WIKIPEDIA engler, cs140e: annotated version for the fat32. start with paul's short writeup. this is a good companion.

Design of the FAT file system

A **FAT file system** is a specific type of 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.[3] It offers good performance even in very 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 nearly all currently developed operating systems for personal computers and many home computers, mobile devices and embedded systems, and thus is a well suited format for data exchange between computers and devices of almost any type and age from 1981 through the present.

Originally designed in 1977 for use on floppy disks, FAT was soon adapted and used almost universally on hard disks throughout the DOS and Windows 9x eras for two decades. Today, FAT file systems are still commonly found on floppy disks, USB sticks, flash and other solid-state memory cards and modules, and many portable and embedded devices. DCF implements FAT as the standard file system for digital cameras since 1998.[4] FAT is also utilized 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] According to Google Patents the "Common name space for long and short filenames"(US5758352A) status was expired in 2019, which may mean that patents expired completely.[9]

Contents

Technical overview

The name of the file system originates from the file system's prominent usage of an index table, the *File Allocation Table*, 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.

Originally designed as an 8-bit file system, the maximum number of clusters has been significantly increased as disk drives have evolved, 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). Except for the original 8-bit FAT precursor, 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.

Layout

FAT	
Developer(s)	Microsoft, SCP, IBM, Compaq, Digital Research, Novell, Caldera
Full name	File Allocation Table: FAT12 (12-bit version), FAT16 (16-bit versions), FAT32 (32-bit version with 28 bits used), exFAT (64-bit versions)
Introduced	1977 (Standalone Disk BASIC-80) FAT12: August 1980 (SCP QDOS) FAT16: August 1984 (IBM PC DOS 3.0) FAT16B: November 1987 (Compaq MS-DOS 3.31) FAT32: August 1996 (Windows 95 OSR2) exFAT: November 2006 (Windows Embedded CE 6.0)
Partition identifier	MBR/EBR: FAT12: `0x01` e.a. FAT16: `0x04` `0x06` `0x0E` e.a. FAT32: `0x0B` `0x0C` e.a. exFAT: `0x07` e.a. BDP: `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7`
Structures	
Directory contents	Table
File allocation	Linked list
Bad blocks	Cluster tagging
Limits	
Max. volume size	FAT12: 32 MB (256 MB for 64 KB clusters) FAT16: 2 GB (4 GB for 64 KB clusters) FAT32: 2 TB (16 TB for 4 KB sectors)
Max. file size	4,294,967,295 bytes (4 GB - 1) with FAT16B and FAT32[1]
Max. number of files	FAT12: 4,068 for 8 KB clusters FAT16: 65,460 for 32 KB clusters FAT32: 268,173,300 for 32 KB clusters
Max. filename length	8.3 filename, or 255 UCS-2 characters when using LFN
Features	
Dates recorded	Modified date/time, creation date/time (DOS 7.0 and higher

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

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

A FAT file system is composed of four regions:

Reserved sectors

The first reserved sector (logical sector 0) is the Boot Sector (also called *Volume Boot Record* or simply *VBR*). It includes an area called the *BIOS Parameter Block* (*BPB*) which contains 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 file systems.[10]
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 span 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.

FAT Region

This typically contains two copies 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.

Root Directory Region

n/a for fat32

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.

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`;[11] by way of Digital Research this value was also used on Atari ST formatted floppies.[nb 1] 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 formatting tools allow to configure the format filler byte.[nb 2])

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. Files are allocated in units of clusters, so if a 1 KB file resides in a 32 KB cluster, 31 KB 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 sector of each FAT copy 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), the root directory (n/a for FAT32), 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 partition, after the volume.

	only), access date (only available with ACCDATE enabled),[2] 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
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

Reserved sectors area

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)	Contents
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`[12][13] as seen since DOS 3.0[nb 3]—and on DOS 1.1[14][15]) or a near jump (`0xE9 0x?? 0x??`[12][13] as seen on most (Compaq, TeleVideo) DOS 2.x formatted disks as well as on some (Epson, Olivetti) DOS 3.1 disks). For backward compatibility MS-DOS, PC DOS and DR-DOS also accept a jump (`0x69 0x?? 0x??`)[12][13][16] on removable disks. On hard disks, DR DOS additionally accepts the swapped JMPS sequence starting with a NOP (`0x90 0xEB 0x??`),[16] 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.[17][18][19] 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 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. Windows 9x reads that self-marked disks without any problems but giving some strange values for non-meaning parameters which not exist or are not used when the disk was formatted with older BPB specification, e.g. disk serial number (which exists only for disks formatted on DOS 5.0 or later, and in Windows 9x after overwriting OEM label with `?????IHC` will report it as `0000-0000` or any other value stored in disk serial number field when using disk formatted on other system).[20] This applies only to removable disk drives. 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`). The boot ROM of the Wang Professional Computer will only treat a disk as bootable if the first four characters of the OEM label are "`Wang`". Similarly, the ROM BIOS of the Philips :YES will only boot from a disk if the first four characters of the OEM label are "`:YES`". If, in an FAT32 EBPB, the signature at sector offset `0x042` is `0x29` and both total sector entries are 0, the file system entry may serve as a 64-bit total sector count entry and the OEM label entry may be used as alternative file system type instead of the normal entry at offset `0x052`. In a similar fashion, if this entry is set to "`EXFAT␣␣␣`", it indicates the usage of an exFAT BPB located at sector offset `0x040` to `0x077`, whereas NTFS volumes use "`NTFS␣␣␣␣`"[21] to indicate an NTFS BPB.
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* (60 or 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; therefore the jump at offset `0x001` cannot be used to reliably derive the exact [E]BPB format from. (In conjunction with at least a DOS 3.31 BPB some GPT boot loaders (like *BootDuet*) use `0x1FA`–`0x1FD` to store the high 4 bytes of the hidden sectors for volumes located outside the first 2^{32}-1 sectors. Since this location may contain code or other data in other boot sectors, it may not be written to when `0x1F9`–`0x1FD` do not all contain zero.)
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`).[10][nb 4] 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[22]). 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[14][15] 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`.[23][nb 5] 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 formatting tool based on faulty documentation,[nb 4] 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)	Contents
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. The offset and length of this entry are different 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 4]) 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.[23] This checksum entry can be used to align the checksum accordingly.[nb 5] If the logical sector size is larger than 512 bytes, the remainder is not included in the checksum and is typically zero-filled.[23] Since some PC operating systems erroneously do not accept FAT formatted floppies if the `0x55 0xAA`[nb 4] 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 5] 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)	Contents
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)	Contents
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.[23] DR-DOS supports booting off FAT12/FAT16 volumes with logical sector sizes up to 32 KB and INT 13h implementations supporting physical sectors up to 1024 bytes/sector.[nb 6] 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 KB, 256 bytes for volumes larger 32 KB 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.[24] 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. Linux, and by extension Android, supports a logical sector size far larger, officially documented in the Man page for the filesystem utilities as up to 32KB.
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 KB only, whereas modern MS-DOS/PC DOS and Windows 95 support a maximum cluster size of 32 KB. Windows 98/SE/ME partially support a cluster size of 64 KB 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 KB clusters. For most DOS-based operating systems, the maximum cluster size remains at 32 KB (or 64 KB) even for sector sizes larger than 512 bytes. For logical sector sizes of 1 KB, 2 KB and 4 KB, Windows NT 4.0 supports cluster sizes of 128 KB, while for 2 KB and 4 KB sectors the cluster size can reach 256 KB. Some versions of DR-DOS provide limited support for 128 KB clusters with 512 bytes/sector using a sectors/cluster value of 0. MS-DOS/PC DOS will hang on startup if this value is erroneously specified as 0.[25]
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.
0x010	0x05	1	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. 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. A value of 0 without a FAT32 EBPB (no signature `0x29` or `0x28` at offset `0x042`) may also indicate a variable-sized root directory in some non-standard FAT12 and FAT16 implementations, which store the root directory start cluster in the cluster 1 entry in the FAT.[26] This extension, however, is not supported by mainstream operating systems,[26] as it can be conflictive with other uses of the cluster 1 entry for maintenance flags, the current end-of-chain-marker, or TFAT extensions. 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,[12] the maximum value supported by MS-DOS/PC DOS on hard disks is 512.[12] 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. 0 for FAT32. (If zero, use 4 byte value at offset `0x020`)
0x015	0x0A	1	Media descriptor (compare: FAT ID):[27][28][29][nb 3] **0xE5** ▪ 8-inch (200 mm) single sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (250.25 KB) (DR-DOS only) **0xED** ▪ 5.25-inch (130 mm) double sided, 80 tracks per side, 9 sector, 720 KB (Tandy 2000 only)[19] **0xEE** ▪ Designated for non-standard custom partitions (utilizing non-standard BPB formats or requiring special media access such as 48-/64-bit addressing); corresponds with `0xF8`, but not recognized by unaware systems by design; value not required to be identical to FAT ID, never used as cluster end-of-chain marker (Reserved for DR-DOS) **0xEF** ▪ Designated for non-standard custom superfloppy formats; corresponds with `0xF0`, but not recognized by unaware systems by design; value not required to be identical to FAT ID, never used as cluster end-of-chain marker (Reserved for DR-DOS) **0xF0**[5][6][7] ▪ 3.5-inch (90 mm) double sided, 80 tracks per side, 18 or 36 sectors per track (1440 KB, known as “1.44 MB”; or 2880 KB, 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.[30] **0xF4** ▪ Double density (Altos MS-DOS 2.11 only)[31] **0xF5**

print this out

			▪ Fixed disk, 4-sided, 12 sectors per track (1.95? MB) (Altos MS-DOS 2.11 only)[31] **0xF8** ▪ Fixed disk (i.e., typically a partition on a hard disk). (since DOS 2.0)[32][33] ▪ 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 KB) (MS-DOS 3.1[13] and MSX-DOS) ▪ 5.25-inch double sided, 80 tracks per side, 9 sectors per track (720 KB) (Sanyo 55x DS-DOS 2.11 only)[19] ▪ Single sided (Altos MS-DOS 2.11 only)[31] **0xF9**[5][6][7] ▪ 3.5-inch double sided, 80 tracks per side, 9 sectors per track (720 KB) (since DOS 3.2)[32] ▪ 3.5-inch double sided, 80 tracks per side, 18 sectors per track (1440 KB) (DOS 3.2 only)[32] ▪ 5.25-inch double sided, 80 tracks per side, 15 sectors per track (1200 KB, known as "1.2 MB") (since DOS 3.0)[32] ▪ Single sided (Altos MS-DOS 2.11 only)[31] **0xFA** ▪ 3.5-inch and 5.25-inch single sided, 80 tracks per side, 8 sectors per track (320 KB) ▪ Used also for RAM disks and ROM disks (e.g., on Columbia Data Products[34] 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 KB) **0xFC** ▪ 5.25-inch single sided, 40 tracks per side, 9 sectors per track (180 KB) (since DOS 2.0)[32] **0xFD** ▪ 5.25-inch double sided, 40 tracks per side, 9 sectors per track (360 KB) (since DOS 2.0)[32] ▪ 8-inch double sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (500.5 KB) ▪ (8-inch double sided, (single and) double density (DOS 1)[32]) **0xFE** ▪ 5.25-inch single sided, 40 tracks per side, 8 sectors per track (160 KB) (since DOS 1.0)[32][35] ▪ 8-inch single sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (250.25 KB)[31][35] ▪ 8-inch double sided, 77 tracks per side, 8 sectors per track, 1024 bytes per sector (1232 KB)[35] ▪ (8-inch single sided, (single and) double density (DOS 1)[32]) **0xFF** ▪ 5.25-inch double sided, 40 tracks per side, 8 sectors per track (320 KB) (since DOS 1.1)[32][35] ▪ Hard disk (Sanyo 55x DS-DOS 2.11 only)[19] 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.[12] On removable drives, DR-DOS will assume the presence of a BPB if this value is greater or equal to `0xF0`,[12] 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 indicate double-sided media.[13] 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. FAT32 sets this to 0 and uses the 32-bit value at offset `0x024` instead.

DOS 3.0 BPB:

The following extensions were documented since DOS 3.0, however, they were already supported by some issues of DOS 2.11.[31] 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)	Contents
0x00B	0x00	13	*DOS 2.0 BPB*
0x018	0x0D	2	Physical sectors per track for disks with INT 13h CHS geometry,[10] e.g., 15 for a "1.20 MB" (1200 KB) 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,[10] 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)	Contents
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)	Contents
0x00B	0x00	13	*DOS 2.0 BPB*
0x018	0x0D	2	Physical sectors per track for disks with INT 13h CHS geometry,[10] e.g., 18 for a “1.44 MB” (1440 KB) 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,[10] 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. (Some GPT boot loaders (like *BootDuet*) use boot sector offsets 0x1FA–0x1FD to store the high 4 bytes of a 64-bit hidden sectors value for volumes located outside the first $2^{32}-1$ sectors.)
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. For partitioned media, if this and the entry at 0x013 are both 0 (as seen on some DOS 3.x FAT16 volumes), many operating systems (including MS-DOS/PC DOS) will retrieve the value from the corresponding partition's entry (at offset 0xC) in the MBR instead. If both of these entries are 0 on volumes using a FAT32 EBPB with signature 0x29, values exceeding the 4,294,967,295 ($2^{32}-1$) limit (f.e. some DR-DOS volumes with 32-bit cluster entries) can use a 64-bit entry at offset 0x052 instead.

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 number of FATsFN 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 LSNs 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)	Contents
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).[33] Should be set to 0 by formatting 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][27] 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 7] and INT 21h/AH=2Ch (get system time)[nb 7] 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. Not available if the signature at 0x026 is set to 0x28. 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. Not available if the signature at 0x026 is set to 0x28.

FAT32 Extended BIOS Parameter Block

In essence FAT32 inserts 28 bytes into the EBPB, followed by the remaining 26 (or sometimes only 7) EBPB bytes as shown above for FAT12 and FAT16. Microsoft and IBM operating systems determine the type of FAT file system 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 possible 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,[nb 8] operating systems should be prepared to cope with these hybrid forms.

Sector offset	FAT32 EBPB offset	Length (bytes)	Contents
0x00B	0x00	25	*DOS 3.31 BPB*
0x024	0x19	4	Logical sectors per file allocation table (corresponds with the old entry at offset 0x0B in the DOS 2.0 BPB). 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.[10] 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). Formatting 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.[10] FAT32 implementations should refuse to mount volumes with version numbers unknown by them.
0x02C	0x21	4	Cluster number of root directory start, typically 2 (first cluster[36]) if it contains no bad sector. (Microsoft's FAT32 implementation imposes an artificial limit of 65,535 entries per directory, 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[25] (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.[10] 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[10] (and/or 0xFFFF[25]) are reserved and indicate that no backup sector is available.
0x034	0x29	12	Reserved (may be changed to format filler byte 0xF6[nb 2] as an artifact 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 9] 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) exFAT BPBs are located at sector offset 0x040 to 0x077, overlapping all the remaining entries of a standard FAT32 EBPB including this one. They can be detected via their OEM label signature "EXFAT␣␣␣" at sector offset 0x003. In this case, the bytes at 0x00B to 0x03F are normally set to 0x00.
0x041	0x36	1	Cf. 0x025 for FAT12/FAT16 (Used for various purposes; see FAT12/FAT16) May hold format filler byte 0xF6[nb 2] artifacts 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[21] 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) Not available if the signature at offset 0x042 is set to 0x28.
0x052	0x47	8	Cf. 0x036 for FAT12/FAT16 (File system type, padded with blanks (0x20), e.g., "FAT32␣␣␣"). Not available if the signature at 0x042 is set to 0x28. If both total logical sectors entries at offset 0x020 and 0x013 are 0 on volumes using a FAT32 EBPB with signature 0x29, volumes with more than 4,294,967,295 (2^{32}-1) sectors (f.e. some DR-DOS volumes with 32-bit cluster entries) can use this entry as *64-bit total logical sectors* entry instead. In this case, the OEM label at sector offset 0x003 may be retrieved as new-style *file system type* instead.

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 3]

Originally, the FAT ID was meant to be a bit flag with all bits set except for bit 2 cleared to indicate an 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,[13] 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)[28][29]	**0xFF**		**0xFE**			**0xFD**			**0xFC**	0xFB	0xFA	**0xF9**			0xF8		
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"
Density	?	DD 48tpi	SD	DD	DD 48tpi	SD	SD	DD 48tpi	DD 48tpi	?	?	HD 96tpi	DD 135tpi	HD 135tpi	QD 96tpi	?	DD
Modulation	?	MFM	FM	MFM	MFM	FM	FM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MFM	MF
Formatted capacity (KB)	?	320	250 ("old")[31][35]	1200	160	250 ("new")[31][35]	500	360	180	640	320	1200	720	1440	720	360	360
Cylinders (CHS)	77	40	77	77	40	77	77	40	40	80	80	80	80	80	80	80	80
Physical sectors / track (BPB offset 0x0D)	?	8	26	8	8	26	26	9	9	8	8	15	9	18	9 (8[34])	9	9
Number of heads (BPB offset 0x0F)	?	2	1[31][35]	2[13][28][35] (1)	1	1[13][31][35]	2[28]	2	1	2	1	2	2	2	2	1	1
Byte payload / physical sector	?	512	128	1024	512	128	128	512	512	512	512	512	512	512	512	512	512
Bytes / logical sector (BPB offset 0x00)	?	512	128	1024	512	128	128	512	512	512	512	512	512	512	512	512	512
Logical sectors / cluster (BPB offset 0x02)	?	2	4	1	1	4	4	2	1	2	1[28] (2?[13])	1	2	1	?	2	?
Reserved logical sectors (BPB offset 0x03)	?	1	1[31][35]	1	1	4[31][35]	4	1	1	1	1	1	1 (2)	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
Root directory entries (BPB offset 0x06)	?	112 (7 sectors)	68 (17 sectors)	192 (6 sectors)	64 (4 sectors)	68 (17 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)	?
Total logical sectors (BPB offset 0x08)	?	640	2002[31][35]	1232[28][35] (616[13])	320	2002[13][31][35]	4004[28]	720	360	1280	640	2400	1440	2880	?	720	?
Logical sectors / FAT (BPB offset 0x0B)	?	1	6[31][35]	2	1	6[31][35]	6?[28]	2	2	2	2[28] (1?[13])	7	3	9 (7)	?	2	?
Hidden sectors (BPB offset 0x11)	?	0	3[28] (0[13])	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total number of clusters	?	315	497	1227	313	?	997?[28]	354	351	?	?	2371	713	2847?	?	?	?
Logical sector order	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
Sector mapping	?	sector+ head+ track+	sector+ head+ track+	sector+ head+ track+	sector+ head+ track+	sector+ track+	sector+ head+ track+	sector+ head+ track+	sector+ head+ track+	sector+ head+ track+	sector+ track+	sector+ head+ track+	sector+ head+ track+	sector+ head+ track+	?	sector+ track+	sector+ track+
First physical sector (CHS)	?	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	?	?	?
BPB Presence	?	?	?	?	?	?	?	?	?	?	?	Yes	Yes	Yes	?	?	?
		5]	1][35]		5]									nly;	.11 only	,1[13]	

Support	?	DOS 1.1[3]	DOS 1.0[3]	DOS 2.0	DOS 1.0[3]	?[31][35]	DOS 2.0	DOS 2.0	DOS 2.0	?	?	DOS 3.0	DOS 3.2	DOS 3.2.0 (DR-DOS)	Sanyo 55x DS-DOS 2	MS-DOS 3	MSX-DOS

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.[29]

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 KB FAT12 format used in MS-DOS for the Apricot PC and F1e[37] 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 KB FAT12 format instead.[37] 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 KB: 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.[37][38]
- 720 KB: 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.[37]

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 KB data disks omitted a boot sector and began with a single copy of the FAT.[38] The first byte of the relocated FAT in logical sector 0 was used to determine the disk's capacity. 640 KB boot disks began with a miniature ADFS file system containing the boot loader, followed by a single FAT.[38][39] Also, the 640 KB 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).[39] 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 KB: 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.[38][39]
- 640 KB: 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.[38][39]

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

The DEC Rainbow 100 (all variations) supported one FAT12 format on 80-track, single-sided, quad-density 5.25" drives. The first two tracks were reserved for the boot loader, but didn't contain an MBR nor a BPB (MS-DOS used a static in-memory BPB instead). The boot sector (track 0, side 0, sector 1) was Z80 code beginning with DI `0xF3`. The 8088 bootstrap was loaded by the Z80. Track 1, side 0, sector 2 starts with the Media/FAT ID byte `0xFA`. Unformatted disks use `0xE5` instead. The file system starts on track 2, side 0, sector 1. There are 2 copies of the FAT and 96 entries in the root directory. In addition, there is a physical to logical track mapping to effect a 2:1 sector interleaving. The disks were formatted with the physical sectors in order numbered 1 to 10 on each track after the reserved tracks, but the logical sectors from 1 to 10 were stored in physical sectors 1, 6, 2, 7, 3, 8, 4, 9, 5, 10.[40]

FS Information Sector

The "FS Information Sector" was introduced in FAT32[41] 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)	Contents
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 file systems (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`)[10][nb 4] (All four bytes should match before the contents of this sector should be assumed to be in valid format.)

check these!

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`[25] (or `0x0000`) in the entry at offset `0x030`.

File Allocation Table

Cluster map

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 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.

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 (f.e. cluster 0) and the 12 most significant bits the second (f.e cluster 1). In other words, while the low eight bits of the first cluster in the row are stored in the first byte, the top four bits are stored in the low nibble of the second byte, whereas the low four bits of the subsequent cluster in the row are stored in the high nibble of the second byte and its higher eight bits in the third byte.

Example of FAT12 table start with several cluster chains

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	F0	FF	FF	03	40	00	05	60	00	07	80	00	FF	AF	00	14
+0010	C0	00	0D	E0	00	0F	00	01	11	F0	FF	00	F0	FF	15	60
+0020	01	19	70	FF	F7	AF	01	FF	0F	00	00	70	FF	00	00	00

- FAT ID / endianness marker (in reserved cluster #0), with 0xF0 indicating a volume on a non-partitioned superfloppy drive (must be 0xF8 for partitioned disks)
- End of chain indicator / maintenance flags (in reserved cluster #1)
- Second chain (7 clusters) for a non-fragmented file (here: #2, #3, #4, #5, #6, #7, #8)
- Third chain (7 clusters) for a fragmented, possibly grown file (here: #9, #A, #14, #15, #16, #19, #1A)
- Fourth chain (7 clusters) for a non-fragmented, possibly truncated file (here: #B, #C, #D, #E, #F, #10, #11)
- Empty clusters (here: #12, #1B, #1C, #1E, #1F)
- Fifth chain (1 cluster) for a sub-directory (here: #13)
- Bad clusters (3 clusters) (here: #17, #18, #1D)

The FAT16 file system uses 16 bits per FAT entry, thus one entry spans two bytes in little-endian byte order:

Example of FAT16 table start with several cluster chains

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	F0	FF	FF	FF	03	00	04	00	05	00	06	00	07	00	08	00
+0010	FF	FF	0A	00	14	00	0C	00	0D	00	0E	00	0F	00	10	00
+0020	11	00	FF	FF	00	00	FF	FF	15	00	16	00	19	00	F7	FF
+0030	F7	FF	1A	00	FF	FF	00	00	00	00	F7	FF	00	00	00	00

The FAT32 file system uses 32 bits per FAT entry, thus one entry spans four bytes in little-endian byte order. The four top bits of each entry are reserved for other purposes, cleared during format and should not be changed otherwise. They must be masked off before interpreting the entry as 28-bit cluster address.

Example of FAT32 table start with several cluster chains

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	F0	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	04	00	00	00
+0010	05	00	00	00	06	00	00	00	07	00	00	00	08	00	00	00
+0020	FF	FF	FF	0F	0A	00	00	00	14	00	00	00	0C	00	00	00
+0030	0D	00	00	00	0E	00	00	00	0F	00	00	00	10	00	00	00
+0040	11	00	00	00	FF	FF	FF	0F	00	00	00	00	FF	FF	FF	0F
+0050	15	00	00	00	16	00	00	00	19	00	00	00	F7	FF	FF	0F
+0060	F7	FF	FF	0F	1A	00	00	00	FF	FF	FF	0F	00	00	00	00
+0070	00	00	00	00	F7	FF	FF	0F	00	00	00	00	00	00	00	00

- First chain (1 cluster) for the root directory, pointed to by an entry in the FAT32 BPB (here: #2)
- Second chain (6 clusters) for a non-fragmented file (here: #3, #4, #5, #6, #7, #8)

The *File Allocation Table* (*FAT*) is a 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

For very early versions of DOS to recognize the file system, the system must have been booted from the volume or the volume's 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.

Special entries

The first two entries in a FAT store special values:

The first entry (cluster 0 in the FAT) holds the FAT ID since MS-DOS 1.20 and PC DOS 1.1 (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. These values were arranged so that the entry would also function as an "trap-all" end-of-chain marker for all data clusters holding a value of zero. Additionally, 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. 86-DOS 0.42 up to MS-DOS 1.14 used hard-wired drive profiles instead of a FAT ID, but used this byte to distinguish between media formatted with 32-byte or 16-byte directory entries, as they were used prior to 86-DOS 0.42.

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).[42] (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.) For DOS 1 and 2, the entry was documented as reserved for future use.

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)[43] 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.[30] 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 10][43] (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)[43] is cleared, the operating system has encountered disk I/O errors on startup,[43] 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.[30][43] (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.

Some non-standard FAT12/FAT16 implementations utilize the cluster 1 entry to store the starting cluster of a variable-sized root directory (typically 2[36]). This may occur when the number of root directory entries in the BPB holds a value of 0 and no FAT32 EBPB is found (no signature 0x29 or 0x28 at offset 0x042).[26] This extension, however, is not supported by mainstream operating systems,[26] as it is conflictive with other possible uses of the cluster 1 entry. Most conflicts can be ruled out if this extension is only allowed for FAT12 with less than 0xFEF and FAT16 volumes with less than 0x3FEF clusters and 2 FATs.

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,[36], marking the beginning of the data area.

Cluster values

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.[11][12] 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.[13]
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).[11][12] 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 11] - 0xFF5 (0xFF1 - 0xFF5)	0xFFF0 - 0xFFF5	0x?FFFFFF0 - 0x?FFFFFF5	Reserved in some contexts,[44] or also used[5][6][7][10][45] 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,[12] and should avoid allocating them for files otherwise. MS-DOS/PC DOS 3.3 and higher treats a value of 0xFF0[nb 11][12] on FAT12 (but not on FAT16 or FAT32) volumes as additional end-of-chain marker similar to 0xFF8-0xFFF.[12] 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][10][27][45] (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.[13]
0xFF7	0xFFF7	0x?FFFFFF7	Bad sector in cluster or reserved cluster (since DOS 2.0). 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,[12] respectively) will always be smaller than this value.[12] 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 the bad cluster marker with DOS 2.0 or the introduction of FAT16 with DOS 3.0,[13] 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.[13] 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 11] 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.[13] Most file system implementations (including 86-DOS, MS-DOS, PC DOS and DR-DOS) use 0xFFF[13] / 0xFFFF[13] / 0x0FFFFFFF as end-of-file marker when allocating files, but versions of Linux before 2.5.40 used 0xFF8 / 0xFFF8 / 0x0FFFFFF8.[46] Versions of mkdosfs (dosfstools up to 3.0.26) continue to use 0x0FFFFFF8 for the root directory on FAT32 volumes, 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 Standalone 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.[13] 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;[13] see note above for MS-DOS/PC DOS special usage of 0xFF0[nb 11] on FAT12 volumes.[12])

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.

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 KB)
> FAT16 minimum : 1 sector per cluster × 4,085 clusters = 2,091,520 bytes (2,042.5 KB)
> FAT32 minimum : 1 sector per cluster × 65,525 clusters = 33,548,800 bytes (32,762.5 KB)
>
> FAT12 maximum : 64 sectors per cluster × 4,084 clusters = 133,824,512 bytes (≈ 127 MB)
> [FAT12 maximum : 128 sectors per cluster × 4,084 clusters = 267,694,024 bytes (≈ 255 MB)]
>
> FAT16 maximum : 64 sectors per cluster × 65,524 clusters = 2,147,090,432 bytes (≈2,047 MB)
> [FAT16 maximum : 128 sectors per cluster × 65,524 clusters = 4,294,180,864 bytes (≈4,095 MB)]

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

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.[36] Likewise 65524+3 is `0xFFF7` for FAT16, and 4084+3 is `0xFF7` 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.[47]

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 straightforward. 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[10] 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,[36] 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,[45] or as Microsoft's FAT specification puts it:[10]

> *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.[48] 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 KB 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.[49] 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 KB, 32 KB, 64 KB) 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 inherent design without having to change the layout of the on-disk structures.[50][51] 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.[19] 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.[19] These pointers are initialized to point to the start of the FATs after bootup,[19] 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.[19] 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.[11] 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.[12]

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.[50][51] 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.

Directory table

A *directory table* is a special type of file that represents a directory (also known as a folder). Since 86-DOS 0.42,[52] each file or (since MS-DOS 1.40 and PC DOS 2.0) subdirectory 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 address of the first cluster of the file/directory's data, the size of the file/directory, and the date[52] and (since PC DOS 1.1) also the time of last modification. Earlier versions of 86-DOS used 16-byte directory entries only, supporting no files larger than 16 MB and no time of last modification.[52]

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 internal Current Directory Structure (CDS) under MS-DOS/PC DOS limits the absolute path of a directory to 66 characters (including the drive letter, but excluding the NUL byte 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.[53] 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.[54] 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.

Each entry can be preceded by "fake entries" to support a VFAT 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.
- `! # $ % & ' ( ) - @ ^ _ ` { } ~`
- Characters 128–228
- Characters 230–255

This excludes the following ASCII characters:

- `" * / : < > ? \ |`
  Windows/MS-DOS has no shell escape character

- `+ , . ; = [ ]`
  Allowed in long file names only
- Lower case letters a–z
  Stored as A–Z; allowed in long file names
- Control characters 0–31
- Character 127 (DEL)

Character 229 (`0xE5`) was not allowed as first character in a filename in DOS 1 and 2 due to its use as free entry marker. A special case was added to circumvent this limitation with DOS 3.0 and higher.

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[53] (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.[53])

The at-sign 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.[53]

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.[53]

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).

32-byte 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>Contents</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 1.25 and PC DOS 2.0, not in earlier versions of MS-DOS, PC DOS, or 86-DOS. Instead, they will treat such entries as allocated. Therefore, this value must not be used as end marker, if a volume should remain accessible under PC DOS 1.0/1.1 as well.[nb 12][55][56][57])</td></tr><tr><td>0x05</td><td>Initial character is actually 0xE5. (since DOS 3.0)
Under DR DOS 6.0 and higher, including PalmDOS, Novell DOS and OpenDOS, 0x05 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 0xE5.</td></tr><tr><td>0x2E</td><td>'Dot' entry; either "." or ".." (since MS-DOS 1.40 and PC DOS 2.0)</td></tr><tr><td>0xE5</td><td>Entry has been previously erased and/or is available.[nb 12][55][56][57] File undelete utilities must replace this character with a regular character as part of the undeletion process. See also: 0x05.
(The reason, why 0xE5 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.[11][nb 1])</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>0x0B</td><td>1</td><td>File Attributes
<table><tr><th>Bit</th><th>Mask</th><th>Description</th></tr><tr><td>0</td><td>0x01</td><td>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).
The DCF digital camera file system standard utilizes the Read Only attribute to allow directories or individual files (DCF objects) to be marked as "protected" from deletion by the user.[4]</td></tr><tr><td>1</td><td>0x02</td><td>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.[53] 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.</td></tr><tr><td>2</td><td>0x04</td><td>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.).</td></tr><tr><td>3</td><td>0x08</td><td>Volume Label. (Since MS-DOS 1.28 and PC 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 an 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.[53]</td></tr><tr><td>4</td><td>0x10</td><td>Subdirectory. (Since MS-DOS 1.40 and PC 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.</td></tr><tr><td>5</td><td>0x20</td><td>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.</td></tr><tr><td>6</td><td>0x40</td><td>Device (internally set for character device names found in filespecs, never found on disk), must not be changed by disk tools.</td></tr><tr><td>7</td><td>0x80</td><td>Reserved, must not be changed by disk tools.</td></tr></table>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. This check does not work on FAT32 volumes.</td></tr>
<tr><td>0x0C</td><td>1</td><td>▪ CP/M-86 and DOS Plus store user attributes F1'—F4' here.[58] (DOS Plus 1.2 with BDOS 4.1 supports passwords only on CP/M media, not on FAT12 or FAT16 media.[59] 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 MB. Therefore, this usage is not conflictive with FAT32.IFS, FAT16+ or FAT32+ as they can never occur on the same type of volume.):</td></tr>
</table>

Bit	Mask	Description
7	0x80	F1': Modify default open rules[53]
6	0x40	F2': Partial close default[53]
5	0x20	F3': Ignore Close Checksum Error[53]
4	0x10	F4': Disable checksums[53]
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 VFAT); otherwise 0.[60]
- DR-DOS 7.0x reserved bits other than 3 and 4 for internal purposes since 1997. The value should be set to 0 by formatting tools and must not be changed by disk tools.[53]
- 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),[61] 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.[62][63]

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.[53] 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[58] 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.[53] 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.[53]
- 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.[58] 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.[64]
- 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.[53][58] 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.[53] Offset 0x12 holds the user ID, 0x13 the group ID of a file's creator.[58]

  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.[64]

- 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.[53][58] 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);[2][53] 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.[53] 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.

<table>
<tr><td>0x14</td><td>2</td><td>

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,[58] and under FlexOS,[58] 4680 OS, 4690 OS, Concurrent DOS, Multiuser DOS, System Manager, and REAL/32.

Typical values stored on a single-user system are <code>0x0000</code> (<code>PASSWORD /N</code> for all access rights "RWED"), <code>0x0111</code> (<code>PASSWORD /D</code> for access rights "RW?-"), <code>0x0555</code> (<code>PASSWORD /W</code> for access rights "R-?-") and <code>0x0DDD</code> (<code>PASSWORD /R</code> for files or <code>PASSWORD /P</code> for directories for access rights "--?-").[53] 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 (<code>PASSWORD /V</code>) 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.[53] 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.[53]
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 <code>/U:name</code> parameter.[53]

File access rights bitmap:[65]
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th></tr>
<tr><td>0</td><td>0x0001</td><td>Owner delete/rename/attribute change requires permission[53][58][65]</td></tr>
<tr><td>1</td><td>0x0002</td><td>Owner execute requires permission (FlexOS, 4680 OS, 4690 OS only)[65]</td></tr>
<tr><td>2</td><td>0x0004</td><td>Owner write/modify requires permission[53][58][65]</td></tr>
<tr><td>3</td><td>0x0008</td><td>Owner read/copy requires permission[53][58][65]</td></tr>
<tr><td>4</td><td>0x0010</td><td>Group delete/rename/attribute change requires permission[53][58][65]</td></tr>
<tr><td>5</td><td>0x0020</td><td>Group execute requires permission (FlexOS, 4680 OS, 4690 OS only)[65]</td></tr>
<tr><td>6</td><td>0x0040</td><td>Group write/modify requires permission[53][58][65]</td></tr>
<tr><td>7</td><td>0x0080</td><td>Group read/copy requires permission[53][58][65]</td></tr>
<tr><td>8</td><td>0x0100</td><td>World delete/rename/attribute change requires permission[53][58][65]</td></tr>
<tr><td>9</td><td>0x0200</td><td>World execute requires permission (FlexOS, 4680 OS, 4690 OS only)[65]</td></tr>
<tr><td>10</td><td>0x0400</td><td>World write/modify requires permission[53][58][65]</td></tr>
<tr><td>11</td><td>0x0800</td><td>World read/copy requires permission[53][58][65]</td></tr>
<tr><td>12-15</td><td></td><td>bits must be set to 0 during format and must not be modified by disk tools later on;[53] bit 15 is used internally,[65] but not on disk</td></tr>
</table>
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.[53][66] A different method to store extended attributes has been devised for FAT32 volumes, see FAT32.IFS under offset <code>0x0C</code>.
High two bytes of first cluster number in FAT32; with the low two bytes stored at offset <code>0x1A</code>.

The storage of the high two bytes of the first cluster in a file on FAT32 is partially conflictive with access rights bitmaps.
</td></tr>
<tr><td>0x16</td><td>2</td><td>

Last modified time (since PC DOS 1.1/MS-DOS 1.20); see offset <code>0x0E</code> 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 <code>0x10</code> for possible later restoration.[53] See offset <code>0x0E</code> for a format description.

</td></tr>
<tr><td>0x18</td><td>2</td><td>

Last modified date; see offset <code>0x10</code> 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 <code>0x12</code> for possible later restoration.[53] See offset <code>0x10</code> for a format description.

</td></tr>
<tr><td>0x1A</td><td>2</td><td>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 <code>0x14</code>.
Entries with the Volume Label flag, subdirectory ".." pointing to the FAT12 and 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.</td></tr>
<tr><td>0x1C</td><td>4</td><td>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 <code>0x00000000</code> here. This can be used as part of a detection mechanism to distinguish between pending delete files under DELWATCH and VFAT LFNs; see above.</td></tr>
</table>

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:[67]

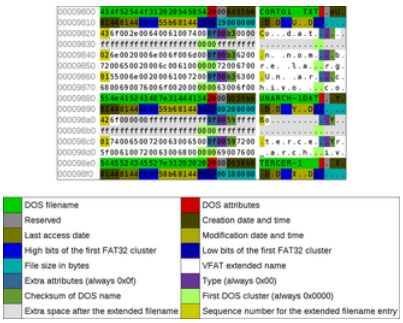
1. Local: Don't distribute file but keep on local controller only.[nb 14]
2. Mirror file on update: Distribute file to server only when file is updated.
3. Mirror file on close: Distribute file to server only when file is closed.
4. Compound file on update: Distribute file to all controllers when file is updated.
5. Compound file on close: Distribute file to all controllers when file is closed.[68]

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 (LFNs) are stored on a FAT file system using a trick: adding 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. It is also similar to a method publicly discussed to store long filenames on Ataris and under Linux in 1992.[69][70]



FAT32 directory structure with three files, two of which use VFAT long file names.

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 entries (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.[60]

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 further up)
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, the entry representing the *end* of the filename comes first. The sequence number of this entry has bit 6 (0x40) set to represent that it is the last logical LFN entry, and it has the highest sequence number. The sequence number decreases in the following entries. The entry representing the *start* of the filename 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, a filename like "File with very long filename.ext" 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. (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.[71]

See also

- Comparison of file systems
- Drive letter assignment
- exFAT
- Extended Boot Record (EBR)
- FAT filesystem and Linux
- List of file systems
- Master Boot Record (MBR)
- Partition type
- Timeline of DOS operating systems
- Transaction-Safe FAT File System
- Turbo FAT
- Volume Boot Record (VBR)

Notes

1. This is the reason, why `0xE5` had a special meaning in directory entries.
2. 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.
3. 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).
4. 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.
5. 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.
6. DR-DOS is able to boot off FAT12/FAT16 logical sectored media with logical sector sizes up to 1024 bytes.
7. 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).
8. Windows XP has been observed to create such hybrid disks when reformatting FAT16B formatted ZIP-100 disks to FAT32 format. The resulting volumes were FAT32 by format, but still used the FAT16B EBPB. (It is unclear how Windows determines the location of the root directory on FAT32 volumes, if only a FAT16 EBPB was used.)
9. 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`".
10. 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.
11. 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.
12. Some versions of FORMAT since MS-DOS 1.25 and PC DOS 2.0 supported an option `/0` (for *old*) to fill the first byte of all directory entries with `0xE5` instead of utilizing the end marker `0x00`. Thereby. the volume remained accessible under PC DOS 1.0-1.1, while formatting took somewhat longer and newer versions of DOS could not take advantage of the considerable speed-up caused by using the end marker `0x00`.
13. 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. "File Systems" (https://technet.microsoft.com/en-us/library/cc938937.aspx). Microsoft TechNet. 2001. Retrieved 2011-07-31.
2. 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+|-]..."
3. "FAT File System (Windows Embedded CE 6.0)" (http://msdn.microsoft.com/en-us/library/ee489982%28v=winembedded.60%29.aspx). Microsoft. 2010-01-06. Retrieved 2013-07-07.
4. JEIDA/JEITA/CIPA (2010). "Standard of the Camera & Imaging Products Association, CIPA DC-009-Translation-2010, Design rule for Camera File system: DCF Version 2.0 (Edition 2010)" (https://web.archive.org/web/20130930190707/http://www.cipa.jp/english/hyoujunka/kikaku/pdf/DC-009-2010_E.pdf) (PDF). Archived from the original (http://www.cipa.jp/english/hyoujunka/kikaku/pdf/DC-009-2010_E.pdf) (PDF) on 2013-09-30. Retrieved 2011-04-13.

5. "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. "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. "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" (https://www.google.com/patents?id=bUohAAAAEBAJ). *US Patent 5758352*. Retrieved 2012-01-19.
9. https://patents.google.com/patent/US5758352
10. "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.
11. Schulman, Andrew; Brown, Ralf D.; Maxey, David; Michels, Raymond J.; Kyle, Jim (1994) [November 1993]. ***Undocumented DOS: A programmer's guide to reserved MS-DOS functions and data structures - expanded to include MS-DOS 6, Novell DOS and Windows 3.1*** (https://archive.org/details/undocumenteddosp00andr_0/page/11) (2 ed.). Reading, Massachusetts: Addison Wesley. p. 11 (https://archive.org/details/undocumenteddosp00andr_0/page/11). ISBN 0-201-63287-X. ISBN 978-0-201-63287-3. (xviii+856+vi pages, 3.5"-floppy) Errata: [1] (https://web.archive.org/web/20190417215556/https://www.cs.cmu.edu/afs/cs/user/ralf/pub/books/UndocumentedDOS/errata.ud2)[2] (https://web.archive.org/web/20190417212906/https://www.pcjs.org/pubs/pc/programming/Undocumented_DOS/#errata-2nd-edition)
12. Chappell, Geoff (1994). *DOS Internals*. Addison Wesley. ISBN 978-0-201-60835-9. ISBN 0-201-60835-9. [3] (https://gopher.tildeverse.org/gopher.viste.fr/9/programming/PC/DOS/DOS%20Internals/DOS_Internals.zip)[4] (https://www.pcjs.org/pubs/pc/programming/DOS_Internals/) Errata: [5] (http://www.geoffchappell.com/notes/dos/internals/)[6] (https://gopher.tildeverse.org/gopher.viste.fr/1/programming/PC/DOS/DOS%20Internals/crtdrvr)[7] (https://gopher.tildeverse.org/gopher.viste.fr/1/programming/PC/DOS/DOS%20Internals/xmswatch)
13. *Microsoft MS-DOS 3.1 Programmierhandbuch in englischer Sprache* [*Microsoft MS-DOS 3.1 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.)
14. Daniel B. Sedory. *The Boot Sector of IBM Personal Computer DOS Version 1.00 (1981)*. 2005-08-02 ([8] (http://thestarman.narod.ru/DOS/ibm100/Boot.htm#AuthID)).
15. Daniel B. Sedory. *The Boot Sector of IBM Personal Computer DOS Version 1.10 (1982)*. 2005-07-29 ([9] (http://thestarman.narod.ru/DOS/ibm110/Boot.htm#AuthID)).
16. 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.
17. Paul, Matthias R. (2002-02-20). "Need DOS 6.22 (Not OEM)" (https://groups.google.com/d/msg/alt.msdos.programmer/MAv9LuYQjs8/HuYCph4KsQYJ). Newsgroup: alt.msdos.programmer (news:alt.msdos.programmer). Archived (https://archive.today/20170909145248/https://groups.google.com/forum/%23!msg/alt.msdos.programmer/MAv9LuYQjs8/HuYCph4KsQYJ) from the original on 2017-09-09. Retrieved 2006-10-14.
18. Bass, Wally (1994-02-14). "Cluster Size" (https://groups.google.com/d/msg/comp.os.msdos.programmer/ZGE7qkh4eI0/1vssg3Yt3nkJ). Newsgroup: comp.os.msdos.programmer (news:comp.os.msdos.programmer). Archived (https://archive.today/20170909144921/https://groups.google.com/forum/%23!msg/comp.os.msdos.programmer/ZGE7qkh4eI0/1vssg3Yt3nkJ) from the original on 2017-09-09. Retrieved 2006-10-14.
19. Dave Williams (1992). *Programmer's Technical Reference for MSDOS and the IBM PC*. DOSREF, Shareware version 01/12/1992. ISBN 1-878830-02-3. ([10] (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).
20. Paul, Matthias R. (2004-08-25). "NOVOLTRK.REG" (https://web.archive.org/web/20160304124755/http://www.ibiblio.org/pub/micro/pc-stuff/freedos/win9x/NOVOLTRK.ZIP). *www.drdos.org*. Archived from the original (http://www.ibiblio.org/pub/micro/pc-stuff/freedos/win9x/NOVOLTRK.ZIP) on 2016-03-04. Retrieved 2011-12-17. [11] (https://web.archive.org/web/20051111015142/http://public.www.planetmirror.com/pub/freedos/win9x/)
21. "Troubleshooting Disks and File Systems" (https://technet.microsoft.com/en-us/library/bb457122.aspx). Microsoft TechNet. 2005-11-05. Retrieved 2014-06-15.
22. IBM (1983). *IBM PC Technical Reference Handbook*. Comment: Includes a complete listing of the ROM BIOS source code of the original IBM PC.
23. 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.
24. Seagate Technologies, "The Transition to Advanced Format 4K Sector Hard Drives (archived by Wayback Machine @Archive.org)", 2010 ([12] (https://web.archive.org/web/20110902031330/http://seagate.com/docs/pdf/whitepaper/tp613_transition_to_4k_sectors.pdf)).
25. Brown, Ralf D. (2002-12-29). "The x86 Interrupt List" (https://www.cs.cmu.edu/~ralf/files.html). Retrieved 2011-10-14.
26. de Boyne Pollard, Jonathan (2010) [2006]. "All about BIOS parameter blocks" (http://jdebp.eu./FGA/bios-parameter-block.html). *Frequently Given Answers*. Retrieved 2014-06-02.
27. ***Microsoft MS-DOS Programmer's Reference: version 5.0*** (https://archive.org/details/isbn_9781556153297). Microsoft press. 1991. ISBN 1-55615-329-5.
28. "Standard Floppy Disk Formats Supported by MS-DOS" (http://support.microsoft.com/kb/75131). Microsoft Help and Support. 2003-05-12. Retrieved 2012-09-11.
29. Microsoft (1987-07). *MS-DOS 3.3 Programmer's Reference*.
30. Andries Brouwer (2002-09-20). "The FAT file system" (http://www.win.tue.nl/~aeb/linux/fs/fat/fat.html). Retrieved 2011-10-16.
31. Paterson, Tim; Microsoft (2013-12-19) [1983]. "Microsoft DOS V1.1 and V2.0: /msdos/v20source/SKELIO.TXT, /msdos/v20source/HRDDRV.ASM" (http://www.computerhistory.org/atchm/microsoft-research-license-agreement-msdos-v1-1-v2-0/). Computer History Museum, Microsoft. Retrieved 2014-03-25. (NB. While the publishers claim this would be MS-DOS 1.1 and 2.0, it actually is SCP MS-DOS 1.25 and a mixture of Altos MS-DOS 2.11 and TeleVideo PC DOS 2.11.)
32. Zbikowski, Mark; Allen, Paul; Ballmer, Steve; Borman, Reuben; Borman, Rob; Butler, John; Carroll, Chuck; Chamberlain, Mark; Chell, David; Colee, Mike; Courtney, Mike; Dryfoos, Mike; Duncan, Rachel; Eckhardt, Kurt; Evans, Eric; Farmer, Rick; Gates, Bill; Geary, Michael; Griffin, Bob; Hogarth, Doug; Johnson, James W.; Kermaani, Kaamel; King, Adrian; Koch, Reed; Landowski, James; Larson, Chris; Lennon, Thomas; Lipkie, Dan; McDonald, Marc; McKinney, Bruce; Martin, Pascal; Mathers, Estelle; Matthews, Bob; Melin, David; Mergentime, Charles; Nevin, Randy; Newell, Dan; Newell, Tani; Norris, David; O'Leary, Mike; O'Rear, Bob; Olsson, Mike; Osterman, Larry; Ostling, Ridge; Pai, Sunil; Paterson, Tim; Perez, Gary; Peters, Chris; Petzold, Charles; Pollock, John; Reynolds, Aaron; Rubin, Darryl; Ryan, Ralph; Schulmeisters, Karl; Shah, Rajen; Shaw, Barry; Short, Anthony; Slivka, Ben; Smirl, Jon; Stillmaker, Betty; Stoddard, John; Tillman, Dennis; Whitten, Greg; Yount, Natalie; Zeck, Steve (1988). "Technical advisors". *The MS-DOS Encyclopedia: versions 1.0 through 3.2*. By Duncan, Ray; Bostwick, Steve; Burgoyne, Keith; Byers, Robert A.; Hogan, Thom; Kyle, Jim; Letwin, Gordon; Petzold, Charles; Rabinowitz, Chip; Tomlin, Jim; Wilton, Richard; Wolverton, Van; Wong, William; Woodcock, JoAnne (Completely reworked ed.). Redmond, Washington, USA: Microsoft Press. ISBN 1-55615-049-0. LCCN 87-21452 (https://lccn.loc.gov/87-21452). OCLC 16581341 (https://www.worldcat.org/oclc/16581341). (xix+1570 pages; 26 cm) (NB. This edition was published in 1988 after extensive rework of the withdrawn 1986 first edition by a different team of authors. [13] (https://www.pcjs.org/pubs/pc/reference/microsoft/mspl13/msdos/encyclopedia/))
33. "Detailed Explanation of FAT Boot Sector" (http://support.microsoft.com/kb/140418). *Microsoft Knowledge Base*. 2003-12-06. Retrieved 2011-10-16.
34. Lai, Robert S.; The Waite Group (1987). ***Writing MS-DOS Device Drivers*** (https://archive.org/details/writingmsdosdevi00lair_0) (2nd ed.). Addison Wesley. ISBN 0-201-60837-5.
35. Paterson, Tim; Microsoft (2013-12-19) [1983]. "Microsoft DOS V1.1 and V2.0: /msdos/v20source/DEVDRIV.txt" (http://www.computerhistory.org/atchm/microsoft-research-license-agreement-msdos-v1-1-v2-0/). Computer History Museum, Microsoft. Retrieved 2014-03-25. (NB. While the publishers claim this would be MS-DOS 1.1 and 2.0, it actually is SCP MS-DOS 1.25 and a mixture of Altos MS-DOS 2.11 and TeleVideo PC DOS 2.11.)
36. Tim Paterson (1983). "An Inside Look at MS-DOS" (https://web.archive.org/web/20110720115141/http://patersontech.com/Dos/Byte/InsideDos.htm#InsideDos_44). Byte. Archived from the original (http://www.patersontech.com/dos/Byte/InsideDos.htm#InsideDos_44) on 2011-07-20. Retrieved 2011-07-18. "The numbering starts with 2; the first two numbers, 0 and 1, are reserved."
37. *PORT-DOS - Userprompt Guide for Apricot Portable*. User-Prompt Guides, UK ([14] (http://actapricot.org/support/apricot_user_prompt_guide_portable.pdf)).
38. John C. Elliott (1998). *DOSPLUS disc formats*. ([15] (http://www.seasip.demon.co.uk/Cpm/fat_formats.html)).
39. *The BBC Master 512*. Yellow Pig's BBC Computer Pages ([16] (http://www.cowsarenotpurple.co.uk/bbccomputer/master512/format.html)).
40. Digital Equipment Corporation. Rainbow 100 MS-DOS 2.01 Technical Documentation Volume 1 (QV025-GZ), Microsoft MS-DOS Operating System BIOS Listing (AA-X432A-TV), Universal Disk Driver, Page 1-17. 1983.
41. "Detailed Explanation of FAT Boot Sector" (http://www.dewassoc.com/kbase/hard_drives/boot_sector.htm). DEW Associates Corporation. 2002. Retrieved 2011-10-16.
42. Daniel B. Sedory. *Detailed Notes on the "Dirty Shutdown Flag" under MS-Windows*. 2001-12-04. ([17] (http://thestarman.narod.ru/DOS/DirtyShutdownFlag.html)).
43. "Windows 98 Resource Kit - Chapter 10 - Disks and File Systems" (https://technet.microsoft.com/en-us/library/cc768180.aspx). Microsoft TechNet. 1998. Retrieved 2012-07-16.
44. Peter Norton (1986). *Inside the IBM PC, Revised and Enlarged*, Brady. ISBN 0-89303-583-1, p. 157.
45. Andries Brouwer. "FAT under Linux" (http://www.win.tue.nl/~aeb/linux/fs/fat/fat-2.html).
46. Andries Brouwer (2002-09-20). "FAT" (http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html#ss1.3). Retrieved 2012-01-11.

47. "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 or larger"
48. 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. [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.]
49. Chen, Raymond (July 2006). "Microsoft TechNet: A Brief and Incomplete History of FAT32" (http://www.microsoft.com/technet/technetmag/issues/2006/07/WindowsConfidential/). Microsoft TechNet Magazine.
50. Les Bell; Associates Pty Ltd (1996-09-02) [1990]. "OS/2 High Performance File System" (https://web.archive.org/web/20140301101837/http://www.lesbell.com.au/hpfstest.html). *PC Support Advisor*. Archived from the original (http://www.lesbell.com.au/hpfstest.html) on 2014-03-01. Retrieved 2014-06-24.
51. Bridges, Dan (February 1996). "Inside the High Performance File System - Part 2/6: Introduction" (http://www.edm2.com/0411/hpfs2.html). *Significant Bits, Brisbug PC User Group Inc*. Retrieved 2014-06-24.
52. Seattle Computer Products (1981). "SCP 86-DOS 1.0 Addendum" (http://bitsavers.informatik.uni-stuttgart.de/pdf/seattleComputer/86-DOS_1.0_Addendum.pdf) (PDF). Retrieved 2013-03-10.
53. Paul, Matthias R. (1997-07-30) [1994-05-01]. *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). *MPDOSTIP*. Release 157 (in German) (3 ed.). Archived (https://web.archive.org/web/20161105172944/http://www.antonis.de/dos/dos-tuts/mpdostip/html/nwdostip.htm) from the original on 2016-11-05. Retrieved 2012-01-11. (NB. 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 file.) [18] (https://web.archive.org/web/20190601152204/https://www.sac.sk/download/text/mpdostip.zip)
54. IBM. *4690 OS User's Guide Version 5.2*, IBM document SC30-4134-01, 2008-01-10 ([19] (ftp://ftp.software.ibm.com/software/retail/pubs/sw/opsys/4690/ver5r2/bsf1_UG_mst.pdf)).
55. Paterson, Tim; Microsoft (2013-12-19) [1983]. "Microsoft DOS V1.1 and V2.0: /msdos/v20source/FORMAT.TXT" (http://www.computerhistory.org/atchm/microsoft-research-license-agreement-msdos-v1-1-v2-0/). Computer History Museum, Microsoft. Retrieved 2014-03-25. (NB. While the publishers claim this would be MS-DOS 1.1 and 2.0, it actually is SCP MS-DOS 1.25 and a mixture of Altos MS-DOS 2.11 and TeleVideo PC DOS 2.11.)
56. Shustek, Len (2014-03-24). "Microsoft MS-DOS early source code" (http://www.computerhistory.org/atchm/microsoft-ms-dos-early-source-code/). Software Gems: The Computer History Museum Historical Source Code Series. Retrieved 2014-03-29. (NB. While the author claims this would be MS-DOS 1.1 and 2.0, it actually is SCP MS-DOS 1.25 and a mixture of Altos MS-DOS 2.11 and TeleVideo PC DOS 2.11.)
57. Levin, Roy (2014-03-25). "Microsoft makes source code for MS-DOS and Word for Windows available to public" (https://web.archive.org/web/20140328094124/http://blogs.technet.com/b/microsoft_blog/archive/2014/03/25/microsoft-makes-source-code-for-ms-dos-and-word-for-windows-available-to-public.aspx). *Official Microsoft Blog*. Archived from the original (http://blogs.technet.com/b/microsoft_blog/archive/2014/03/25/microsoft-makes-source-code-for-ms-dos-and-word-for-windows-available-to-public.aspx) on 2014-03-28. Retrieved 2014-03-29. (NB. While the author claims this would be MS-DOS 1.1 and 2.0, it actually is SCP MS-DOS 1.25 and a mixture of Altos MS-DOS 2.11 and TeleVideo PC DOS 2.11.)
58. 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.
59. John C. Elliott (1998). *CP/M 4.1 disc formats*. ([20] (http://www.seasip.demon.co.uk/Cpm/format41.html)): "CP/M 4.1 (DOS Plus [1.2]) allows the use of two file systems - 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 file system 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."
60. vinDaci (1998-01-06). "Long Filename Specification" (https://web.archive.org/web/20010420094054/http://www.teleport.com/~brainy/lfn.htm). Archived from the original (http://www.teleport.com/~brainy/lfn.htm) on 2001-04-20. Retrieved 2007-03-13.
61. Henk Kelder. *FAT32.TXT for FAT32.IFS version 0.74*. ("Archived copy" (https://web.archive.org/web/20120330171510/http://macarlo.com/fat32v074.htm). Archived from the original (http://macarlo.com/fat32v074.htm) on 2012-03-30. Retrieved 2012-01-14.). Comment: This older version of the README file still discusses the old `0xEA` and `0xEC` magic values.
62. Henk Kelder (2003). *FAT32.TXT for FAT32.IFS version 0.9.13."* (*[21] (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".*
63. Netlabs. *FAT32.IFS Wiki and Sources*. ([22] (http://svn.netlabs.org/fat32/wiki/WikiStart)).
64. IBM. *4690 OS Programming Guide Version 5.2*, IBM document SC30-4137-01, 2007-12-06 ([23] (ftp://ftp.software.ibm.com/software/retail/pubs/sw/opsys/4690/ver5r2/bsi1_PG_mst.pdf)).
65. *OpenDOS Developer's Reference Series — System and Programmer's Guide — Programmer's Guide* (https://web.archive.org/web/20171007025631/http://www.drdos.net/documentation/sysprog/httoc.htm). Caldera, Inc. August 1997. Caldera Part No. 200-DODG-003. Archived from the original (http://www.drdos.net/documentation/sysprog/httoc.htm) on 2017-10-07. Retrieved 2014-05-20. (Printed in the UK.)
66. Bob Eager, Tavi Systems (2000-10-28). *Implementation of extended attributes on the FAT file system*. ([24] (http://www.tavi.co.uk/os2pages/eadata.html)).
67. IBM (2003). *Information about 4690 OS unique file distribution attributes*, IBM document R1001487, 2003-07-30. ("Archived copy" (https://web.archive.org/web/20140521070339/http://www-01.ibm.com/support/docview.wss?uid=pos1R1001487). Archived from the original (http://www-01.ibm.com/support/docview.wss?uid=pos1R1001487) on 2014-05-21. Retrieved 2014-05-20.): "[...] 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."
68. IBM. *4690 save and restore file distribution attributes*. IBM document R1000622, 2010-08-31 ("Archived copy" (https://web.archive.org/web/20140521070536/http://www-01.ibm.com/support/docview.wss?uid=pos1R1000622). Archived from the original (http://www-01.ibm.com/support/docview.wss?uid=pos1R1000622) on 2014-05-21. Retrieved 2014-05-20.).
69. Natuerlich! (1992-03-24). "Getting longer filenames out of GEMDOS" (https://groups.google.com/forum/?_escaped_fragment_=topic/comp.sys.atari.st.tech/ADwk_y6-nYg#!topic/comp.sys.atari.st.tech/ADwk_y6-nYg). comp.sys.atari.st.tech. Retrieved 2014-05-05.
70. Torvalds, Linus (1992-12-23). "Long filenames" (https://groups.google.com/forum/?_escaped_fragment_=topic/comp.os.minix/0rgZpprg_Eo#!topic/comp.os.minix/0rgZpprg_Eo). comp.os.minix. Retrieved 2014-05-05.
71. "mount(8): mount file system" (http://linux.die.net/man/8/mount). *Linux man page*.

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 (https://web.archive.org/web/20131220004435/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* (https://web.archive.org/web/20060307082555/http://www.microsoft.com/technet/prodtechnol/winxppro/reskit/c13621675.mspx): Microsoft Technet, copy made by Internet Archive Wayback Machine (https://archive.org/)
- 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 GB 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 (https://web.archive.org/web/20050319235548/http://www.microsoft.com/resources/documentation/Windows/XP/all/reskit/en-us/prkc_fil_cycz.asp). Copy made by Internet Archive Wayback Machine (https://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 (https://web.archive.org/web/20140528122441/http://www.beginningtoseethelight.org/fat16/)
