How FAT Works

Updated: March 28, 2003

Applies To: Windows Server 2003, Windows Server 2003 R2, Windows Server 2003 with SP1, Windows Server 2003 with SP2

In this section

A file system is a required part of the operating system on a volume that determines how files are named, stored, and organized. A file system manages files and folders, and the information needed to locate and access these items by local and remote users.

Microsoft Windows Server 2003 supports the file allocation table (FAT) file system on basic disks and on readable/writable disks. Basic disks and volumes are the storage types most often used with Windows operating systems.

During the format of a volume, you can choose the type of file system for the volume. When you choose the FAT file system, the formatting process places key file data structures on the volume to tell the hard disk which file system the operating system uses.

FAT Architecture

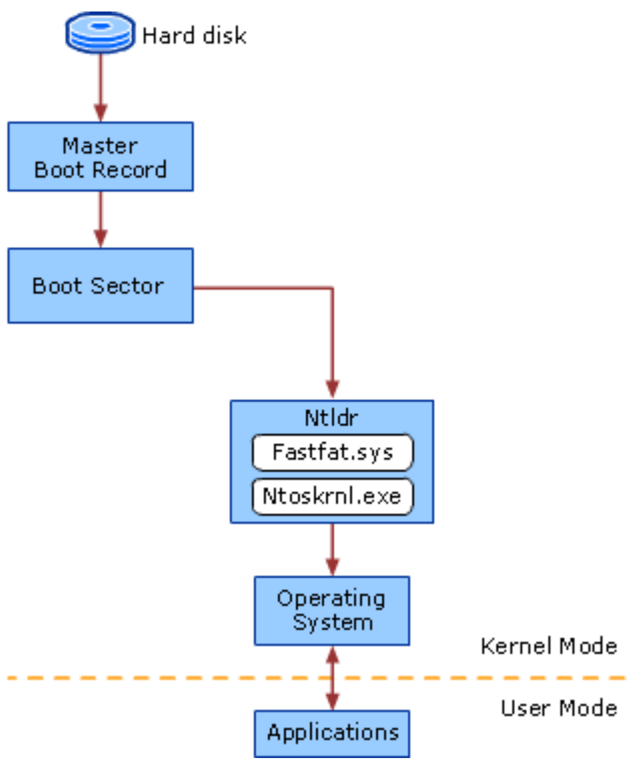
During format and setup of a volume file system on a hard disk, a master boot record (MBR) is created. The MBR contains a small amount of executable code called the master boot code as well as a partition table for the disk. When a volume is mounted, the MBR executes the master boot code and transfers control to the boot sector on the disk, allowing the server to start the operating system on the file system of that specific volume.

Note

- The partition table contains a number of fields used to describe the partition. One of these fields is the System ID field, which defines the file system, such as FAT, on the partition. For FAT16 volumes, the system ID is 0x04; for FAT32 volumes, the system ID is 0x0B.

The following figure shows the architecture of this process.

FAT Architecture



The following table describes the components of the FAT file system.

FAT Components

Component	**Component Description**
Hard disk	Contains one or more partitions.
Boot sector	Bootable partition that stores information about the layout of the volume and the file system structures, as well as the boot code that loads Ntdlr.
Master Boot Record	Contains executable code that the system BIOS loads into memory. The code scans the MBR to find the partition table to determine which partition is the active, or bootable, partition.
Ntldlr.dll	Switches the CPU to protected mode, starts the file system, and then reads the contents of the Boot.ini file. This information determines the startup options and initial boot menu selections.
Fastfat.sys	System file driver for FAT16 and FAT32.
Ntoskrnl.exe	Extracts information about which system device drivers to load and the load order.
Kernel Mode	The processing mode that allows code to have direct access to all hardware and memory in the system.

User Mode	The processing mode in which applications run.

FAT Physical Structure

The following information describes the physical structure of the FAT file system, and includes information about how clusters affect the FAT file system, as well as descriptions of the partition table.

Clusters and Sectors on a FAT Volume

A cluster (or allocation unit) is the smallest amount of disk space that can be allocated to hold a file. All file systems used by Windows Server 2003 organize hard disks based on cluster size, which is determined by the number of sectors (units of storage on a hard disk) that the cluster contains. For example, on a disk that uses 512-byte sectors, a 512-byte cluster contains one sector, whereas a 4-kilobyte (KB) cluster contains eight sectors.

Computers access certain sectors on a hard disk during startup to determine which operating system to start and where the partitions are located. The data stored on these sectors varies depending on the computer platform.

Sequence of Clusters on a FAT Volume

Clusters are numbered sequentially from the beginning of the partition into logical cluster numbers. Because FAT file system data clusters are located after the BIOS Parameter Blocks (BPB), reserved sectors, and two FAT structures, FAT formatting cannot guarantee that data clusters are aligned on a cluster boundary.

Limitations of Cluster Sizes on a FAT Volume

Because FAT16 and FAT32 each use different cluster sizes depending on the size of the volume, each file system has a maximum number of clusters it can support. The smaller the cluster size, the more efficiently a disk stores information because unused space within a cluster cannot be used by other files. And the more clusters a file system supports, the larger the volumes you can create and format by using a particular file system.

The size of each cluster is a multiple of the sector size. The numerals in the names, FAT16 and FAT32 refer to the number of bits required for a file allocation table entry as follows:

- FAT16 uses a 16-bit file allocation table entry (216 clusters).
- FAT32 uses a 32-bit file allocation table entry. However, FAT32 reserves the first 4 bits of a FAT32 file allocation table entry, which means FAT32 has a theoretical maximum of 2^{28} clusters.
- Floppy disks are always formatted as FAT.

The following table provides a comparison of FAT16 and FAT32 volume and default cluster sizes.

Default FAT Cluster Sizes

Volume Size	FAT16 Cluster Size	FAT32 Cluster Size

7 megabytes (MB)–16 MB	2 KB	Not supported
17 MB–32 MB	512 bytes	Not supported
33 MB–64 MB	1 KB	512 bytes
65 MB–128 MB	2 KB	1 KB
129 MB–256 MB	4 KB	2 KB
257 MB–512 MB	8 KB	4 KB
513 MB–1,024 MB	16 KB	4 KB
1,025 MB–2 gigabytes (GB)	32 KB	4 KB
2 GB–4 GB	64 KB	4 KB
4 GB–8 GB	Not supported	4 KB
8 GB–16 GB	Not supported	8 KB
16 GB–32 GB	Not supported	16 KB
32 GB–2 terabytes	Not supported	Not supported

Windows Server 2003 formats FAT32 volumes up to 32 GB regardless of cluster size. To format volumes larger than 32 GB, you must use NTFS. However, Windows Server 2003 can mount FAT32 volumes larger than 32 GB that were created by other operating systems.

Maximum Sizes on FAT Volumes

Before you format a volume, you can evaluate the types of files to be stored on the volume so that you can determine whether to use the default cluster size. If you format a volume but do not specify a cluster size, default values are used. If you want to change the cluster size after the volume is formatted, you must reformat the volume.

FAT16 and FAT32 have the following size limitations:

- FAT volumes smaller than 16 MB are formatted as FAT12.
- FAT16 volumes larger than 2 GB are not accessible from computers running MS-DOS, Windows 95, Windows 98, Windows Millennium Edition (Me), and many other operating systems. This limitation occurs because these operating systems do not support cluster sizes larger than 32 KB, which results in the 2 GB limit.
- In theory, FAT32 volumes can be about 8 terabytes; however, the maximum FAT32 volume size that Windows Server 2003 can format is 32 GB. Therefore, you must use NTFS to format volumes larger than 32 GB. However, Windows Server 2003 can read and write to larger FAT32 volumes formatted by other operating systems.

The largest possible file for a FAT32 volume is 4 GB minus 1 byte. FAT32 contains 4 bytes per cluster in the file allocation table; FAT16 contains 2 bytes per cluster; and FAT12 contains 1.5 bytes per cluster. A FAT32 volume must have at least 65,527 clusters

FAT16 supports a maximum of 65,524 clusters per volume. The following table lists FAT16 size limits.

FAT16 Size Limits

Description	Limit
Maximum file size	Tested: 4 GB minus 1 byte (2^{32} bytes minus 1 byte)
Maximum volume size	Tested: 4 GB
Files per volume	Approximately 65,536 (2^{16} files)
Maximum number of files and folders within the root folder	512 (Long file names can reduce the number of available files and folders in the root folder.)

A FAT32 volume must have a minimum of 65,527 clusters. Windows Server 2003 can format FAT32 volumes up to 32 GB, but it can mount larger FAT32 volumes created by other operating systems. The following table lists FAT32 size limits.

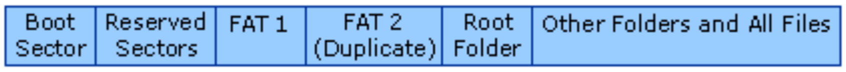
FAT32 Size Limits

Description	Limit
Maximum file size	Tested: 4 GB minus 1 byte (2^{32} bytes minus 1 byte)
Maximum volume size	Tested: 32 GB (implementation)
Files per volume	4,177,920
Maximum number of files and subfolders within a single folder	65,534 (The use of long file names can significantly reduce the number of available files and subfolders within a folder.)

Organization of a FAT Volume

A volume formatted with FAT is organized as illustrated in the following figure.

Organization of a FAT Volume



The following table describes each of the organizational structures on a FAT volume.

FAT Volume Components

Component	**Description**
Boot Sector	Contains the BIOS parameter block that stores information about the layout of the volume and the file system structures, as well as the boot code that loads Windows Server 2003.
Reserved Sectors	The number of sectors that precede the start of the first FAT, including the boot sector.
FAT 1	Original FAT.
FAT 2 (Duplicate)	Backup copy of the FAT.
Root folder	Describes the files and folders in the root of the partition.
Other folders and all files	Contains the data for the files and folders within the file system.

Boot Sectors on MBR and GPT Disks

On MBR disks (disks that contain a master boot record), the boot sector, which is located at the first logical sector of each partition, is a critical disk structure for starting your computer. It contains executable code and the data required by the code, including information that the file system uses to access the volume. The boot sector is created when you format a volume. At the end of the boot sector is a 2-byte structure called a signature word or end of sector marker, which is always set to 0x55AA. On computers running Windows Server 2003, the boot sector on the active partition loads into memory and starts Ntldr, which loads the boot menu if multiple versions of Windows are installed or loads the operating system if only one operating system is installed.

GUID partition table (GPT) disks are similar to MBR disks, except they use primary and backup partition structures to provide redundancy. These structures are located at the beginning and the end of the disk. GPT identifies these structures by their logical block address (LBA) rather than by their relative sectors.

A boot sector consists of the following elements:

- An x86-based CPU jump instruction.
- The original equipment manufacturer identification (OEM ID).

- The BIOS parameter block (BPB), a data structure.
- The extended BPB.
- The executable boot code (or bootstrap code) that starts the operating system.

All Windows Server 2003 boot sectors contain the preceding elements regardless of the type of disk (basic disk or dynamic disk).

Components of a Boot Sector

The MBR transfers CPU execution to the boot sector, so the first three bytes of the boot sector must be valid, executable x86-based CPU instructions. This includes a jump instruction that skips the next several nonexecutable bytes.

Following the jump instruction is the 8-byte OEM ID, a string of characters that identifies the name and version number of the operating system that formatted the volume. To preserve compatibility with MS-DOS, Windows Server 2003 records “MSDOS5.0” in this field on FAT16 and FAT32 disks.

Note

- You might also see the OEM ID "MSWIN4.0" on disks formatted by Windows 95 and "MSWIN4.1" on disks formatted by Windows 95 OEM Service Release 2 (OSR2), Windows 98, and Windows Me. Windows Server 2003 does not use the OEM ID field in the boot sector except for verifying NTFS volumes.

Following the OEM ID is the BPB, which provides information that enables the executable boot code to locate Ntldr. The BPB always starts at the same offset, so standard parameters are in a known location. Disk size and geometry variables are encapsulated in the BPB. Because the first part of the boot sector is an x86 jump instruction, the BPB can be extended in the future by appending new information at the end. The jump instruction needs only a minor adjustment to accommodate this change. The BPB is stored in a packed (unaligned) format.

FAT16 Boot Sector

The following table describes the sections of a boot sector of a volume formatted with the FAT16 file system.

Boot Sector Sections on a FAT16 Volume

Byte Offset	**Field Length**	**Field Name**
0x00	3 bytes	Jump instruction
0x03	8 bytes	OEM ID
0x0B	25 bytes	BPB
0x24	26 bytes	Extended BPB

0x3E	448 bytes	Bootstrap code
0x01FE	2 bytes	End of sector marker

The following example illustrates a hexadecimal printout of the boot sector on a FAT16 volume. The printout is formatted in three sections:

- Bytes 0x00– 0x0A are the jump instruction and the OEM ID (shown in bold print).
- Bytes 0x0B– 0x3D are the BPB and the extended BPB.
- The remaining section is the bootstrap code and the end of sector marker (shown in bold print).

```
Physical Sector: Cyl 0, Side 1, Sector 1
00000000: EB 3C 90 4D 53 44 4F 53  - 35 2E 30 00 02 40 01 00
.<.MSDOS5.0..@..
00000010: 02 00 02 00 00 F8 FC 00  - 3F 00 40 00 3F 00 00 00
........?.@.?...
00000020: 01 F0 3E 00 80 00 29 A8  - 8B 36 52 4E 4F 20 4E 41
..>...)..6RNO NA
00000030: 4D 45 20 20 20 20 46 41  - 54 31 36 20 20 20 33 C0   ME
FAT16   3.
00000040: 8E D0 BC 00 7C 68 C0 07  - 1F A0 10 00 F7 26 16 00
....|h......&..
00000050: 03 06 0E 00 50 91 B8 20  - 00 F7 26 11 00 8B 1E 0B   ....P..
..&.....
00000060: 00 03 C3 48 F7 F3 03 C8  - 89 0E 08 02 68 00 10 07
...H........h...
00000070: 33 DB 8F 06 13 02 89 1E  - 15 02 0E E8 90 00 72 57
3.............rW
00000080: 33 DB 8B 0E 11 00 8B FB  - 51 B9 0B 00 BE DC 01 F3
3.......Q.......
00000090: A6 59 74 05 83 C3 20 E2  - ED E3 37 26 8B 57 1A 52   .Yt...
...7&.W.R
000000A0: B8 01 00 68 00 20 07 33  - DB 0E E8 48 00 72 28 5B   ...h.
.3...H.r([
000000B0: 8D 36 0B 00 8D 3E 0B 02  - 1E 8F 45 02 C7 05 F5 00
.6...>....E.....
000000C0: 1E 8F 45 06 C7 45 04 0E  - 01 8A 16 24 00 EA 03 00
..E..E.....$....
000000D0: 00 20 BE 86 01 EB 03 BE  - A2 01 E8 09 00 BE C1 01   .
..............
000000E0: E8 03 00 FB EB FE AC 0A  - C0 74 09 B4 0E BB 07 00
.........t......
000000F0: CD 10 EB F2 C3 50 4A 4A  - A0 0D 00 32 E4 F7 E2 03
.....PJJ...2....
00000100: 06 08 02 83 D2 00 A3 13  - 02 89 16 15 02 58 A2 07
.............X..
```

```
00000110: 02 A1 13 02 8B 16 15 02  - 03 06 1C 00 13 16 1E 00
................
00000120: F7 36 18 00 FE C2 88 16  - 06 02 33 D2 F7 36 1A 00
.6........3..6..
00000130: 88 16 25 00 A3 04 02 A1  - 18 00 2A 06 06 02 40 3A
..%.......*...@:
00000140: 06 07 02 76 05 A0 07 02  - 32 E4 50 B4 02 8B 0E 04
...v....2.P.....
00000150: 02 C0 E5 06 0A 2E 06 02  - 86 E9 8B 16 24 00 CD 13
............$...
00000160: 0F 83 05 00 83 C4 02 F9  - CB 58 28 06 07 02 76 11
.........X(...v.
00000170: 01 06 13 02 83 16 15 02  - 00 F7 26 0B 00 03 D8 EB
..........&.....
00000180: 90 A2 07 02 F8 CB 42 4F  - 4F 54 3A 20 43 6F 75 6C   ......BOOT:
Coul
00000190: 64 6E 27 74 20 66 69 6E  - 64 20 4E 54 4C 44 52 0D   dn't find
NTLDR.
000001A0: 0A 00 42 4F 4F 54 3A 20  - 49 2F 4F 20 65 72 72 6F   ..BOOT: I/O
erro
000001B0: 72 20 72 65 61 64 69 6E  - 67 20 64 69 73 6B 0D 0A   r reading
disk..
000001C0: 00 50 6C 65 61 73 65 20  - 69 6E 73 65 72 74 20 61   .Please
insert a
000001D0: 6E 6F 74 68 65 72 20 64  - 69 73 6B 00 4E 54 4C 44   nother
disk.NTLD
000001E0: 52 20 20 20 20 20 20 00  - 00 00 00 00 00 00 00 00   R
.........
000001F0: 00 00 00 00 00 00 00 00  - 00 00 00 00 00 00 55 AA
..............U.
```

The following two tables illustrate the layout of the BPB and the extended BPB for FAT16 volumes. The sample values correspond to the data in this example of a hexadecimal printout of the boot sector on a FAT16 volume.

BPB Fields for FAT16 Volumes

Byte Offset	Field Length	Sample Value	Field Name and Definition
0x0B	2 bytes	00 02	**Bytes Per Sector**.The size of a hardware sector. Valid decimal values for this field are 512, 1024, 2048, and 4096.
0x0D	1 byte	40	**Sectors Per Cluster**.The number of sectors in a cluster. Because FAT16 can track only a limited number of clusters (up to 65,524), FAT16 supports large volumes by

			increasing the number of sectors per cluster. The default cluster size for a volume depends on the volume size. Valid decimal values for this field are 1, 2, 4, 8, 16, 32, 64, and 128.
0x0E	2 bytes	01 00	**Reserved Sectors**. The number of sectors that precede the start of the first FAT, including the boot sector. The value of this field is typically 1.
0x10	1 byte	02	**Number of FATs**. The number of copies of the FAT on the volume. The value of this field is typically 2.
0x11	2 bytes	00 02	**Root Entries**. The total number of 32-byte file and folder name entries that can be stored in the root folder of the volume. On a typical hard disk, the value of this field is 512. One entry can be used as a volume label, and files and folders with long names use multiple entries per file. The largest number of file and folder entries is typically 511; however, if long file names are used, entries usually run out before you reach that number.
0x13	2 bytes	00 00	**Small Sectors**. The number of sectors on the volume represented in 16 bits (< 65,536). For volumes larger than 65,536 sectors, this field has a value of zero and the **Large Sectors** field is used instead.
0x15	1 byte	F8	**Media Descriptor**. Provides information about the media being used. A value of 0xF8 indicates a hard disk and 0xF0 indicates a high-density 3.5-inch floppy disk. Media descriptor entries are a legacy of MS-DOS FAT16 and FAT12 disks and are not used in Windows Server 2003.
0x16	2 bytes	FC 00	**Sectors Per FAT**. The number of sectors occupied by each FAT on the volume. The computer uses this number and the number of FATs and reserved sectors to determine where the root directory begins. The computer can also determine where the user data area of the volume begins based on the number of entries in the root directory (512).
0x18	2 bytes	3F 00	**Sectors Per Track**. Part of the apparent disk geometry used on a low-level formatted disk.
0x1A	2 bytes	40 00	**Number of Heads**. Part of the apparent disk geometry used on a low-level formatted disk.
0x1C	4 bytes	3F 00 00 00	**Hidden Sectors**. The number of sectors on the volume before the boot sector. This value is used during the boot sequence to calculate the absolute offset to the root

			directory and data areas.
0x20	4 bytes	01 F0 3E 00	**Large Sectors**. If the value of the **Small Sectors** field is zero, this field contains the total number of sectors in the FAT16 volume. If the value of the **Small Sectors** field is not zero, the value of this field is zero.

Extended BPB Fields for FAT16 Volumes

Byte Offset	Field Length	Sample Value	Field Name and Definition
0x24	1 byte	80	**Physical Drive Number**. Related to the BIOS physical drive number. Floppy drives are identified as 0x00 and physical hard disks are identified as 0x80, regardless of the number of physical disk drives. Typically, this value is set prior to issuing an INT 13h BIOS call to specify the device to access. This value is only relevant if the device is a boot device.
0x25	1 byte	00	**Reserved**. FAT16 volumes are always set to zero.
0x26	1 byte	29	**Extended Boot Signature**. A field that must have the value 0x28 or 0x29 to be recognized by Windows Server 2003.
0x27	4 bytes	A8 8B 36 52	**Volume Serial Number**. A random serial number that is created when a volume is formatted and that helps to distinguish between disks.
0x2B	11 bytes	NO NAME	**Volume Label**. A field that was once used to store the volume label. The volume label is now stored as a special file in the root directory.
0x36	8 bytes	FAT16	Not used by Windows Server 2003.

FAT32 Boot Sector

The FAT32 boot sector is structurally very similar to the FAT16 boot sector, but the FAT32 BPBcontains additional fields. The FAT32 extended BPB uses the same fields as FAT16, but the offset addresses of these fields within the boot sector are different than those found in FAT16 boot sectors. Volumes formatted in FAT32 are not readable by operating systems that are incompatible with FAT32.

The following table describes the sections of a boot sector of a volume formatted with the FAT32 file system.

Boot Sector Sections on a FAT32 Volume

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump instruction
0x03	8 bytes	OEM ID
0x0B	53 bytes	BPB
0x40	26 bytes	Extended BPB
0x5A	420 bytes	Bootstrap code
0x01FE	2 bytes	End of sector marker

The following example illustrates a hexadecimal printout of the boot sector on a FAT32 volume. The printout is formatted in three sections:

- Bytes 0x00– 0x0A are the jump instruction and the OEM ID (shown in bold print).
- Bytes 0x0B– 0x59 are the BPB and the extended BPB.
- The remaining section is the bootstrap code and the end of sector marker (shown in bold print).

```
Physical Sector: Cyl 878, Side 0, Sector 1
00000000: EB 58 90 4D 53 44 4F 53  - 35 2E 30 00 02 10 24 00
.X.MSDOS5.0...$.
00000010: 02 00 00 00 00 F8 00 00  - 3F 00 FF 00 3F 00 00 00
........?...?...
00000020: 1D 91 11 01 2A 22 00 00  - 00 00 00 00 02 00 00 00
....*"..........
00000030: 01 00 06 00 00 00 00 00  - 00 00 00 00 00 00 00 00
................
00000040: 80 00 29 F1 9E 5E 5E 4E  - 4F 20 4E 41 4D 45 20 20   ..)..^^NO
NAME
00000050: 20 20 46 41 54 33 32 20  - 20 20 33 C9 8E D1 BC F4     FAT32
3.....
00000060: 7B 8E C1 8E D9 BD 00 7C  - 88 4E 02 8A 56 40 B4 08
{......|.N..V@..
00000070: CD 13 73 05 B9 FF FF 8A  - F1 66 0F B6 C6 40 66 0F
..s......f...@f.
00000080: B6 D1 80 E2 3F F7 E2 86  - CD C0 ED 06 41 66 0F B7
....?.......Af..
00000090: C9 66 F7 E1 66 89 46 F8  - 83 7E 16 00 75 38 83 7E
.f..f.F..~..u8.~
```

```
000000A0: 2A 00 77 32 66 8B 46 1C  - 66 83 C0 0C BB 00 80 B9
*.w2f.F.f.......
000000B0: 01 00 E8 2B 00 E9 48 03  - A0 FA 7D B4 7D 8B F0 AC
...+..H...}.}...
000000C0: 84 C0 74 17 3C FF 74 09  - B4 0E BB 07 00 CD 10 EB
..t.<.t.........
000000D0: EE A0 FB 7D EB E5 A0 F9  - 7D EB E0 98 CD 16 CD 19
...}....}.......
000000E0: 66 60 66 3B 46 F8 0F 82  - 4A 00 66 6A 00 66 50 06
f`f;F...J.fj.fP.
000000F0: 53 66 68 10 00 01 00 80  - 7E 02 00 0F 85 20 00 B4
Sfh.....~.... ..
00000100: 41 BB AA 55 8A 56 40 CD  - 13 0F 82 1C 00 81 FB 55
A..U.V@........U
00000110: AA 0F 85 14 00 F6 C1 01  - 0F 84 0D 00 FE 46 02 B4
.............F..
00000120: 42 8A 56 40 8B F4 CD 13  - B0 F9 66 58 66 58 66 58
B.V@......fXfXfX
00000130: 66 58 EB 2A 66 33 D2 66  - 0F B7 4E 18 66 F7 F1 FE
fX.*f3.f..N.f...
00000140: C2 8A CA 66 8B D0 66 C1  - EA 10 F7 76 1A 86 D6 8A
...f..f....v....
00000150: 56 40 8A E8 C0 E4 06 0A  - CC B8 01 02 CD 13 66 61
V@............fa
00000160: 0F 82 54 FF 81 C3 00 02  - 66 40 49 0F 85 71 FF C3
..T.....f@I..q..
00000170: 4E 54 4C 44 52 20 20 20  - 20 20 20 00 00 00 00 00   NTLDR
.....
00000180: 00 00 00 00 00 00 00 00  - 00 00 00 00 00 00 00 00
................
00000190: 00 00 00 00 00 00 00 00  - 00 00 00 00 00 00 00 00
................
000001A0: 00 00 00 00 00 00 00 00  - 00 00 00 00 0D 0A 43 61
..............Ca
000001B0: 6E 6E 6F 74 20 73 74 61  - 72 74 2E 20 20 52 65 6D   nnot start.
Rem
000001C0: 6F 76 65 20 6D 65 64 69  - 61 2E FF 0D 0A 44 69 73   ove
media....Dis
000001D0: 6B 20 65 72 72 6F 72 FF  - 0D 0A 50 72 65 73 73 20   k
error...Press
000001E0: 61 6E 79 20 6B 65 79 20  - 74 6F 20 72 65 73 74 61   any key to
resta
000001F0: 72 74 0D 0A 00 00 00 00  - 00 AC CB D8 00 00 55 AA
rt...........U.
```

The following two tables illustrate the layout of the BPB and the extended BPB for FAT32 volumes. The sample values correspond to the data in this example of a hexadecimal printout of the boot sector on a FAT32 volume.

BPB Fields for FAT32 Volumes

Byte Offset	Field Length	Sample Value	Field Name and Definition
0x0B	2 bytes	00 02	**Bytes Per Sector**. The size of a hardware sector. Valid decimal values for this field are 512, 1024, 2048, and 4096.
0x0D	1 byte	10	**Sectors Per Cluster**.The number of sectors in a cluster. The default cluster size for a volume depends on the volume size. Valid decimal values for this field are 1, 2, 4, 8, 16, 32, 64, and 128. The Windows Server 2003 implementation of FAT32 allows for creation of volumes up to a maximum of 32 GB. However, larger volumes created by other operating systems (Windows 95 OSR2 and later) are accessible in Windows Server 2003.
0x0E	2 bytes	24 00	**Reserved Sectors**. The number of sectors that precede the start of the first FAT, including the boot sector.
0x10	1 byte	02	**Number of FATs**. The number of copies of the FAT on the volume. The value of this field is always 2.
0x11	2 bytes	00 00	**Root Entries (FAT12/FAT16 only)**. For FAT32 volumes, this field must be set to zero.
0x13	2 bytes	00 00	**Small Sectors (FAT12/FAT16 only)**. For FAT32 volumes, this field must be set to zero.
0x15	1 byte	F8	**Media Descriptor**. Provides information about the media being used. A value of 0xF8 indicates a hard disk and 0xF0 indicates a high-density 3.5-inch floppy disk. Media descriptor entries are a legacy of MS-DOS FAT16 disks and are not used in Windows Server 2003.
0x16	2 bytes	00 00	**Sectors Per FAT (FAT12/FAT16 only)**. For FAT32 volumes, this field must be set to zero.
0x18	2 bytes	3F 00	**Sectors Per Track**. Contains the "sectors per track" geometry value for disks that use INT 13h. The volume is broken down into tracks by multiple heads and cylinders.
0x1A	2 bytes	FF 00	**Number of Heads**. Contains the "count of heads" geometry value for disks that use INT 13h. For

			example, on a 1.44-MB, 3.5-inch floppy disk this value is 2.
0x1C	4 bytes	3F 00 00 00	**Hidden Sectors**. The number of sectors on the volume before the boot sector. This value is used during the boot sequence to calculate the absolute offset to the root directory and data areas. This field is generally only relevant for media that are visible on interrupt 13h. It must always be zero on media that are not partitioned.
0x20	4 bytes	1D 91 11 01	**Large Sectors**. Contains the total number of sectors in the FAT32 volume.
0x24	4 bytes	2A 22 00 00	**Sectors Per FAT (FAT32 only)**. The number of sectors occupied by each FAT on the volume. The computer uses this number and the number of FATs and reserved sectors (described in this table) to determine where the root directory begins. The computer can also determine where the user data area of the volume begins based on the number of entries in the root directory.
0x28	2 bytes	00 00	Not used by Windows Server 2003.
0x2A	2 bytes	00 00	**File System Version (FAT32 only)**. The high byte is the major revision number; the low byte is the minor revision number. This field supports the ability to extend the FAT32 media type in the future with concern for old FAT32 drivers mounting the volume. Both bytes are zero in Windows Server 2003, Windows 2000, and Windows Me and earlier.
0x2C	4 bytes	02 00 00 00	**Root Cluster Number (FAT32 only)**. The cluster number of the first cluster of the root directory. This value is typically, but not always, 2.
0x30	2 bytes	01 00	**File System Information Sector Number (FAT32 only)**. The sector number of the File System Information (FSINFO) structure in the reserved area of the FAT32 volume. The value is typically 1. A copy of the FSINFO structure is kept in the Backup Boot Sector, but it is not kept up-to-date.
0x32	2 bytes	06 00	**Backup Boot Sector (FAT32 only)**. A value other than zero specifies the sector number in the reserved area of the volume where a copy of the boot sector is stored. The value of this field is typically 6. No other value is recommended.

0x34	12 bytes	00 00 00 00 00 00 00 00 00 00 00 00	**Reserved (FAT32 only)**. Reserved space for future expansion. The value of this field must always be zero.

Extended BPB Fields for FAT32 Volumes

Byte Offset	Field Length	Sample Value	Field Name and Definition
0x40	1 byte	80	**Physical Drive Number**. Related to the BIOS physical drive number. Floppy drives are identified as 0x00 and physical hard disks are identified as 0x80, regardless of the number of physical disk drives. Typically, this value is set prior to issuing an INT 13h BIOS call to specify the device to access. This value is only relevant if the device is a boot device.
0x41	1 byte	00	**Reserved**. FAT32 volumes are always set to zero.
0x42	1 byte	29	**Extended Boot Signature**. A field that must have the value 0x28 or 0x29 to be recognized by Windows Server 2003.
0x43	4 bytes	F1 9E 5E 5E	**Volume Serial Number**. A random serial number that is created when a volume is formatted and that helps to distinguish between disks.
0x47	11 bytes	NO NAME	**Volume Label**. A field that was once used to store the volume label. The volume label is now stored as a special file in the root directory.
0x52	8 bytes	FAT32	**System ID**. A text field with a value of FAT32.

FAT 1 and FAT 2

The file allocation table, FAT 1, identifies each cluster in the volume as one of the following:

- Unused
- Cluster in use by a file
- Bad cluster
- Last cluster in a file

Because the file allocation table is so important, a copy of the table, FAT 2, is maintained on the volume. This allows the file system driver or consistency-checking program to access the duplicate copy if it cannot access FAT 1.

FAT Root Folder

The root folder describes the files and folders in the root of the partition. It has a maximum number of available entries fixed at 512. The maximum number of entries on a floppy disk depends on the size of the disk.

FAT Root Folder Structure

Root Folder Entry	**Size**	**Description**
Name	11 bytes	Name in 8.3 format.
Attribute byte	1 byte	Information about the entry.
Creation time and date	5 bytes	Time and date file was created.
Last access date	2 bytes	Date file was last accessed.
Last modification time and date	4 bytes	Time and date file was last modified.
First cluster	2 bytes	Starting cluster number in the file allocation table.
File size	4 bytes	Size of the file.

Note

- Three bytes in each 32-byte folder entry are held in reserve.

The information stored in the root folder is used by all operating systems that support FAT. Windows Server 2003 can store additional timestamps in a FAT folder entry. These timestamps show when the file was created or last accessed.

The attribute byte for each entry in a folder describes what kind of entry it is. For example, one bit indicates that the entry is for a subfolder, and another bit marks the entry as a volume. Typically, the operating system controls the settings of these bits.

The attribute byte includes four bits that can be turned on or off by the user — archive, system, hidden, and read-only.

FAT Processes and Interactions

FAT is affected by many factors, such as cluster size, fragmentation level, and the use of programs such as antivirus software.

Mounting a FAT Volume

When mounting a FAT volume, the MBR executes code to start up the boot sector. The boot sector then executes additional code to mount the volume.

Master Boot Code Startup Process

The MBR contains a small amount of executable code called the master boot code, the disk signature, and the partition table for the disk. During startup, the master boot code performs the following activities:

1. Scans the partition table for the active partition.
2. Finds the starting sector of the active partition.
3. Loads a copy of the boot sector from the active partition into memory.
4. Transfers control to the executable code in the boot sector.

Boot Sector Startup Process

Computers use the boot sector to run instructions during startup. The initial startup process is summarized in the following steps:

1. The system BIOS and the CPU initiate the power-on self test (POST).
2. The BIOS finds the boot device, which is typically the first disk the BIOS finds, unless the controller is configured to boot from a different disk.
3. The BIOS loads the first physical sector of the boot device into memory and transfers CPU execution to that memory address.

If the boot device is on a hard disk, the BIOS loads the MBR. The master boot code in the MBR loads the boot sector of the active partition, and transfers CPU execution to that memory address. On computers that are running Windows Server 2003, the executable boot code in the boot sector finds Ntldr, loads it into memory, and transfers execution to that file.

Note

- Windows Server 2003 cannot start up from a spanned, striped, or RAID-5 volume on dynamic disks. These disk structures cannot be registered into the MBR partition table; therefore, a system volume that uses these structures cannot start.

If drive A contains a floppy disk, the system BIOS loads the first sector (the boot sector) of the disk into memory. If the disk is a startup disk (formatted by MS-DOS with core operating system files applied), the boot sector loads into memory and uses the executable boot code to transfer CPU execution to Io.sys, a core MS-DOS operating system file. If the floppy disk is not a startup disk, the executable boot code displays an error message.

Note

- These messages do not appear on normally functioning systems that are configured to look for the startup files on drive C first. On many computers, an option in the CMOS setup program allows the user to set the sequence of installed disks that the system searches to find the startup files.

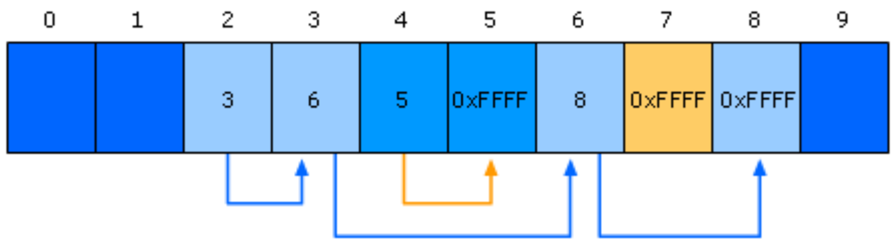
If you get similar errors when trying to start the computer from the hard disk, the boot sector might be corrupted.

Initially, the startup process is independent of disk format and operating system. The unique characteristics of operating and file systems become important when the boot sector's executable boot code starts.

File Processing on FAT Clusters

When a file is saved to a FAT-formatted volume, it stores file information in clusters on the hard disk. If a file requires space greater than the cluster’s size, FAT continues to store file information in the next available cluster until all information about the file is stored. The following figure shows an example of how FAT stores and retrieves file information from clusters.

File Processing on a FAT Volume



In this figure, 10 clusters (0 through 9) contain 3 files. One file occupies clusters 2, 3, 6, and 8; a second file occupies clusters 4 and 5; and a third file occupies cluster 7. The starting cluster number is the address of the first cluster used by the file. Each cluster contains a pointer to the next cluster in the file, or an end-of-file indicator (0xFFFF), which indicates that this cluster is the end of the file.

The attribute byte for each entry in a folder describes what kind of entry it is. For example, one bit indicates that the entry is for a subfolder, and another bit marks the entry as a volume. Typically, the operating system controls the settings of these bits.

The attribute byte includes four bits that can be turned on or off by the user — archive, system, hidden, and read-only.

FAT16 File System

FAT16 works best on small disks with simple folder structures. It is included in Windows Server 2003 for the following reasons:

- It provides backward compatibility in the form of an upgrade path for earlier versions of Windows-compatible products.
- It is compatible with most other operating systems.

Because FAT16 uses a larger cluster size, a FAT16 volume does not efficiently manage disk space when relatively small files are placed on it. A small file still uses a full cluster, even if the file does not require all the space allocated by the cluster. The cluster, once used by a file, cannot be used by any other file. This results in wasted space. Therefore, FAT16 is not recommended for volumes larger than 511 MB, and you cannot use

FAT16 on volumes larger than 4 gigabytes(GB).

In the file allocation table of a FAT16 volume, files are given the first available location on the volume.

FAT32 File System

Because FAT32 requires 4 bytes to store cluster values, many internal and on-disk data structures have been revised or expanded. Most programs are unaffected by these changes; however, disk tools that read the on-disk format must be updated to support FAT32.

The most significant difference between FAT16 and FAT32 is the maximum number of clusters supported, which in turn affects a volume’s maximum size and storage efficiency. FAT32 breaks the 4-GB volume limitation of FAT16 by extending the maximum number of clusters, and therefore works well on large disks with complex folder structures.

Due to the greater number of available clusters within FAT32, each cluster can be made smaller for a particular volume, increasing the efficiency of data storage. For example, FAT16 volumes between 2 and 4 GB use a 64-KB cluster, whereas FAT32 volumes between 16 GB and 32 GB use a 16-KB cluster.

The largest possible file for a FAT32 volume is 4 GB minus 1 byte. FAT32 contains 4 bytes per cluster in the file allocation table; FAT16 contains 2 bytes per cluster; and FAT12 contains 1.5 bytes per cluster. A FAT32 volume must have at least 65,527 clusters.

Formatting Volumes

During volume format, Windows Server 2003 places key file system structures on the volume, including the boot sector as well as replacing Ntldr. A check of the integrity of all sectors on the volume is performed, and you have the opportunity to change the cluster size used on the volume. If a volume is formatted using Quick format, the file system structure on the volume is created, but the integrity of every sector in the volume is not checked.

The version of FAT that Setup uses depends on the size of the volume. For volumes smaller than 2 GB (2048 MB), Setup uses FAT16. For volumes 2 GB or larger, Setup uses FAT32. For volumes 32 GB or larger, Setup uses NTFS and does not offer FAT.

Disk Management requires volumes on dynamic disks and GPT disks to be formatted as NTFS.

Converting Volumes

Windows Server 2003 can convert FAT16 or FAT32 to NTFS. This type of conversion is a one-way process. After you convert a volume to NTFS, you cannot reconvert the volume to FAT without backing up your data, reformatting the volume as FAT, and then restoring your data.

During the formatting process, Windows Server 2003 aligns FAT data clusters at the cluster size boundary. Convert.exe can preserve the cluster size for the size of the volume (up to 4 KB) instead of using the 512-byte cluster size used in Windows 2000 for converted volumes. The following table lists cluster sizes for volumes converted to NTFS.

Cluster Sizes for Volumes Converted to NTFS

Original FAT Cluster Size	Converted NTFS Cluster Size

512 bytes	512 bytes
1 KB	1 KB
2 KB	2 KB
4 KB and larger	4 KB

Lack of Support for Compression

Unlike NTFS, FAT does not support compression of files. If you move or copy a file from a folder on a FAT volume to a folder on an NTFS volume, the file takes on the attributes of the folder on the NTFS volume. Compressed files copied or moved from an NTFS volume to a FAT volume automatically uncompress, because FAT does not support compression.

How Cluster Fragmentation Affects Performance

Lack of disk space can become an issue on a volume, especially when multiple users access the volume to create, open, modify, or delete files.

Fragmentation occurs when the virtual clusters of a file are on non-contiguous logical clusters. Because the file is spread out across the disk, access is slower, so fragmentation affects the performance of the file system. As users create new files or change the size of existing ones, the file system uses additional free clusters. If the file system cannot use contiguous clusters, it will find any available cluster on the disk. The closer that clusters for each file are located on the hard disk, the more efficiently the disk can retrieve the information. Disk defragmentation is a process that groups all the virtual clusters of a file into contiguous logical clusters, and is usually done in order to improved disk drive performance.

File Naming

Windows Server 2003 supports both long and short file names on FAT volumes.

File Names on FAT Volumes

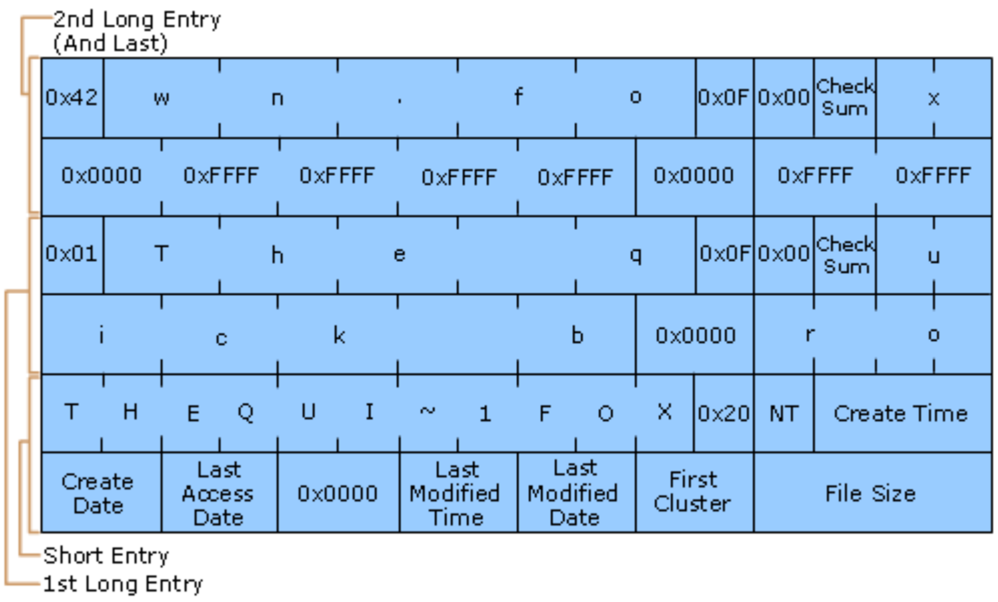
Files created or renamed on FAT volumes use attribute bits to support long file names in a way that does not interfere with how MS-DOS gains access to the volume.

When you create a file that has a long file name, Windows Server 2003 creates a conventional 8.3 name for the file and one or more secondary folder entries for the file, one for each set of 13 characters in the long file name. Each secondary folder entry stores a corresponding part of the long file name in Unicode. MS-DOS accesses the file by using the conventional 8.3 file name contained in the folder entry for the file.

Windows Server 2003 marks the secondary folder entries as part of a long file name by setting the volume ID, read-only, system, and hidden attribute bits. MS-DOS typically ignores folder entries with all these attribute bits set.

The following figure shows all of the folder entries for the file Thequi~1.fox, which has a long name, “The quick brown.fox.” The long name is in Unicode and each character in the name uses 2 bytes in the folder entry. The attribute field for the long-name entries has the value 0x0F. The attribute field for the short name has the value 0x20.

Long File Name on a FAT Volume



Note

- Windows NT and Windows Server 2003 do not use the same algorithm to create long and short file names as Windows 95, Windows 98, and Windows Me. However, on computers that use a multiple-boot configuration to start these operating systems, files that you create while running one operating system can be accessed while running another.

By default, Windows Server 2003 supports long file names on FAT volumes. You can disable long file names on FAT volumes if you use MS-DOS–based disk tools regularly on the computer. These tools might either eliminate the long file names created by Windows Server 2003 or delete the files that have long file names.

Note

- It is advised that you do not disable long file names on FAT volumes.

How Windows Server 2003 Creates File Names

File names in Windows Server 2003 can be up to 255 characters and can contain spaces, multiple periods, and special characters that are not allowed in MS-DOS file names. Windows Server 2003 makes it possible for other operating systems to access files that have long names by generating an MS-DOS-readable (8.3) name for each file. These MS-DOS-readable names also enable MS-DOS-based and Windows 3.*x*–based applications to recognize and load files that have long file names. When a program saves a file on a computer running Windows Server 2003, both the 8.3 file name and long file name are retained.

Note

- The 8.3 format means that files can have between 1 and 8 characters in the file name. The name must start with a letter or a number and can contain any characters except the following:
- . " / \ [ ] : ; | = , * ? (space)
- An 8.3 file name typically has a file name extension that is from one to three characters long and has the same character restrictions. A period separates the file name from the file name extension.

- Several special file names are reserved by the system and cannot be used for files or folders: CON, AUX, COM1, COM2, COM3, COM4, LPT1, LPT2, LPT3, PRN, and NUL.

When you create a file that has a long file name, Windows Server 2003 creates a conventional 8.3 name for the file and one or more secondary folder entries for the file, one for each set of 13 characters in the long file name. Each secondary folder entry stores a corresponding part of the long file name in Unicode. MS-DOS accesses the file by using the conventional 8.3 file name contained in the folder entry for the file.

Windows Server 2003 marks the secondary folder entries as part of a long file name by setting the volume ID, read-only, system, and hidden attribute bits. MS-DOS typically ignores folder entries with all these attribute bits set.

Generating Short File Names

In Windows Server 2003, FAT uses the Unicode character set, which contains several prohibited characters that MS-DOS cannot read, for their names. To generate a short MS-DOS-readable file name, Windows Server 2003 deletes all of these characters from the long file name and removes any spaces, but ensures that the generated short file name is unique. Because an MS-DOS-readable file name can have only one period, Windows Server 2003 also removes extra periods from the file name.

Related Information

- Basic Disks and Volumes Technical Reference
- NTFS Technical Reference

Community Additions