Design of the FAT file system

The **FAT file system** is a file system used on MS-DOS and Windows 9x family of operating systems.[3] It continues to be used on mobile devices and embedded systems, and thus is a well-suited file system for data exchange between computers and devices of almost any type and age from 1981 through to the present.

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 IDs	MBR/EBR: FAT12: `0x01` e.a. FAT16: `0x040x060x0E` e.a. FAT32: `0x0B0x0C` e.a. exFAT: `0x07` e.a. BDP: `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7`
Structures	
Directory contents	Table

Structure

A FAT file system is composed of four regions:

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 no. 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 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

FAT file system regions, which includes four main regions

Region	Size in sectors	Contents	Notes
Reserved sectors	(number of reserved sectors)	Boot Sector	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
		FS Information Sector (FAT32 only)	

			OS/2.
		More reserved sectors (optional)	The total count of reserved sectors is indicated by a field inside the Boot Sector, and is usually 32 on FAT32 file systems.[4] 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	(number of FATs) * (sectors per FAT)	File Allocation Table #1	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. The first two clusters (cluster 0 and 1) in the map contain special values.
		File Allocation Table #2 ... (optional)	
Root Directory Region	(number of root entries * 32) / (bytes per sector)	Root Directory (FAT12 and FAT16 only)	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	(number of clusters) * (sectors per cluster)	Data Region (for files and directories) ... (to end of partition or disk)	This is where the actual file and directory data is stored and takes up most of the partition. 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, some entries on Atari ST boot sectors) and the FAT(s).[5] 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.

Reserved sectors area

Boot Sector

On non-partitioned storage 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 storage devices such as hard disks, the Boot Sector is the first sector of a partition, as specified in the partition table of the device.

Common structure of the Boot Sector used by most FAT versions for IBM compatible x86 machines since DOS 2.0

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`[6][7] as seen since DOS 3.0[nb 1]—and on DOS 1.1[8][9]) or a near jump (`0xE9 0x?? 0x??`[6][7] 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??`)[6][7][10] on removable disks. On hard disks, DR DOS additionally accepts the swapped JMPS sequence starting with a NOP (`0x90 0xEB 0x??`), [10] 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.[11][12][13] 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,

Byte offset	Length (bytes)	Contents
		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).[14] 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␣␣␣␣"[15] 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

Byte offset	Length (bytes)	Contents
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).[4][nb 2] 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[16]). 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[8][9] 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.[17][nb 3] 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

Byte offset	Length (bytes)	Contents
		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 2] 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??`).[5] 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[5] (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 (TOS.IMG[5]) fails, the code can return to the Atari ST BIOS with a 68000 RTS (opcode `0x4E75` with big-endian byte sequence `0x4E 0x75`[nb 2]) 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.[17] This checksum entry can be used to align the checksum accordingly.[nb 3] If the logical sector size is larger than 512 bytes, the remainder is not included in the checksum and is typically zero-filled.[17] Since some PC operating systems erroneously do not accept FAT formatted floppies if the `0x55 0xAA`[nb 2] 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 3] 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; 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 65,535 logical sectors is 32 bytes, or 64 bytes for more than 65,535 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.[17] 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 4] 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.[18] 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.

Sector offset	BPB offset	Length (bytes)	Contents
			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.[19]:INT 21h AX=53h
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.[20] This extension, however, is not supported by mainstream operating systems,[20] as it can conflict with other uses of the cluster 1 entry for maintenance flags, the current end-of-chain-marker, or TFAT extensions.

Sector offset	BPB offset	Length (bytes)	Contents
			This value must be adjusted so that directory entries always consume full logical sectors, given that 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,[6] the maximum value supported by MS-DOS/PC DOS on hard disks is 512.[6] 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):[21][22][23][nb 1] **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)[13] **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**[24][25][26] ▪ 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.[27] **0xF4**

Sector offset	BPB offset	Length (bytes)	Contents
			▪ Double density (Altos MS-DOS 2.11 only)[28] **0xF5** ▪ Fixed disk, 4-sided, 12 sectors per track (1.95? MB) (Altos MS-DOS 2.11 only)[28] **0xF8** ▪ Fixed disk (i.e., typically a partition on a hard disk). (since DOS 2.0) [29][30] ▪ 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[7] 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)[13] ▪ Single sided (Altos MS-DOS 2.11 only)[28] **0xF9**[24][25][26] ▪ 3.5-inch double sided, 80 tracks per side, 9 sectors per track (720 KB) (since DOS 3.2)[29] ▪ 3.5-inch double sided, 80 tracks per side, 18 sectors per track (1440 KB) (DOS 3.2 only)[29] ▪ 5.25-inch double sided, 80 tracks per side, 15 sectors per track (1200 KB, known as "1.2 MB") (since DOS 3.0)[29] ▪ Single sided (Altos MS-DOS 2.11 only)[28] **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[31] 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)[29] **0xFD** ▪ 5.25-inch double sided, 40 tracks per side, 9 sectors per track (360 KB) (since DOS 2.0)[29] ▪ 8-inch double sided, 77 tracks per side, 26 sectors per track, 128

Sector offset	BPB offset	Length (bytes)	Contents
			bytes per sector (500.5 KB) ▪ (8-inch double sided, (single and) double density (DOS 1)[29]) **0xFE** ▪ 5.25-inch single sided, 40 tracks per side, 8 sectors per track (160 KB) (since DOS 1.0)[29][32] ▪ 8-inch single sided, 77 tracks per side, 26 sectors per track, 128 bytes per sector (250.25 KB)[28][32] ▪ 8-inch double sided, 77 tracks per side, 8 sectors per track, 1024 bytes per sector (1232 KB)[32] ▪ (8-inch single sided, (single and) double density (DOS 1)[29]) **0xFF** ▪ 5.25-inch double sided, 40 tracks per side, 8 sectors per track (320 KB) (since DOS 1.1)[29][32] ▪ Hard disk (Sanyo 55x DS-DOS 2.11 only)[13] 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.[6] On removable drives, DR-DOS will assume the presence of a BPB if this value is greater or equal to `0xF0`,[6] 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.[7] 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.[28] 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,[4] 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,[4] 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,[4] 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,[4] 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.[24][25][26] 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

Sector offset	BPB offset	Length (bytes)	Contents
			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:[24][25][26]

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 the SYS command 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. ▪ VGA-Copy 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).[30] Should be set to 0 by formatting tools.[24][25][26] See also: Bitflags in the second cluster entry in the FAT.
0x026	0x1B	1	Extended boot signature. (Should be 0x29[24][25][26][21] 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

Sector offset	EBPB offset	Length (bytes)	Contents
			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 5] and INT 21h/AH=2Ch (get system time)[nb 5] 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 6] 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. Fortunately, under normal circumstances (sector size of 512 bytes), this cannot happen, as a FAT32 file system has at most 0xffffff6 = 268435446 clusters. One Fat sector fits 512 / 4 = 128 cluster descriptors. So at most only ceil(268435446 / 128) = 2097152 = 0x200000 sectors would be needed, making third byte of the number of FAT sectors 0x20 at most, which is less than the forbidden 0x28 and 0x29 values.
0x028	0x1D	2	Drive description / mirroring flags (bits 3-0: zero-based number of active FAT, if bit 7 set.[4] 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`.[4] 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[33]) 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

Sector offset	FAT32 EBPB offset	Length (bytes)	Contents
			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[19] (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.[4] 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[4] (and/or 0xFFFF[19]) are reserved and indicate that no backup sector is available.
0x034	0x29	12	Reserved (may be changed to format filler byte 0xF6[nb 7] 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 8] 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

Sector offset	FAT32 EBPB offset	Length (bytes)	Contents
			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 7] 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[15] 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 1]

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,[7] 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) [22][23]	**0xFF**		**0xFE**			**0xFD**			**0xFC**	0
Size	8"	5.25"	8"	8"	5.25"	8"	8"	5.25"	5.25"	5 3
Density	?	DD 48tpi	SD	DD	DD 48tpi	SD	SD	DD 48tpi	DD 48tpi	
Modulation	?	MFM	FM	MFM	MFM	FM	FM	MFM	MFM	M
Formatted capacity (KB)	?	320	250 ("old") [28][32]	1200	160	250 ("new") [28][32]	500	360	180	6
Cylinders (CHS)	77	40	77	77	40	77	77	40	40	8
Physical sectors / track (BPB offset 0x0D)	?	8	26	8	8	26	26	9	9	8
Number of heads (BPB offset 0x0F)	?	2	1[28][32]	2[7][22][32] (1)	1	1[7][28][32]	2[22]	2	1	2
Byte payload / physical sector	?	512	128	1024	512	128	128	512	512	5
Bytes / logical sector (BPB offset 0x00)	?	512	128	1024	512	128	128	512	512	5
Logical sectors / cluster (BPB offset 0x02)	?	2	4	1	1	4	4	2	1	2
Reserved logical sectors (BPB offset 0x03)	?	1	1[28][32]	1	1	4[28][32]	4	1	1	1
Number of FATs	?	2	2	2	2	2	2	2	2	2

(BPB offset 0x05)										
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)	1 s
Total logical sectors (BPB offset 0x08)	?	640	2002[28][32]	1232[22][32] (616[7])	320	2002[7][28][32]	4004[22]	720	360	1
Logical sectors / FAT (BPB offset 0x0B)	?	1	6[28][32]	2	1	6[28][32]	6?[22]	2	2	2
Hidden sectors (BPB offset 0x11)	?	0	3[22] (0[7])	0	0	0	0	0	0	0
Total number of clusters	?	315	497	1227	313	?	997?[22]	354	351	
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+
First physical sector (CHS)	?	1	1	1	1	1	1	1	1	
DRIVER.SYS /F:n	?	0	3	4	0	?	3	0	0	
BPB Presence	?	?	?	?	?	?	?	?	?	
Support	?	DOS 1.1[32]	DOS 1.0[28][32]	DOS 2.0	DOS 1.0[32]	?[28][32]	DOS 2.0	DOS 2.0	DOS 2.0	

Microsoft recommends to distinguish between the two 8-inch formats for FAT ID `0xFE` by trying to read

of a single-density address mark. If this results in an error, the medium must be double-density.[23]

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[34] 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.[34] 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.[34][35]
- 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.[34]

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.[35] 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.[35][36] 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).[36] 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.[35][36]
- 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.[35][36]

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.[37]

FS Information Sector

The "FS Information Sector" was introduced in FAT32[38] 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 less than 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`)[4][nb 2] (All four bytes should match before the contents of this sector should be assumed to be in valid format.)

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

If this sector is present on a FAT32 volume, the minimum allowed logical sector size is 512 bytes, whereas otherwise it would be 128 bytes. Some FAT32 implementations support a slight variation of Microsoft's specification by making the FS information sector optional by specifying a value of `0xFFFF`[19] (or `0x0000`) in the entry at offset `0x030`.

FAT region

File Allocation Table

Cluster map

A volume's data area is divided 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 drive; typical cluster sizes range from 2 to 32 KiB.[39]

Each file may occupy one or more clusters depending on its size. Thus, a file is represented by a chain of clusters (referred to as a singly linked list). 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 (e.g. cluster 0) and the 12 most significant bits the second (e.g. 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; they are cleared during formatting 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 four 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, the 4 MSB bits are zero) of this entry are always 1. These values were arranged so that the entry would also function as a "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).[40] (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)[41] 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.[27] 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 9][41] (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)[41] is cleared, the operating system has encountered disk I/O errors on startup,[41] 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.[27][41] (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[33]). 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`).[20] This extension, however, is not supported by mainstream operating systems,[20] as it conflicts 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,[33] 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.[42][6] 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.[7]
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).[42][6] 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 10] - 0xFF5 (0xFF1 - 0xFF5)	0xFFF0 - 0xFFF5	0x?FFFFFF0 - 0x?FFFFFF5	Reserved in some contexts,[43] or also used[24][25][26][4][44] 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,[6] and should avoid allocating them for files otherwise. MS-DOS/PC DOS 3.3 and higher treats a value of 0xFF0[nb 10][6] on FAT12 (but not on FAT16 or FAT32) volumes as additional end-of-chain marker similar to 0xFF8-0xFFF.[6] 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.[24][25][26][4][21][44] (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