File Allocation Table

How It Seems To Work

Thomas Kjoernes,
Thu, 11th of May 2000

Introduction

In this article I will talk about FAT, the MS-DOS file system supported by most of today's OSes.

I will start out by explaining a little bit about the various flavours of FAT as used by MS-DOS, then I'll move on to explaining the various parts that make up the FAT file system and describe the structures

Some FAT source code and information:

OS.ZIP - This is my OS package! It contains three boot sector sources.

FAT32.HTML - This is an excerpt from the FAT32API.HLP file from Microsoft.

FAT12.ASM - Example of a FAT12 boot sector.

FAT16.ASM - Example of a FAT16 boot sector.

FAT32.ASM - Example of a FAT32 boot sector.

I will try to update this document with sample code and descriptions of how to interpret the FAT and perform common operations, such as searching for and reading/writing files.

FAT Types

Today FAT comes in three different flavours – FAT12, FAT16 and FAT32. The names refer to the number of bits used by the entries in table that gave the file system its name!

The "File Allocation Table" itself is actually one of the structures inside the FAT file system as seen on-disk. The purpose of this table is to keep track of which areas of the disk are available and which areas are in use.

Another important part about FAT is the "Long File Name" extension to FAT sometimes referred to as VFAT. The terms LFN and VFAT are closely related, but VFAT really means just Virtual FAT.

FAT Overview

It's time to get slightly technical. I'll first just mention all the structures almost in the order in which they usually appear inside the partition. When talking about the order of things I'm referring to order as seen through the Logical Block Address of a particular structure.

Cluster

This term is very fundamental for FAT. A cluster is a group of sectors on the FAT media. Only the part of the partition called the "data area" is divided into clusters. The rest of the partition is simply sectors.

Files and directories store their data in these clusters. The size of one cluster is specified in a structure called the Boot Record and can range from a single sector to 128 sector(s).

Boot Record

All the three flavours of FAT have a Boot Record, which is located within an area of reserved sectors. The DOS format program reserves 1 sector for FAT12 and FAT16 and usually 32 sectors for FAT32.

- *FAT overview from MS:*

  Microsoft has recently made a document available that attempts to teach us about FAT. The document is good as a reference if you already know something about FAT, but it's not very good at describing how to actually use the information. The document can be found at - http://www.microsoft.com/hwdev.

  I will not attempt to duplicate all the technical documentation presented in that document. I suggest you get it and read if you a very detailed explanation of every single field in the Boot Record for instance.

File Allocate Table

The actual "File Allocation Table" structure is a relatively simple structure, as are all of the FAT structures really. The FAT is a simple array of 12-bit, 16-bit or 32-bit data elements. Usually there will be two identical copies of the FAT.

There is a field in the Boot Record that specifies the number of FAT copies. With FAT12 and FAT16, MS-DOS uses only the first copy, but the other copies are kept in sync. FAT32 was enhanced to specify which FAT copy is the active one in a 4-bit value part of a "Flags" field.

It's quite common to think of the FAT as a singly linked list. Each of the chains in the FAT specify

which parts of the disk belong to a given file or directory.

Root Directory

The Root Directory is formatted like any other directory except it does not contain the "dot" and "dot-dot" entries. See the details section for more information. The root directory can always be found immediately following the file allocation table(s) for FAT12 and FAT16 volumes.

Data Area

Time has come to describe the user data area. What is there to say really? The user data area (or just data area if you like) is where the contents of files and directories are stored. Simple as that…

See the formulas above for how to calculate the size of the data area. And yes, the data area is divided into sector groups called clusters. All the clusters in a single FAT volume have the same size.

To further educate you, the term slack space refers to any unused space at the end of a cluster and cannot be used by any other file or directory.

Note that directories are not known to suffer from slack space problems. This is simply because the exact size in bytes of a directory is not recorded as with files and generally no one seem to care anyway.

The data area section will not be explained in detail. There is simply nothing more to say about it. Information on how to access files and directories is the closest we get to data area details.

Wasted Sectors

If the number of data sectors is not evenly divisible by the cluster size you end up with a few wasted data sectors. Also if the partition as declared in the partition table is larger than what is claimed in the Boot Record the volume can be said to have wasted sectors.

If you are not familiar with the term partition table, I suggest that you go to Hale Landis' web site and look for the How It Works series of documents at - http://www.ata-atapi.com.

Boot Record Details

The Boot Record is located at the very beginning of a FAT volume. FAT12 and FAT16 boot sectors occupy a single sector while FAT32 boot sectors are generally said to consist of three sectors.

The first sector of the volume or the first few sectors are also known as the "reserved" sectors or the reserved area. The Reserved_Sectors field in the boot record tells us how large this area is. Note

that the first FAT follows directly after the reserved area.

As mentioned in the overview section, the DOS format program reserves 1 sector for FAT12 and FAT16 and usually 32 sectors for FAT32. The reserved area for FAT32 contains not only the boot record but also a backup copy of the boot record.

The boot record includes a field describing the sector size for the media and it is possible to have a boot record size different than the 512-byte commonly seen on harddisks and diskettes (usually seen on RAM-disks).

Inside the boot record there is a structure called the BPB or BIOS Parameter Block. This structure is what really makes up the boot record. To further complicate things there are different versions of the boot record as well. Basically, as storage technology developed and disks got bigger, a few new fields were added to the boot record to support larger disk sizes.

Of course being a lazy person, I will completely ignore the older boot record layouts and concentrate on the most current implementation. The structure definitions that follows come from one of the include files used by my OS – FAT.INC.

I only attempt to show you the layout of the BPB and remaining boot sector structures. If you want details of the purpose of each field and what they may and may not contain I urge you to read the Microsoft FAT overview document, which describes this quite well.

Note that the BPB and Boot Sector layouts for FAT12 and FAT16 are identical. For this reason a show only the FAT12 and FAT32 structures.

```
; BPB for FAT12 and FAT16 volumes

BPB_FAT12                       STRUC           (offset)
bpbBytesPerSector               DW ?              0x0B
bpbSectorsPerCluster            DB ?              0x0D
bpbReservedSectors              DW ?              0x0E
bpbNumberOfFATs                 DB ?              0x10
bpbRootEntries                  DW ?              0x11
bpbTotalSectors                 DW ?              0x13
bpbMedia                        DB ?              0x15
bpbSectorsPerFAT                DW ?              0x16
bpbSectorsPerTrack              DW ?              0x18
bpbHeadsPerCylinder             DW ?              0x1A
bpbHiddenSectors                DD ?              0x1C
bpbTotalSectorsBig              DD ?              0x20
BPB_FAT12                       ENDS
```

```
; BPB for FAT32 volumes

BPB_FAT32               STRUC           (offset)
bpbBytesPerSector       DW ?              0x0B
bpbSectorsPerCluster    DB ?              0x0D
bpbReservedSectors      DW ?              0x0E
bpbNumberOfFATs         DB ?              0x10
bpbRootEntries          DW ?              0x11
bpbTotalSectors         DW ?              0x13
bpbMedia                DB ?              0x15
bpbSectorsPerFAT        DW ?              0x16
bpbSectorsPerTrack      DW ?              0x18
bpbHeadsPerCylinder     DW ?              0x1A
bpbHiddenSectors        DD ?              0x1C
bpbTotalSectorsBig      DD ?              0x20
bpb32SectorsPerFAT      DD ?              0x24
bpb32Flags              DW ?              0x28
bpb32Version            DW ?              0x2A
bpb32RootCluster        DD ?              0x2C
bpb32InfoSector         DW ?              0x30
bpb32BootBackupStart    DW ?              0x32
bpb32Reserved           DB 12 DUP (?)     0x34
BPB_FAT32               ENDS

; Boot Sector layout for FAT12 and FAT16

BS_FAT12                STRUC                       (offset)
bsJmp                   DB 3 DUP (?)                  0x00
bsOemName               DB 8 DUP (?)                  0x03
bsFAT12                 BPB_FAT12 Structure           0x0B
bsDriveNumber           DB ?                          0x24
bsUnused                DB ?                          0x25
bsExtBootSignature      DB ?                          0x26
bsSerialNumber          DD ?                          0x27
bsVolumeLabel           DB "NO NAME    "              0x2B
bsFileSystem            DB "FAT12   "                 0x36
bsBootCode              DB 450 DUP (?)                0x3E
BS_FAT12                ENDS

;Boot Sector layout for FAT32
```

```
BS_FAT32                STRUC                   (offset)
bsJmp                   DB 3 DUP (?)              0x00
bsOemName               DB 8 DUP (?)              0x03
bsFAT32                 BPB_FAT32 Structure       0x0B
bs32DriveNumber         DB ?                      0x40
bs32Unused              DB ?                      0x41
bs32ExtBootSignature    DB ?                      0x42
bs32SerialNumber        DD ?                      0x43
bs32VolumeLabel         DB "NO NAME    "          0x47
bs32FileSystem          DB "FAT32    "            0x52
bs32BootCode            DB 422 DUP (?)            0x5A
BS_FAT32                ENDS
```

File Allocation Table Details

It's time to mention clusters again. The 0th and 1st FAT entries are reserved and contain some special information. The 2nd entry and up tells you the state of the corresponding cluster in the data area.

A value of zero indicates that the cluster represented by that FAT entry is available and can be used when allocating new space for a file or directory.

A hexadecimal

value in the range FFF8-FFFF indicates that the cluster is the last in that chain of clusters. Yes, as you might have guessed, this example is only valid for FAT16. Check out the table below for cluster values for all FAT types:

Cluster Values

	FAT12	**FAT16**	**FAT32**
Available	000	0000	00000000
Reserved	001	0001	00000001
User Data	002-FF6	0002-FFF6	00000002-0FFFFFF6
Bad Cluster	FF7	FFF7	0FFFFFF7
End Marker	FF8-FFF	FFF8-FFFF	0FFFFFF8-0FFFFFFF

If you have a closer look at the FAT32 values, you'll see that only they are actually 28-bits long. This is actually correct. The upper nibble is stated by Microsoft as being reserved and might have a special meaning in future FAT32 implementations.

They also tell you that you should mask the bits off when interpreting the FAT. Also, care should be taken to preserve the upper bits when changing the FAT.

- *If I remember correctly, the wonderful MS FAT overview document states that a format utility should initialise the reserved bits to zero. If the meaning of those bits are changed dramatically the FAT32 version number in the Boot Record will most likely also change. Currently the version number is zero.*

The MS document tells us that the FAT type is determined by how many clusters there is room for in the user data area. Have a look at the table below:

FAT Type Determination

	FAT12	**FAT16**	**FAT32**
# of clusters	n-4084	4085-65524	65525-n

As seen in the previous table, 11 values were reserved for other purposes, which tells us to simply subtract that number from the maximum number of values represented by a 12-bit, 16-bit and 28-bit number respectively (remember that the upper 4-bits are reserved in FAT32).

This does not match what the MS document says unfortunately. The MS numbers are one less that when just taking account for the reserved values. Since the document explicitly states that THIS INFORMATION IS CORRECT and the NUMBERS ARE CORRECT we must obey and use those numbers instead of the ones I really want to use.

If you take a look at another of "my" documents FAT32.HTML, which was actually written by Microsoft, they use "my" numbers. The actual statement is:

> *FAT12 Less than 4086 clusters*
> *FAT16 Between 4096 and 65526 clusters*
> *FAT32 Greater than 65526 clusters.*

The size of the data area is determined using the horrible pseudo-formulas shown below. I split the formula into smaller logical parts, simply because it looked horrible all in one go.

- *Root_Sectors = Root_Directory_Entries * 32 / Bytes_Per_Sector*

*FAT_Sectors = Number_Of_FATs * Sectors_Per_FAT*

Data_Sectors = Total_Sectors – (Reserved_Sectors + FAT_Sectors + Root_Sectors)

Total_Clusters = Data_Sectors / Sectors_Per_Cluster

To locate the start sector of the data area you can use the following formula:

- *Data_Area_Start = Reserved_Sectors + FAT_Sectors + Root_Sectors*

Root Directory Details

As stated in the overview section, the root directory is formatted just like any other directory. See the directory and LFN details section for information about the directory entry format.

FAT12 and FAT16 volumes have the root directory located immediately following the file allocation table(s). The following formula gives you the starting sector number for the root directory:

- *Root_Starting_Sector = Reserved_Sectors + FAT_Sectors*

*See the formula in the previous example for the **FAT_Sectors** details.*

On FAT32 volumes the root directory is made up of an ordinary cluster chain. A field in the Boot Record will tell you the initial cluster number. Once you've got the initial cluster number you can easily get the starting sector number for FAT32 as well:

- *Root_Starting_Sector = ((Root_Cluster – 2) * Sectors_Per_Cluser) + Data_Area_Start*

This is exactly the same way you would find the starting sector number for a file or a directory. The *Data_Area_Start* formula includes *Root_Sectors*, but that does not apply to this formula. The Boot Record for a FAT32 volume will have the field holding number of entries in the root directory set to zero anyway.

Directory and LFN details

I feel it is time to describe how directories are implemented in the FAT file system. All directories except the root directory for FAT12 and FAT16 drives are actually files. A file is a stream of bytes located in the data area portion of the volume made up by one or more clusters. The exact size of the file is recorded in the directory structure.

The size of directory files is not recorded anywhere. Directories are always treated as being a multiple of the cluster size. Directories will be expanded when a new entry is added and the

directory is already full.

Note that the root directory for FAT12 and FAT16 drives cannot be expanded. The size of the root directory for these FAT types was determined when the volume was formatted.

The directory is divided into small structures called Directory Entries. Each directory entry is always exactly 32 bytes long. The entry holds the name, attribute, size, date, time and initial cluster number for the file or directory.

The Long File Names supported with VFAT are stored in addition to the short/normal name in entries preceding the short name. The LFN entry is hidden from older utility software (basically older MS-DOS versions) by using a traditionally illegal attribute value.

Long names are stored in UNICODE, which is the 16-bit successor of ASCII. Each UNICODE character normally appears as an ASCII character followed by a null byte.

Each LFN entry can hold up to 13 UNCODE characters. If the long name is longer than that additional entries are stored in the directory. Also note that the LFN entries are stored in reverse order with the first part of the long name appearing just before the short name entry.

Normal/Short Entry Format

8 BYTEs	Blank-padded name - The first byte tells you some special information: 00h – entry is available and no entry beyond this one has been used. 05h – first character is actually E5h 2Eh – first character is a dot; this is a special entry. It can either be the "dot" or "dot-dot" entry, which is present in all directories except the root directory. The "dot" entry has a cluster number that points to the directory itself. The "dot-dot" entry has a cluster number that points to the parent directory (or null if the parent is the root directory)/ E5h – entry has been erased and is available.
3 BYTEs	Blank-padded extension

1 BYTE	Attribute - This field is bit-mapped. Only bits 0-4 should be used, bits 5-7 are said to be reserved. Note that the special value of 0Fh indicates an LFN entry. The value 0Fh is an otherwise illegal attribute value. - Older software might think that LFN entries are "Read-Only System Hidden Volume Directories", even a fool will realise that means "CAN'T TOUCH THIS, do, do-do-do, da, do-do, da-da"… 00001b – Read-Only 00010b – System 00100b – Hidden 01000b – Volume 10000b – Directory
1 BYTE	Reserved, used by Windows NT, Novell DELWATCH apparently uses this to store the original first character for erased entries.
1 BYTE	10-ms unit's "Create Time" refinement (added with VFAT).
1 WORD	Creation time (added with VFAT).
1 WORD	Creation date (added with VFAT).
1 WORD	Access date (added with VFAT).
1 WORD	High 16-bits of Cluster # (added with and used for FAT32).
1 WORD	Update time (set on creation as well)
1 WORD	Update date (set on creation as well)
1 WORD	16-bit Cluster #
1 DWORD	File size in bytes (always zero for directories).

The time and date fields use the following "packed" storage format:

Packed Time:

15:11	10:5	4:0
Hour	Minute	Second / 2

Packed Date:

15:9	8:5	4:0
Year – 1980	Month	Day

I haven't seen any official MS documentation on LFN entries, but after studying directory sectors and reading lots about it from various sources, including Ralf Brown's Interrupt List, which you BTW, should get from this location – <insert WWW page here>, I've come to the conclusion that this information is very accurate.

Also note that it's merely a guess that the new file access/creation time/date fields added with VFAT are stored in the same format as the standard time/date fields.

Long File Name Entry Format

1 BYTE	LFN Record Sequence Number - Bits 5:0 hold a 6-bit LFN sequence number (1..63). Note that this number is one-based. This limits the number of LFN entries per long name to 63 entries or 63 * 13 = 819 characters per name. - The longest filename I was able to create using Windows 95 Explorer was 250 characters. I managed to use a 251-character name when saving a file in Microsoft Word. I don't know if this is the limitation of the FS driver or if it is limited at the application level. - Bit 6 is set for the last LFN record in the sequence. - Bit 7 is set if the LFN record is an erased long name entry or maybe if it is part of an erased long name?
10 BYTEs	5 UNICODE characters, LFN first part.

1 BYTE	Attribute - This field contains the special value of 0Fh, which indicates an LFN entry.
1 BYTE	Reserved (probably just set to zero).
1 BYTE	Checksum of short name entry, used to validate that the LFN entry belongs to the short name entry following. According to Ralf Brown's interrupt list, the checksum is computed by adding up all the short name characters and rotating the intermediate value right by one bit position before adding each character.
12 BYTEs	6 UNICODE characters, LFN second part.
1 WORD	Initial cluster number, which is always zero for LFN entries.
4 BYTEs	2 UNICODE characters, LFN third part.

The Long File Name entries include every single character in the long name including the "dot" which is implied by short names.

The very last LFN entry will have a null-terminator if less than 13 characters are stored in that long name entry. If only 12 characters or less are stored the remainder of the long name parts are filled with FFFFs.

- TK -