FAT File System

FAT. DOS File Allocation Table that is. The FAT is the first sectors on a disk that tell the operating system where to find a file, how much of it there is and where all the pieces are. It also marks the sectors as used, full or bad so the system can determine where to place a file. Ok...Sounds easy. This information can be found all over the internet. But have you ever tried to figure it out? Well I did, and after a lot of looking and reading and playing with disk, this is what I've learned...

The DOS disk is laid out in tracks, also called cylinders. Each track has sectors. And there are two sides, so 2 heads. That is where simple ends.

A disk can have any number of tracks, but most 1.44 floppies have 80. This is pretty easy but remember tracks are also called cylinders. Track 0 is on the outer edge of the disk with track 79 on the inter edge. All tracks have the same number of sectors even though the outer tracks have more room.

A track can have any number of sectors, but most 1.44 floppies have 18. This is set by a value in the Bios Parameter Block, BPB, which is in first of the boot sector.

A disk can also have clusters. The value can also be determined from the BPB. Clusters are a group of sectors. On most 1.44 floppies there are 1 sector per cluster.

Now...With all this, are you confused? Well, just hold on, it gets better.

Absolute Sector

Absolute sector is addressing the disk by head, track and sector. It is the way the controller and the BIOS access the disk. It is also the way to install you own bootstrap code or manually work on the FAT.So where is the FAT? Let's take a look.

```
HEAD 0    TRACK 0    SECTOR 1
```

This is the boot code sector. It is also where the BPB is located.

```
Program start.
Jump over data        ;Three bytes off start.
NOP                   ;If short jump, must have NOP to make 3 bytes
OEM_ID            db 8 bytes
BytesPerSector    dw 0x0200        ;512 bytes per sector
SectorsPerCluster db 0x01          ;1 sector per cluster
ReservedSectors   dw 0x0001        ;Reserved sectors.. ie boot sector
TotalFats         db 0x02          ;2 copies of the FAT
MaxRootEntries    dw 0x0E0         ;Number of entries in the root. 224
TotalSectors      dw 0x0B40        ;Number of sectors in volume 2880
MediaDescriptor   db 0xF0          ; 1.44 floppy
SectorsPerFat     dw 0x0009        ;Number of sectors in a FAT 9
SectorsPerTrack   dw 0x0012        ;Sectors per Track 18
NumHeads          dw 0x0002        ;2 heads
HiddenSectors     dd 0x00000000
TotalSectorsLarge dd 0x00000000
DriveNumber       db 0x00
Flags             db 0x00
Signature        db 0x29
VolumeID          dd 0xFFFFFFFF
VolumeLabel       db "XXXXXXXXXXX" ;12 bytes 8+"."+3
```

```
SystemID          db "FAT12   "     ;8 bytes

Code starts here.
```

At byte 510 is stored the word 0xAA55. This is what the BIOS looks for to see if the disk is a boot disk.

```
HEAD 0    TRACK 0    SECTOR 2
```

This is the start of the FAT. To defy logic, instead of going head 0 then head 1, DOS sticks with head 0 till sector 18. This can really confuse you at first. Anyway. From looking at the BPB we see that the FAT is 9 sectors long and there are two copies. So Starting here the FAT track table goes all the way to HEAD 0 TRACK 0 SECTOR 18

```
HEAD 1    TRACK 0    SECTOR 1
```

Because the boot sector took one sector, one sector of the second FAT wraps to head 1. So head 1, track 0 and sector 1 is the last sector of the FAT sector table.

```
HEAD 1    TRACK 0    SECTOR 2
```

This is the first sector of the FAT directory. This is where programs look to see what's on the disk. Each entry is 32 bytes long. The name of the file, date/time stamp, attribute, start sector and length are located here. 32 bytes times the number of entries, 224, gives 7168 bytes. Divide that by the the number of bytes per sector, 512, you get 14 sectors. So the directory table is 14 sectors long.

```
HEAD 1    TRACK 0    SECTOR 16
```

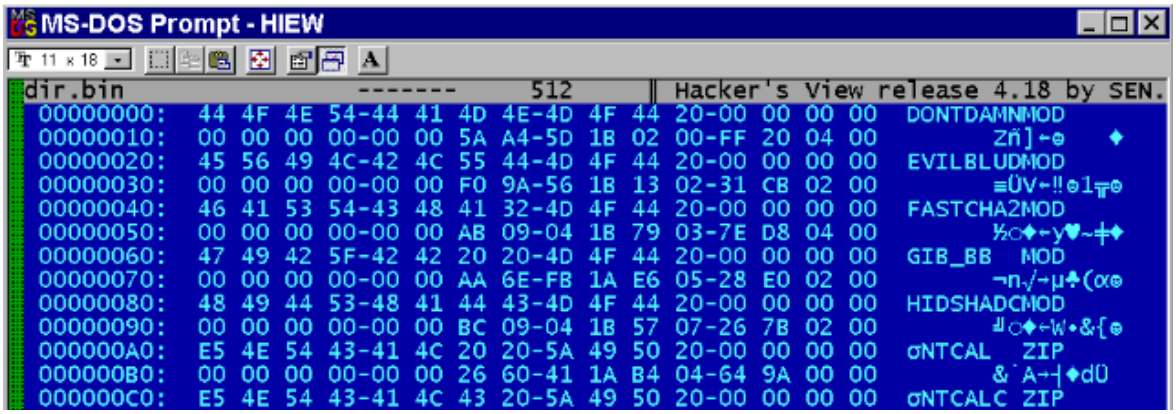
This is the first sector of the data area. It is here that the actual files are kept.

So. Now that we know where we are at, you might be asking how do you find the file. Well..Now for the really fun, fun stuff.

Logical Sector Addressing

The location of the file is marked in the FAT in logical sector addressing. Instead of saying the file is at head 0, track 0, sector 1, it is at sector 0. This does make it easier to store the location, but requires a couple of math formulas to get back to absolute sector.

If you take a look at a directory table you can see how this works.



The first 11 bytes are the file name: 0x44 0x4F 0x4E 0x54 0x44 0x41 0x4D 0x4E 0x4D 0x4F 0x44 Dontdamn mod DOS adds the period.

The next 1 byte is the attribute: 0x20 100000 Binary which stands for Archive Others are:

```
0x01    000001    Read only
0x02    000010    Hidden
0x04    000100    System file
0x08    001000    Volume ID
0x10    010000    Directory
```

The next 8 bytes are reserved:

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

The next 2 bytes are index in EA Data:

```
0x00 0x00
```

The next 4 bytes are the time/date stamp, encoded:

```
0x5A 0xA4 0x5D 0x18
```

The next 2 bytes are the entry cluster in the file chain:

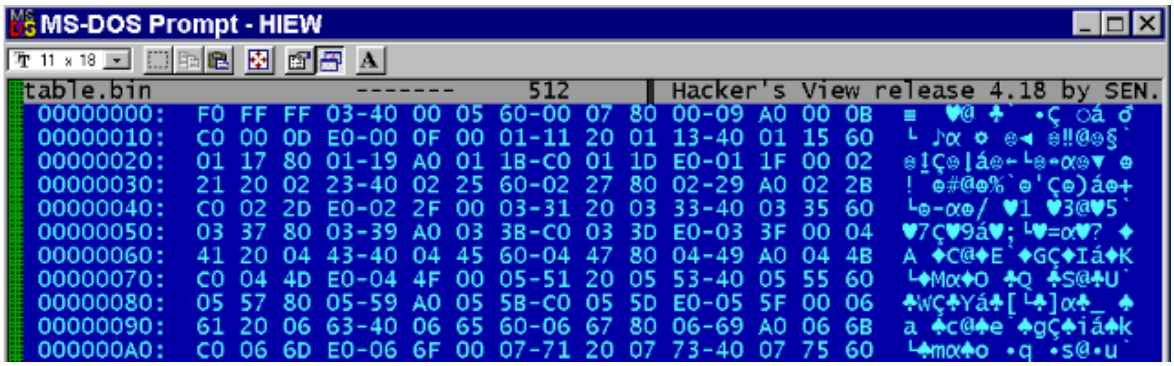
```
0x02 0x00
```

The next 4 bytes are the file size in bytes:

```
0xFF 0x20 0x04 0x00
```

If you look at bytes 26 and 27,(remember to start counting from 0) 0x02 and 0x00 in this example. This is the starting cluster for this file. Because of the way DOS stores numbers, this is actually 0x00 0x02 or 2. So this file starts at cluster 2. More on how to find where this is in a minute.

First, let's look at the sector table at head 0, track 0 and sector 2.



Take your starting number, 2 and multiply it by 1.5. This gives you three. Now look at the sector table and get the third and fourth byte. 0x03 0x40 Reverse their order to get 0x4003. Because our starting number, 3, was a whole number we AND the hexadecimal number with 0x0FFF to get 3.

Multiply 3 by 1.5 to get 4.5. So go to the table and get the 4th and 5th bytes. 0x40 0x00 Reverse them 0x00 0x40 to get 0x40. Because 4.5 was not a whole number we shift 0x40 to the right to get 0x04. Or 4 decimal.

Multiply 4 by 1.5 to get 6. So go to the table and get the 6th and 7th bytes. 0x05 0x60 Reverse them 0x60 0x05 to get 0x6005. Because 6 is a whole number we AND 0x6005 and 0x0FFF to get 5.

Continue until you get the bytes 0xFF 0x0F. When they are reversed they form 0x0FFF which marks the end sector of the file.

To find the logical sectors, take the numbers and subtract 2: 3-2=1, 4-2=2, 5-2=3 and so on.

Multiply that number by the number of bytes in a sector. 1*0x200=0x200 2*0x200=0x400 3*0x200=600 and so on.

Now add that number to the offset of the FAT. On this disk:

```
1 reserved                          0x0200
2 FATs at 9 sectors each. (9*2)*512  0x2400
```

```
1 Directory 224 entries of 32 byte   0x1C00
Total                                0x4200
```

So add 0x200 to 0x4200 to get 0x4400. Now divide this by bytes per sector (512 or 0x200) to get logical sector 34
0x4200 + 0x400 = 0x4600 / 0x200 = 0x23 sector 35
0x4200 + 0x600 = 0x4800 / 0x200 = 0x24 sector 36
and so on till the end.

Now that we know the logical sector, you might be wondering how to get it to absolute sector so we can read it. Well we use a few formulas to figure it out. They are:

```
sector = ( logical sector MOD sector per track) + 1
head = (logical sector \ sector per track ) MOD number of heads
track = logical sector \ (sector per track * number of heads)
```

Note that this is integer division instead of floating point division. And also Modulo math. If you can't figure this out just turn on QBasic and enter a quick formula. So. Using the formulas above we get:

```
sector= (34 MOD 512)+1=        17
head=   (34 \ 512) MOD 2=       1
track=   34 \ (512 *2)=         0
Our file starts at Head 1 Track 0 Sector 17
The next sector    Head 1 Track 0 Sector 18
The next sector    Head 0 Track 1 Sector 1
and so on till the end.
```

There you go. The skinny on the FAT. I hope that this helps someone. I know I did a lot of searching and playing with disk to figure this out. Most places tell you how it works but not where it is located on the disk.

A note for the OS programer. DOS does not read the second FAT. I have placed a 512 byte banner at the last sector of the second fat and called it on boot.

Back