.

MS-DOS 3.0 also introduced support for high-density 1.2 MB 5.25" diskettes, which notably had 15 sectors per track, hence more space for the FATs. This probably prompted a dubious optimization of the cluster size, which went down from 2 sectors to just 1. The net effect was that high density diskettes were significantly slower than

older *double density* ones.

Extended partition and logical drives

Apart from improving the structure of the FAT file system itself, a parallel development allowing an increase in the maximum possible FAT size was the introduction of multiple FAT partitions. Originally partitions were supposed to be used only for sharing the disk between operating systems, typically DOS and Xenix at the time, so DOS was only prepared to handle one FAT partition. It was not possible to create multiple DOS partitions using DOS tools, and third party tools would warn that such a scheme would not be compatible with DOS. Simply allowing several identical-looking DOS partitions could lead to naming problems: should C: be the first FAT partition on disk, for simplicity, or rather the partition marked as *active* in the partition table, so that several DOS versions can co-exist? And which partition should be C: if the system was booted from a diskette?

To allow the use of more FAT partitions in a compatible way, a new partition type was introduced (in MS-DOS 3.2, January 1986), the *extended partition*, which is a container for additional partitions called *logical drives*. Originally only one logical drive was possible, permitting hard disks up to 64 MB. In MS-DOS 3.3 (August 1987) this limit was increased to 24 drives, equal to the maximum number of available letters for drive names (A and B being reserved for the first two floppy drives, at least one of which many, if not most, systems of the era were equipped; where only one was installed, B always simulated a second drive using A.) Logical drives are described by on-disk structures which closely resemble the Master Boot Record (MBR) of the disk (which describes the *primary* partitions), likely to simplify the implementation. Though some believe these partitions were *nested* in a way analogous to Russian matryoshka dolls, that isn't the case. They are stored as a row of separate blocks within a single box; these blocks are often referred to as being *chained* together, by the *links* in their extended boot record (EBR) sectors. Only one extended partition is allowed. Under MS-DOS, logical drives are not bootable, and the extended partition can only be created after the primary FAT partition, which removes all ambiguity but also eliminates the possibility of booting several DOS versions from the same hard disk. (A few systems other than MS-DOS can boot logical drives, and partitions can be created in any order using third party formatting tools.)

A useful side-effect of the extended partition scheme was to significantly increase the maximum number of partitions possible on a PC hard disk beyond the four which could be described by the MBR alone.

Prior to the introduction of extended partitions, some hard disk controllers (which at that time were separate option boards, since the IDE standard did not yet exist) could make large hard disks appear at the hardware (or BIOS?) interface level as two separate disks.

Final FAT16

Finally in November 1987, Compaq DOS 3.31 (an OEM version of MS-DOS 3.3 released by Compaq with their machines) introduced what is today called the *FAT16* format, with the expansion of the 16-bit disk sector count to 32 bits. The result was initially called the *DOS 3.31 Large File System*. 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.

In 1988 the improvement became more generally available through MS-DOS 4.0 and OS/2 1.1. The limit on partition size was dictated by the 8-bit signed count of sectors-per-cluster, which 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 KB clusters, thereby fixing the "definitive" limit for the FAT16 partition size at 2 gigabytes. On magneto-optical media, which can have 1 or 2 KB sectors, the limit is proportionally greater.

Much later, Windows NT increased the maximum cluster size to 64 KB 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 also supported reading and writing this variant, but its disk utilities did not work with it.

The number of root directory entries available is determined when the volume is formatted, and is stored in a 16-bit signed field, defining an absolute limit of 32767 entries (32736, a multiple of 32, in practice). For historical reasons, FAT12 and FAT16 media generally use 512 root directory entries on non-floppy media. Other sizes may be incompatible with some software or devices (entries being file and/or folder names in the original 8.3 format).[12] Some third party tools like mkdosfs allow the user to set this parameter.[13]

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 UTF-16 code points long), in addition to classic 8.3 filenames. LFNs were implemented using a work-around in the way directory entries are laid out (see below). The version of the file system with this extension is usually known as VFAT after the Windows 95 VxD device driver, also known as "Virtual FAT" in Microsoft's documentation.

Interestingly, the VFAT driver actually appeared before Windows 95, in Windows for Workgroups 3.11, but was only used for implementing 32-bit File Access, a higher performance protected mode file access method, bypassing DOS and directly using either the BIOS, or, better, the Windows-native protected mode disk drivers.

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

FAT32

In order to overcome the volume size limit of FAT16, while still allowing DOS real mode code to handle the format without unnecessarily reducing the available conventional memory, Microsoft implemented a newer generation of FAT, known as **FAT32**, with cluster values held in a 32-bit field, of which 28 bits are used to hold the cluster number, for a maximum of approximately 268 million (2^{28}) clusters. This allows for drive sizes of up to 8 terabytes with 32KB clusters, but the boot sector uses a 32-bit field for the sector count, limiting volume size to 2 TB on a hard disk with 512 byte sectors.

On Windows 95/98, due to the version of Microsoft's SCANDISK utility included with these operating systems being a 16-bit application, the FAT structure is not allowed to grow beyond around 4.2 million ($< 2^{22}$) clusters, placing the volume limit at 127.53 gigabytes.[14] A limitation in original versions of Windows 98/98SE's Fdisk utility causes it to incorrectly report disk sizes over 64GB.[15] A corrected version is available from Microsoft. These limitations do not apply to Windows 2000/XP except during Setup, in which there is a 32GB limit.[16] Windows Me supports the FAT32 file system without any limits.[17] However, similarly to Windows 95/98/98SE there is no native support for 48-bit LBA in Windows ME, meaning that the maximum disk size for ATA disks is 127.6GB, the maximum size of an ATA disk using the previous long-standard 28-bit LBA.

FAT32 was introduced with Windows 95 OSR2, 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 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.

Windows 2000 and Windows XP can read and write to FAT32 file systems of any size, but the format program included in Windows 2000 and higher can only create FAT32 file systems of 32 GB or less. This limitation is by design and according to Microsoft was imposed because many tasks on a very large FAT32 file system become slow and inefficient.[14][18] This limitation can be bypassed by using third-party formatting utilities or by using the built-in FORMAT.EXE command-line utility.[19][20]

The maximum possible size for a file on a FAT32 volume is 4 GB minus 1 "null" byte ($2^{32}-1$ bytes). Video applications, large databases, and some other software easily exceed this limit. Larger files require another formatting type such as HFS+ or NTFS. Until mid-2006, those who run dual boot systems or who move external data drives between computers with different operating systems had little choice but to stick with FAT32. Since then, full support for NTFS has become available in Linux and many other operating systems, by installing the FUSE library (on Linux) together with the NTFS-3G driver. Data exchange is also possible between Windows and Linux by using the Linux-native ext2 or ext3 file systems through the use of external drivers for Windows, such as ext2 IFS; however, Windows cannot boot from ext2 or ext3 partitions.

Fragmentation

The FAT file system does not contain mechanisms which prevent newly written files from becoming scattered across the partition.[6] Other file systems, like HPFS, use free space bitmaps that indicate used and available clusters, which could then be quickly looked up in order to find free contiguous areas (improved in exFAT). 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, computing free disk space on FAT is one of the most resource intensive operations, as it requires reading the entire FAT linearly. 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.[18] Displaying this line took longer and longer as the number of clusters increased.

The High Performance File System (HPFS) divides disk space into *bands*, which have their own free space bitmap, where multiple files opened for simultaneous write could be expanded separately.[6]

Some of the perceived problems with fragmentation resulted from operating system and hardware limitations.

The single-tasking DOS and the traditionally single-tasking PC hard disk architecture (only 1 outstanding input/output request at a time, no DMA transfers) did not contain mechanisms which could alleviate fragmentation by asynchronously prefetching next data while the application was processing the previous chunks.

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

MS-DOS also did not offer a system call which would allow applications to make sure a particular file has been completely written to disk in the presence of deferred writes (cf. fsync in Unix or DosBufReset in OS/2). Disk caches on MS-DOS were operating on disk block level and were not aware of higher-level structures of the file system. In this situation, cheating with regard to the real progress of a disk operation was most dangerous.

Modern operating systems have introduced these optimizations to FAT partitions, but optimizations can still produce unwanted artifacts in case of a system crash. A Windows NT system will allocate space to files on FAT in advance, selecting large contiguous areas, but in case of a crash, files which were being appended will appear larger than they were ever written into, with dozens of random kilobytes at the end.

With the large cluster sizes, 16 or 32K, forced by larger FAT32 partitions, the *external* fragmentation becomes somewhat less significant, and *internal* fragmentation, ie. disk space waste (since files are rarely exact multiples of cluster size), starts to be a problem as well, especially when there are a great many small files.

Third party support

Other IBM PC operating systems—such as Linux, FreeBSD, BeOS and JNode—have all supported FAT, and most added support for VFAT, FAT32, JFAT (http://www.jnode.org/node/844) shortly after the corresponding Windows versions were released. Early Linux distributions also supported a format known as UMSDOS, which was FAT 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 is not enabled by default in Linux kernels from version 2.5.7 onwards.[21] The Mac OS X operating system also supports the FAT file systems on volumes other than the boot disk. The Amiga supports FAT through the CrossDOS file system.

FAT 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). The second design, while presumably more efficient, prevents any copying or backing-up of those volumes using non-aware tools; manipulating such volumes using non-aware disk utilities (e.g. defragmenters or CHKDSK) will probably lose the information.

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".

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 2 previously reserved bytes in the file's (or directory's) directory entry. In the FAT32 format, these bytes hold the upper 16 bits of the starting cluster number of the file or directory, hence making it difficult to store EAs on FAT32. Extended attributes are accessible via the Workplace Shell desktop, through REXX scripts, and many system GUI and command-line utilities (such as 4OS2).[22]

To accommodate its OS/2 subsystem, Windows NT supports the handling of extended attributes in HPFS, NTFS, and FAT. It stores EAs on FAT 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.

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").

Future

Microsoft has recently secured patents for VFAT and FAT32 (but not the original FAT). Despite two earlier rulings against them, Microsoft prevailed and was awarded the patents.

Since Microsoft has announced the discontinuation of its MS-DOS-based consumer operating systems with Windows Me, it remains unlikely that any new versions of FAT will appear. For most purposes, the NTFS file system that was developed for the Windows NT line is superior to FAT from the points of view of efficiency, performance, 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 or Linux (with NTFS-3G) is finally eroding this drawback.

FAT is still the normal file system for removable media (with the exception of CDs and DVDs), with FAT12 used on floppies, and FAT16 on most other removable media (such as flash memory cards for digital cameras and USB flash drives). Most removable media are not yet large enough to benefit from FAT32, although some larger flash drives, like SDHC, do make use of it. FAT16 is used on these drives for reasons of compatibility and size overhead.

The FAT32 formatting support in Windows 2000 and XP is limited to volumes of 32 GB, which effectively forces users of modern hard drives either to use NTFS, to partition the drive into smaller volumes (below 32 GB), or to format the drive using third party tools.

exFAT

exFAT is an incompatible replacement for FAT file systems that was introduced with Windows Embedded CE 6.0. It is intended to be used on flash drives, where FAT is used today. Windows XP file system drivers will be offered by Microsoft shortly after the release of Windows CE 6.0, while Windows Vista Service Pack 1 added exFAT support to Windows Vista.[23] exFAT introduces a free space bitmap allowing faster space allocation and faster deletes, support for files up to 2^{64} bytes, larger cluster sizes (up to 32 MB in the first implementation), an extensible directory structure and name hashes for filenames for faster comparisons. It does not have short 8.3 filenames anymore. It does not appear to have security access control lists or file system journaling like NTFS, though device manufacturers can choose to implement simplified support for transactions (backup file allocation table used for the write operations, primary FAT for storing last known good allocation table).

Design

The following is an overview of the order of structures in a FAT partition or disk:

Boot sector	More reserved sectors (optional)	File Allocation Table #1	File Allocation Table #2	Root Directory (FAT12/16 only)	Data Region (for files and directories) ... (To end of partition or disk)

A FAT file system is composed of four different sections.

1. The **Reserved sectors**, located at the very beginning. The first reserved sector is the Boot Sector (aka *Partition Boot Record*). 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. The total count of reserved sectors is indicated by a field inside the Boot Sector. Important information from the Boot Sector is accessible through an operating system structure called the *Drive Parameter Block* in DOS and OS/2. For FAT32 file systems, the reserved sectors include a *Backup Boot Sector* at Sector 6.
2. The **FAT Region**. This typically contains two copies (may vary) of the *File Allocation Table* for the sake of redundancy checking, although the extra copy is rarely used, even by disk repair utilities. These are maps of the Data Region, indicating which clusters are used by files and directories.
3. 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 means that the root directory has 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 instead, allowing it to grow without such a restraint.
4. The **Data Region**. This is where the actual file and directory data is stored and takes up most of the partition. 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 entirely in a cluster, and so if a 1 KB file resides in a 32 KB cluster, 31 KB are wasted.

FAT uses little endian format for entries in the header and the FAT(s).

Boot Sector

Common structure of the first 36 bytes used by all FAT versions:

Byte Offset	Length (bytes)	Description
0x00	3	Jump instruction. This instruction will be executed and will skip past the rest of the (non-executable) header if the partition is booted from. See Volume Boot Record. If the jump is two-byte near jmp it is followed by a NOP instruction.
0x03	8	OEM Name (padded with spaces). MS-DOS checks this field to determine which other parts of the boot record can be relied on.[24][25] Common values are `IBM 3.3` (with two spaces between the "IBM" and the "3.3"), `MSDOS5.0` and `MSWIN4.1`.
0x0b	2	Bytes per sector. A common value is 512, especially for file systems on IDE (or compatible) disks. The *BIOS Parameter Block* starts here.
0x0d	1	Sectors per cluster. Allowed values are powers of two from 1 to 128. However, the value must not be such that the number of bytes per cluster becomes greater

<table>
<tr><td></td><td></td><td>than 32 KB.</td></tr>
<tr><td>0x0e</td><td>2</td><td>Reserved sector count. The number of sectors before the first FAT in the file system image. Should be 1 for FAT12/FAT16. Usually 32 for FAT32.</td></tr>
<tr><td>0x10</td><td>1</td><td>Number of file allocation tables. Almost always 2.</td></tr>
<tr><td>0x11</td><td>2</td><td>Maximum number of root directory entries. Only used on FAT12 and FAT16, where the root directory is handled specially. Should be 0 for FAT32. This value should always be such that the root directory ends on a sector boundary (i.e. such that its size becomes a multiple of the sector size). 224 is typical for floppy disks.</td></tr>
<tr><td>0x13</td><td>2</td><td>Total sectors (if zero, use 4 byte value at offset 0x20)</td></tr>
<tr><td>0x15</td><td>1</td><td>Media descriptor[26]
<table>
<tr><td>0xF0</td><td>3.5" Double Sided, 80 tracks per side, 18 or 36 sectors per track (1.44MB or 2.88MB). 5.25" Double Sided, 15 sectors per track (1.2MB). Used also for other media types.</td></tr>
<tr><td>0xF8</td><td>Hard disk. Single sided, 80 tracks per side, 9 sectors per track</td></tr>
<tr><td>0xF9</td><td>3.5" Double sided, 80 tracks per side, 9 sectors per track (720K). 5.25" Double sided, 40 tracks per side, 15 sectors per track (1.2MB)</td></tr>
<tr><td>0xFA</td><td>5.25" Single sided, 80 tracks per side, 8 sectors per track (320K)</td></tr>
<tr><td>0xFB</td><td>3.5" Double sided, 80 tracks per side, 8 sectors per track (640K)</td></tr>
<tr><td>0xFC</td><td>5.25" Single sided, 40 tracks per side, 9 sectors per track (180K)</td></tr>
<tr><td>0xFD</td><td>5.25" Double sided, 40 tracks per side, 9 sectors per track (360K). Also used for 8".</td></tr>
<tr><td>0xFE</td><td>5.25" Single sided, 40 tracks per side, 8 sectors per track (160K). Also used for 8".</td></tr>
<tr><td>0xFF</td><td>5.25" Double sided, 40 tracks per side, 8 sectors per track (320K)</td></tr>
</table>
Same value of media descriptor should be repeated as first byte of each copy of FAT. Certain operating systems (MSX-DOS version 1.0) ignore boot sector parameters altogether and use media descriptor value from the first byte of FAT to determine file system parameters.</td></tr>
<tr><td>0x16</td><td>2</td><td>Sectors per File Allocation Table for FAT12/FAT16</td></tr>
<tr><td>0x18</td><td>2</td><td>Sectors per track</td></tr>
<tr><td>0x1a</td><td>2</td><td>Number of heads</td></tr>
<tr><td>0x1c</td><td>4</td><td>Hidden sectors</td></tr>
<tr><td>0x20</td><td>4</td><td>Total sectors (if greater than 65535; otherwise, see offset 0x13)</td></tr>
</table>

Further structure used by FAT12 and FAT16, also known as *Extended BIOS Parameter Block*:

Byte Offset	Length (bytes)	Description
0x24	1	Physical drive number
0x25	1	Reserved ("current head")
0x26	1	Extended boot signature. Value is 0x29[26] or 0x28.
0x27	4	ID (serial number)
0x2b	11	Volume Label
0x36	8	FAT file system type, padded with blanks (0x20), e.g.: "FAT12 ", "FAT16 ". This is not meant to be used to determine drive type, however, some utilities use it in this way.
0x3e	448	Operating system boot code
0x1FE	2	Boot sector signature (0x55 0xAA)

The boot sector is portrayed here as found on e.g. an OS/2 1.3 boot diskette. Earlier versions used a shorter BIOS Parameter Block and their boot code would start earlier (for example at offset 0x2b in OS/2 1.1).

Further structure used by FAT32:

Byte Offset	Length (bytes)	Description
0x24	4	Sectors per file allocation table
0x28	2	FAT Flags
0x2a	2	Version
0x2c	4	Cluster number of root directory start
0x30	2	Sector number of FS Information Sector
0x32	2	Sector number of a copy of this boot sector
0x34	12	Reserved
0x40	1	Physical Drive Number
0x41	1	Reserved
0x42	1	Extended boot signature.
0x43	4	ID (serial number)
0x47	11	Volume Label
0x52	8	FAT file system type: "FAT32 "
0x5a	420	Operating system boot code
0x1FE	2	Boot sector signature (0x55 0xAA)

Exceptions

The implementation of FAT used in MS-DOS for the Apricot PC 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 file system parameters (offsets 0x0B - 0x17 in the standard sector) were located at offset 0x50. 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.

DOS Plus on the BBC Master 512 did not use conventional boot sectors at all. Data disks omitted the boot sector and began with a single copy of the FAT (the first byte of the FAT was used to determine disk capacity) while boot disks began with a miniature ADFS file system containing the boot loader, followed by a single FAT. It could also access standard PC disks formatted to 180 KB or 360 KB, again using the first byte of the FAT to determine capacity.

File Allocation Table

A partition 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 KB and 32 KB. 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 a list of entries that map to each cluster on the partition. Each entry records one of five things:

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

Each version of the FAT file system uses a different size for FAT entries. The size is indicated by the name, for example the FAT16 file system uses 16 bits for each entry while the FAT32 file system uses 32 bits. Only 28 of these are actually used, however. This difference means that the File Allocation Table of a FAT32 system can map a greater number of clusters than FAT16, allowing for larger partition sizes with FAT32. This also allows for more efficient use of space than FAT16, because on the same hard drive a FAT32 table can address smaller clusters which means less wasted space.

FAT entry values:

FAT12	FAT16	FAT32	Description
0x000	0x0000	0x?0000000	Free Cluster
0x001	0x0001	0x?0000001	Reserved value; do not use
0x002 - 0xFEF	0x0002 - 0xFFEF	0x?0000002 - 0x?FFFFFEF	Used cluster; value points to next cluster
0xFF0 - 0xFF6	0xFFF0 - 0xFFF6	0x?FFFFFF0 - 0x?FFFFFF6	Reserved values; do not use[26].
0xFF7	0xFFF7	0x?FFFFFF7	Bad sector in cluster or reserved cluster

0xFF8 - 0xFFF	0xFFF8 - 0xFFFF	0x?FFFFFF8 - 0x?FFFFFFF	Last cluster in file

Note that 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. In the table above these are denoted by a question mark.

The first cluster of the Data Region is cluster #2. That leaves the first two entries of the FAT unused. In the first byte of the first entry a copy of the media descriptor is stored. The remaining 8 bits (if FAT16), or 20 bits (if Fat32) of this entry are 1. In the second entry the end-of-cluster-chain marker is stored. The high order two bits of the second entry are sometimes, in the case of FAT16 and FAT32, used for dirty volume management: high order bit 1: last shutdown was clean; next highest bit 1: during the previous mount no disk I/O errors were detected.[27]

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 creation, 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.

Legal characters for DOS file names 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 be used on the DOS command line because of the lack of a suitable escaping system)
- `! # $ % & ' ( ) - @ ^ _ ` { } ~`
- (FAT-32 only) `+ , . ; = [ ]`
- Values 128–255

This excludes the following ASCII characters:

- `" * / : < > ? \ |`
  Windows/MSDOS has no shell escape character
- Lower case letters `a–z`
  stored as `A–Z` on FAT-12/16
- Control characters 0–31
- Value 127 (DEL)

The DOS file names are in the OEM character set.

Directory entries, both in the Root Directory Region and in subdirectories, are of the following format:

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

The first byte can have the following special values:

<table>
<tr><td>0x00</td><td>Entry is available and no subsequent entry is in use</td></tr>
<tr><td>0x05</td><td>Initial character is actually 0xE5</td></tr>
<tr><td>0x2E</td><td>'Dot' entry; either '.' or '..'</td></tr>
<tr><td>0xE5</td><td>Entry has been previously erased and is available. File undelete utilities must replace this character with a regular character as part of the undeletion process.</td></tr>
</table></td></tr>
<tr><td>0x08</td><td>3</td><td>DOS file extension (padded with spaces)</td></tr>
<tr><td>0x0b</td><td>1</td><td>File Attributes

The first byte can have the following special values:

<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th></tr>
<tr><td>0</td><td>0x01</td><td>Read Only</td></tr>
<tr><td>1</td><td>0x02</td><td>Hidden</td></tr>
<tr><td>2</td><td>0x04</td><td>System</td></tr>
<tr><td>3</td><td>0x08</td><td>Volume Label (only allowed in entry in root directory)</td></tr>
<tr><td>4</td><td>0x10</td><td>Subdirectory</td></tr>
<tr><td>5</td><td>0x20</td><td>Archive</td></tr>
<tr><td>6</td><td>0x40</td><td>Device (internal use only, never found on disk)</td></tr>
<tr><td>7</td><td>0x80</td><td>Unused</td></tr>
</table>
An attribute value of 0x0F is used to designate a long file name entry.</td></tr>
<tr><td>0x0c</td><td>1</td><td>Reserved; two bits are used by NT and later versions to encode case information (see below); otherwise 0[28]</td></tr>
<tr><td>0x0d</td><td>1</td><td>Create time, fine resolution: 10ms units, values from 0 to 199.</td></tr>
<tr><td>0x0e</td><td>2</td><td>Create time. The hour, minute and second are encoded according to the following bitmap:

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>15-11</td><td>Hours (0-23)</td></tr>
</table></td></tr>
</table>

		<table><tr><td>10-5</td><td>Minutes (0-59)</td></tr><tr><td>4-0</td><td>Seconds/2 (0-29)</td></tr></table> Note that the *seconds* is recorded only to a 2 second resolution. Finer resolution for file creation is found at offset 0x0d.
0x10	2	Create date. The year, month and day are encoded according to the following bitmap: <table><tr><th>Bits</th><th>Description</th></tr><tr><td>15-9</td><td>Year (0 = 1980, 127 = 2107)</td></tr><tr><td>8-5</td><td>Month (1 = January, 12 = December)</td></tr><tr><td>4-0</td><td>Day (1 - 31)</td></tr></table>
0x12	2	Last access date; see offset 0x10 for description.
0x14	2	EA-Index (used by OS/2 and NT) in FAT12 and FAT16, High 2 bytes of first cluster number in FAT32
0x16	2	Last modified time; see offset 0x0e for description.
0x18	2	Last modified date; see offset 0x10 for description.
0x1a	2	First cluster in FAT12 and FAT16. Low 2 bytes of first cluster in FAT32. Entries with the Volume Label flag, subdirectory ".." pointing to root, and empty files with size 0 should have first cluster 0.
0x1c	4	File size. Entries with the Volume Label or Subdirectory flag set should have a size of 0.

Clusters are numbered from a cluster offset as defined above. That is, a zero in 0x1a would mean the first data segment is at:

reservedSectors + (*noofFAT* * *sectors*2*FAT*) + (*maxRootEntry* * 32 / *bytes*2*sector*) − 2

Long file names

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.

Older versions of PC-DOS mistake LFN names in the root directory for the volume label, and are likely to display an incorrect label.

Each phony entry can contain up to 13 UTF-16 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 UTF-16 characters.[28]

After the last UTF-16 character, a 0x00 0x00 is added. Other not used characters are filled with 0xFF 0xFF.

LFN entries use the following format:

Byte Offset	Length	Description
0x00	1	Sequence Number
0x01	10	Name characters (five UTF-16 characters)
0x0b	1	Attributes (always 0x0F)
0x0c	1	Reserved (always 0x00)
0x0d	1	Checksum of DOS file name
0x0e	12	Name characters (six UTF-16 characters)
0x1a	2	First cluster (always 0x0000)
0x1c	4	Name characters (two UTF-16 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 here also has bit 7 (0x40) checked (this means the last LFN entry. However it's the first entry got when reading the directory file). The last LFN entry has the biggest sequence number which decreases in following entries. The *first* LFN entry has sequence number 1. Bit 8 (0x80) of the sequence number is used to indicate that the entry is deleted.

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 spaces (ASCII 0x20) char. 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 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 this. This creates a backwards-compatibility problem with older Windows versions (95, 98, 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, 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.[29]

Third-party extensions

Before Microsoft added support for long filenames and creation/access time stamps, bytes 0x0C–0x15 of the directory entry were used by alternative operating systems to store additional metadata. These included:

<table>
<tr><th>Byte Offset</th><th>Length</th><th>System</th><th>Description</th></tr>
<tr><td>0x0C</td><td>2</td><td>RISC OS</td><td>File type, 0x000 - 0xFFF</td></tr>
<tr><td>0x0C</td><td>1</td><td>DOS Plus</td><td>User-defined file attributes F1-F4
<table><tr><th>Bit</th><th>Mask</th><th>Description</th></tr><tr><td>7</td><td>0x80</td><td>F1</td></tr><tr><td>6</td><td>0x40</td><td>F2</td></tr><tr><td>5</td><td>0x20</td><td>F3</td></tr><tr><td>4</td><td>0x10</td><td>F4</td></tr></table></td></tr>
<tr><td>0x0D</td><td>1</td><td>DR-DOS</td><td>For a deleted file, the original first character of the filename.</td></tr>
<tr><td>0x0E</td><td>2</td><td>DR-DOS and FlexOS</td><td>Encrypted file password</td></tr>
<tr><td>0x0E</td><td>2</td><td>ANDOS</td><td>File address in the memory</td></tr>
<tr><td>0x10</td><td>4</td><td>DR-DOS 7</td><td>For a deleted file, its original file time and date; deleted files have their normal time and date fields set to the time of deletion</td></tr>
</table>

<table>
<tr><td>0x12</td><td>2</td><td>DR-DOS 6 and FlexOS</td><td>File owner ID</td></tr>
<tr><td>0x14</td><td>2</td><td>DR-DOS and FlexOS</td><td>File permissions bitmap (execute permissions are only used by FlexOS):
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th></tr>
<tr><td>0</td><td>0x0001</td><td>Owner delete requires password</td></tr>
<tr><td>1</td><td>0x0002</td><td>Owner execute requires password</td></tr>
<tr><td>2</td><td>0x0004</td><td>Owner write requires password</td></tr>
<tr><td>3</td><td>0x0008</td><td>Owner read requires password</td></tr>
<tr><td>4</td><td>0x0010</td><td>Group delete requires password</td></tr>
<tr><td>5</td><td>0x0020</td><td>Group execute requires password</td></tr>
<tr><td>6</td><td>0x0040</td><td>Group write requires password</td></tr>
<tr><td>7</td><td>0x0080</td><td>Group read requires password</td></tr>
<tr><td>8</td><td>0x0100</td><td>World delete requires password</td></tr>
<tr><td>9</td><td>0x0200</td><td>World execute requires password</td></tr>
<tr><td>10</td><td>0x0400</td><td>World write requires password</td></tr>
<tr><td>11</td><td>0x0800</td><td>World read requires password</td></tr>
</table></td></tr>
</table>

FAT 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 2003-12-03 Microsoft announced 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.[30]

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://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5745902) - 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"). It is unclear whether this patent would cover an implementation of FAT without explicit long filename capabilities. Hard links in Unix file systems do not appear to be prior art: deleting a FAT file via its long name will also remove its short name. Renaming a file to a "short" name also updates the long file name for coherency; similarly, renaming a file to a "long" name will allocate a new "short" name. In NTFS, hard links and dual names are separate concepts and each hard link has two names. Finally, at the API level, both names are always provided together when a

directory lookup is requested from the system; they do not appear as two separate files and do not have to be "matched" to determine unique files.

- U.S. Patent 5,579,517 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5579517) - 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[31] on 2004-09-14, due to prior disclosure[32] of the claimed techniques in patents U.S. Patent 5,307,494 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5307494) and U.S. Patent 5,367,671 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5367671) . This decision was later overturned by the Patent Office on 2006-01-10.
- U.S. Patent 5,758,352 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5758352) - 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".[33][34] This decision was also later overturned by the Patent Office on 2006-01-10.
- U.S. Patent 6,286,013 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=6286013) - 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.

Many technical commentators have concluded that these patents only cover FAT implementations that include support for long filenames, and that removable solid state media and consumer devices only using short names would be unaffected.

Additionally, in the document "Microsoft Extensible Firmware Initiative FAT 32 File System Specification, FAT: General Overview of On-Disk Format" published by Microsoft (version 1.03, 2000-12-06), Microsoft specifically grants a number of rights, which many readers have interpreted as permitting operating system vendors to implement FAT.

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://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5367671) - 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.

Appeal

As there was widespread call for these patents to be re-examined, 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.[35]

On 2004-09-30 the USPTO rejected all claims of U.S. Patent 5,579,517 (http://patft.uspto.gov/netacgi /nph-Parser?patentnumber=5579517) , based primarily on evidence provided by PUBPAT. Dan Ravicher, the foundation's executive director, said, "The Patent Office has simply confirmed what we already knew for some time now, Microsoft's FAT patent is bogus."

According to the PUBPAT press release, "Microsoft still has the opportunity to respond to the Patent Office's rejection. Typically, third party requests for re-examination, like the one filed by PUBPAT, are successful in having the subject patent either narrowed or completely revoked roughly 70% of the time."

On 2005-10-05 the Patent Office 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://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5758352) invalid on the grounds that the patent had incorrect assignees.

Finally, on 2006-01-10 the Patent Office ruled that features of Microsoft's implementation of the FAT system were "novel and non-obvious", reversing both earlier non-final decisions.[36]

See also

- Comparison of file systems
- Drive letter assignment
- Software patent
- List of file systems
- Rock Ridge and Joliet — systems for CDs that add long file names similar to what VFAT did for FAT.

Notes and references

1. ^ Microsoft Systems Journal Sept 1989 v4 n5 p1(13) (http://cd.textfiles.com/megademo2/INFO/OS2_HPFS.TXT)
2. ^ www.microsoft.com/mscorp/ip/tech/fathist.asp (http://web.archive.org/web/*/www.microsoft.com/mscorp/ip/tech/fathist.asp) (archive.org)
3. ^ standards - Ecma-107 (http://www.ecma-international.org/publications/standards/Ecma-107.htm)
4. ^ standards - ISO 9293:1987 (http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16948)
5. ^ standards - ISO/IEC 9293:1994 (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21273)
6. ^ ***a*** ***b*** ***c*** Duncan, Ray (1989). "Design goals and implementation of the new High Performance File System (http://cd.textfiles.com/megademo2/INFO/OS2_HPFS.TXT) ". Microsoft Systems Journal. [Note: This particular text file has a number of 'scan' errors; e.g., "Ray" is the author's correct name; not 'Roy' as text shows.]
7. ^ Brian Jenkinson, Sammes, A. J. (2000). *Forensic Computing: A Practitioner's Guide (Practitioner Series)*. Berlin: Springer, 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."
8. ^ Andries Brouwer. "FAT Under Linux (http://www.win.tue.nl/%7Eaeb/linux/fs/fat/fat-2.html) ". Linux source code related to DOS often contains: `#define MSDOS_FAT12 4084` (see line 76 of "KernelAPI: msdos_fs.h (http://www.kernel-api.org/docs/online/2.2.26/dc/dd8/msdos__fs_8h-source.html) ".).
9. ^ File allocation is specified using binary meanings for K (1024^1 instead of 1000^1), M (1024^2 instead of 1000^2), G (1024^3 instead of 1000^3), ...
10. ^ "MS-DOS History (http://www.nukesoft.co.uk/msdos/dosversions.shtml) ".
11. ^ http://support.microsoft.com/kb/q69912/ Microsoft Knowledge Base article: "MS-DOS Partitioning Summary"
12. ^ "Errors Creating Files or Folders in the Root Directory (http://support.microsoft.com/kb/120138) ". Microsoft Help and Support (December 16, 2004). Retrieved on 2006-10-14.
13. ^ "mkdosfs man page (http://www.die.net/doc/linux/man/man8/mkdosfs.8.html) ".
14. ^ ***a*** ***b*** "Limitations of FAT32 File System (http://support.microsoft.com/kb/184006/en-us) ". Microsoft Help and Support (2004-12-16). Retrieved on 2006-10-14.
15. ^ "Fdisk Does Not Recognize Full Size of Hard Disks Larger than 64 GB (http://support.microsoft.com/kb/263044) ". Microsoft Help and Support (2007-01-27). Retrieved on 2007-03-08.

16. ^ "Limitations of the FAT32 File System in Windows XP (http://support.microsoft.com/kb/314463/en-us) ". Microsoft Help and Support (2002-09-04). Retrieved on 2007-01-24.
17. ^ "Windows XP/2000 FAT32 Formatting Limit (http://www.allensmith.net/Storage/HDDlimit/FAT32.htm) ". allensmith.net. Retrieved on 2007-04-08.
18. ^ **a** **b** Chen, Raymond (2006). Microsoft TechNet: A Brief and Incomplete History of FAT32 (http://www.microsoft.com/technet/technetmag/issues/2006/07/WindowsConfidential/) . *TechNet Magazine* July 2006.
19. ^ Fat32Format (http://www.ridgecrop.demon.co.uk/index.htm?fat32format.htm) - Windows program for formatting disks as FAT32 beyond the 32GB limit.
20. ^ Format Large Disks to FAT32 on Windows XP and Vista (http://www.gearhack.com/Forums/DisplayComments.php?file=Computer/Windows/Format_Large_Disks_to_FAT32_on_Windows_XP_and_Vista)
21. ^ "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 on 2006-10-14.
22. ^ 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 on 2006-10-14.
23. ^ Brandon LeBlanc (2007-08-28). "Vista SP1 Whitepaper (http://windowsvistablog.com/blogs/windowsvista/pages/windows-vista-service-pack-1-beta-whitepaper.aspx#_Toc175944550) ". Microsoft. Retrieved on 2007-08-28.
24. ^ 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 on 2006-10-14.
25. ^ Wally Bass (1994-02-14). "Cluster Size (http://groups.google.co.uk/group/comp.os.msdos.programmer/msg/79de2d76832cfbd6) ". *comp.os.msdos.programmer*. Retrieved on 2006-10-14.
26. ^ **a** **b** **c** (1991) *Microsoft MS-DOS Programmer's Reference : version 5.0*. Microsoft press. ISBN 1-55615-329-5.
27. ^ Andries E. Brouwer (2002-09-20). "The FAT filesystem (http://www.win.tue.nl/%7Eaeb/linux/fs/fat/fat-1.html) ". Retrieved on 2006-10-14.
28. ^ **a** **b** vinDaci (1998-01-06). "Long Filename Specification (http://www.teleport.com/%7Ebrainy/lfn.htm) ". Retrieved on 2007-03-13.
29. ^ "mount(8): mount file system – Linux man page (http://linux.die.net/man/8/mount) ".
30. ^ "Intellectual Property Licensing – FAT File System (http://www.microsoft.com/about/legal/intellectualproperty/search/details.mspx?ip_id=IDAEYWHD&keywords=fat) ". Microsoft.
31. ^ "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 on 2006-10-14.
32. ^ Ina Fried (2004-09-30). "Microsoft FAT patent falls flat (http://news.com.com/Microsoft+FAT+patent+falls+flat/2100-1014_3-5390138.html) ". CNET News. Retrieved on 2006-10-14.
33. ^ Andrew Orlowski (2005-10-05). "Microsoft FAT patent rejected - again (http://www.regdeveloper.co.uk/2005/10/05/microsoft_patent/) ". The Register. Retrieved on 2006-10-14.
34. ^ "Patent Office rejects two Microsoft FAT patents (http://www.out-law.com/default.aspx?page=6202) ". out-law.com (2005-06-10). Retrieved on 2006-10-14.
35. ^ 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 on 2006-10-14.
36. ^ Anne Broache (2006-01-10). "Microsoft's file system patent upheld (http://news.com.com/Microsofts+file+system+patent+upheld/2100-1012_3-6025447.html) ". CNET News. Retrieved on 2006-10-14.

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 FAT 32 File System Specification, FAT: General Overview of On-Disk Format (http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx)
- Understanding FAT32 Filesystems (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 Filesystem implementation for *nix (http://sourceforge.net/projects/libfat/) - Includes libfat libraries and fusefat , a FUSE filesystem 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
- FAT 32 Formatter (http://www.ridgecrop.demon.co.uk/index.htm?fat32format.htm) : allows formatting volumes larger than 32GB 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 not longer available on Microsoft website.
- EFSL (http://sf.net/projects/efsl) , Open source FAT implementation for embedded devices
- Visual Layout of a FAT16 drive (http://www.beginningtoseethelight.org/fat16/)
- Understand the Master Boot Record of FAT16 (http://www.viralpatel.net/taj/tutorial/fat.php)

Retrieved from "http://en.wikipedia.org/wiki/File_Allocation_Table"
Categories: Disk file systems | DOS on IBM PC compatibles | Windows disk file systems | Windows components | Ecma standards
Hidden categories: All pages needing cleanup | Articles with disputed statements from November 2008 | All articles with unsourced statements | Articles with unsourced statements since September 2007 | Articles with unsourced statements since August 2007 | Articles with unsourced statements since July 2007
