

Introduction to: Partition Tables

Copyright © 2004,2005 by Daniel B. Sedory
NOT to be reproduced in any form without Permission of the Author!

[Introduction to Partition Tables.](#)

- [Example: A 20 GB Hard Drive \(with NTFS, FAT32 and Linux Partitions\)](#)

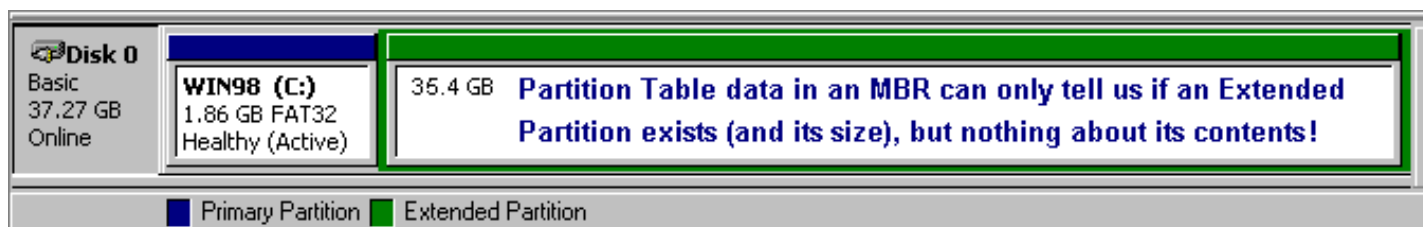
Introduction

The **Master Boot Record** (the very first usable sector) of each hard disk contains that drive's *Master* (and quite often only) **Partition Table**. The Partition Table makes up *only* 12.5% (**64** out of **512** bytes) of the **MBR** sector and it's located only two bytes from the end of the MBR; those last two bytes are called the MBR's Signature Word.

The **Standard** Partition Table is divided into **four 16-byte entries** (there were also some deviations from this around the time of DOS 3.30; such as the [NEC Powermate DOS 3.3](#) which had a special *eight-entry* table). Due to the fact that this table has become the default *worldwide* standard for almost every PC on the planet, you can **not** have more than **four Primary Partitions** per hard disk. However, a long time ago, IBM®/Microsoft® came up with a scheme to **extend** Partition Table entries into other parts of the disk! These are commonly known as **Extended Partition** entries, and they have Partition Types (more on *types* later) of **05** and **0F** hex.

If a hard disk's **MBR** has an **Extended Partition** entry in its Partition Table (it can have only **one**), it's *very important* that you **understand this**: The MBR entry only describes what can be thought of as an *envelope* (or a **black box**) which *might* contain *up to 23 Logical drives* under IBM®/Microsoft® DOS 5.0 or later operating systems. The point here is this: Until you look **inside** the first **Extended Partition Table** **of** that **Extended Partition**, it's impossible for you to know what's in there!

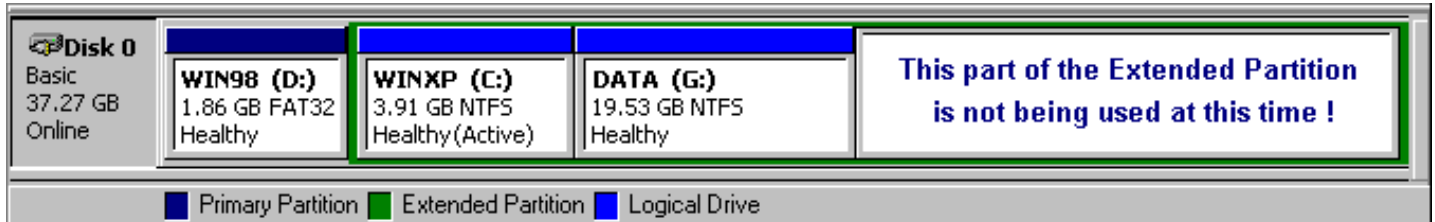
The following diagram shows a typical 40 GB HDD with a small FAT32 partition at the beginning of the disk. The remainder of the disk is enclosed within an Extended partition:



The only way to figure out just how many **Logical Drives** there are *within* an **Extended**

Partition is by *jumping to each Extended* Partition Table in the **Extended Boot Records** until you come to the last **EBR** Table. These EBRs are often described as being *chained* together by each **link** to the next EBR Table from its previous link. Therefore, if you want a complete picture of a hard disk that contains an Extended Partition, you need to copy each of the **Extended** Partition Tables (in their own **EBRs**) as well as the **Master Boot Record** sector!

This next diagram shows only one of many possible ways the Extended Partition shown above might be divided:



We consider it just as important to copy and store the data contained in any **EBR** sectors as you would for the **MBR** and to store these copies elsewhere (and/or write out the data). If the OS can no longer read any EBR sector's contents, you'll have a better chance of fixing problems (or recovering your data) with the contents of these tables! You can use PQ's [PartInfo](#) program (which lists all the data from each of the **daisy-chained** Extended Partition Tables; as well as the **MBR**) to save it as a single **text file** on a floppy diskette; along with a **binary** copy of your **MBR** sector.

In summary, the layout of a **Master Boot Record's** sector is as follows:

Offset	Size	Description
000h	446 bytes	Master bootstrap loader code area (includes the NT Drive S/N if any; see the Standard , FAT32 and NTFS MBRs for example.)
1BEh	16 bytes	Partition entry for partition 1
1CEh	16 bytes	Partition entry for partition 2
1DEh	16 bytes	Partition entry for partition 3
1EEh	16 bytes	Partition entry for partition 4
1FEh	2 bytes	Signature ID , the WORD: AA55h ; indicates a valid MBR sector.

Example: 20GB Hard Drive with Three OSs

Hexadecimal View of the MBR Sector's Partition Table

01B0:	00 00 00 00 00 00 00 00	A8 E1 A8 E1 00 00	80 01
01C0:	01 00 07 FE FF 6D	3F 00 00 00 AF 39 D7 00	00 00m?....9....
01D0:	C1 6E 0C FE FF FF	EE 39 D7 00 BD 86 BB 00	00 FE	.n.....9.....
01E0:	FF FF 83 FE FF FF	AB C0 92 01 CD 2F 03 00	00 FE/.....
01F0:	FF FF 0F FE FF FF	78 F0 95 01 83 AF CC 00	55 AAx.....U.

This is a Hex view of just the last **80 bytes** of a hard disk's MBR sector as they might appear in a disk editor. The **Partition Table** is enclosed within the thin **YELLOW** line in the *diagram above*. Two other items also appear in this diagram: **1)** The thin violet/purple line surrounding the Hex bytes "A8 E1 A8 E1" shows an example of an NT *Disk Signature* (also used by [Win 2000/XP](#)) and: **2)** The sector's **Word-sized Signature ID** of **AA55h**; often called its **Magic number** in older tech documentation. **[Note: on Intel x86 CPU systems, Hex Words are stored with the Low-byte first and the High-byte last so AA55h appears as: **55 AA** in a disk editor].**

In the *upper-right corner* of the table, you'll see a byte with the value of **80**. This is the first byte *in* the first Partition entry, and it's followed by the bytes **01 01 00**. Most HDDs have these bytes, since the first partition is usually the **boot** partition on most hard disks [the **80h** byte tells the Bootstrap Loader code (the MBR code) which partition it is supposed to try loading a Boot sector from into Memory; only one partition entry (at boot-up) can have the 80h byte here] *and* because the next three bytes are the **CHS** values for the Starting Sector of the first partition (CHS **0,1,1** in this case; which corresponds to Absolute Sector 63 of this disk, where most Boot Records begin).

Until HDDs became larger than 8.4GB, the Starting and Ending CHS values and the Total Sector value could be checked against each other; which might help to correct an error for a manual entry in the Table. Example: The Last Sector **CHS** values (**877,254,63**) can be used to check the Total number of Sectors by first computing: **878** cylinders x **255** heads x **63** sectors = **14,105,070** sectors (don't forget that both the **Cylinder and Head counts begin with a zero**), and then subtracting the **63** sectors which preceded it to arrive at: **14,105,007** sectors for its **Total** size or length. Thus, you can see that they must **agree** with each other!

Before launching into all the calculations for the rest of this drive's partition table, we need to point out that this isn't the typical layout for a drive created by a Microsoft OS. Instead of a **single Primary** partition followed by an **Extended** partition, this drive has **three Primary** partitions and **then** an **Extended** partition!

The First 16-byte Entry in our Partition Table

Byte(s) Offset	Value in this Example	Description	Meaning
1BE	80	Bootable? (80h = Yes; 00 = No)	YES
1BF - 1C1	01 01 00	Starting Sector (in CHS) [First byte <i>always</i> = Head value]	0, 1, 1

1C2	07	Partition Type	NTFS
1C3 - 1C5	FE FF 6D	Last Sector (in CHS) [FEh = 254 for Head] FF 6D = (1111 1111) (0110 1101) <i>are regrouped as:</i> [11 1111] [11 0110 1101] 3Fh = 63 sectors and 36Dh = 877 (Cylinder)	877, 254, 63
1C6 - 1C9	3F 00 00 00	Relative Sectors (or Offset) (000000 3Fh = 63)	63
1CA - 1CD	AF 39 D7 00	Total Sectors (or Length) (00 D739AFh = 14,105,007)	14,105,007

In the table above, you'll find that computing CHS values isn't always very easy to do! Though the **whole first byte** for each Starting and Ending **CHS entry** is *always* for the **Head***, the Cylinder and Sector bytes must be **regrouped** before you can compute them! To compute the **Sector** value, you must *remove the leading two bits* of the second (or middle) byte; thus the largest value you can have for a Sector is: **3Fh** (11 1111) = **63**. To compute the Cylinder value, the **two bits** from the middle byte **become the first two bits** of a **ten-bit hex number** that **ends** with the **last byte** in each CHS entry. So, the largest value you can have for a Cylinder is: **3FFh** (11 1111 1111) = **1023** (in the table, which gives you a **count** of **1024** cylinders). Since these are the largest values that can be used in any Standard Partition Table, all CHS values are **limited to**: **1024** x **255*** x **63** = 16,450,560 sectors (or **8,422,686,720** bytes); thus the **8.4 GB limit** for any BIOS *without* the newer **INT 13 Extensions** (such as Functions 42, 43 and 48 for reading, writing and getting drive parameters)!

*For those who missed the note about why we do **not** use 256 heads, see this: [Note about a 'bug' in MS-DOS.](#)

Therefore, when **any part** of a partition reaches further than **8.4 GB** into a drive, you lose some redundancy as can be seen in the **Last Sector** CHS portion of this entry:

The Second 16-byte Entry in our Partition Table			
Byte(s) Offset	Value in this Example	Description	Meaning
1CE	00	Bootable? (80h = Yes; 00 = No)	NO
1CF - 1D1	00 C1 6E	Starting Sector (in CHS) [First byte: 00h = 0 for Head] C1 6E = (1100 0001) (0110 1110) in Binary <i>are regrouped as:</i> [00 0001] [11 0110 1110] 01h = 1 sector and 36Eh = 878 (Cylinder)	878, 0, 1

1D2	0C	Partition Type	FAT32 LBA
1D3 - 1D5	FE FF FF	Last Sector (in CHS) FF FF = (11 11 1111) (1111 1111) <i>are regrouped as:</i> [11 1111] [11 1111 1111] 3Fh = 63 sectors <i>and</i> 3FFh = 1023 (Cylinder)	Beyond 8.4 GB
1D6 - 1D9	EE 39 D7 00	Relative Sectors (or Offset) (00D739EEh = 14,105,070)	14,105,070
1DA - 1DD	BD 86 BB 00	Total Sectors (or Length) (00BB86BDh = 12,289,725)	12,289,725

If the Cylinder value wasn't limited to 1023, then the Last Sector **CHS** triple's cylinder value could be computed as follows: **26,394,795** (14,105,070 + 12,289,725; the same as the Relative Sector Offset in the next entry!) **divided by** 16,065 (255 x 63) = **1643** cylinders. So, the **CHS** would be: **1642,254,63**.

And for any full partition that's beyond 8.4 GB, such as the last two entries in our example Partition Table above, only the values inside of the thin **GREEN** lines can be used to compute both a partition's actual size **and location!** Note both the Starting Sector and Last Sector items in our example Table's third entry:

The Third 16-byte Entry in our Partition Table			
Byte(s) Offset	Value in this Example	Description	Meaning
1DE	00	Bootable? (80h = Yes; 00 = No)	NO
1DF - 1E1	FE FF FF	Starting Sector (in CHS) FF FF = (11 11 1111) (1111 1111) <i>are regrouped as:</i> [11 1111] [11 1111 1111] 3Fh = 63 sectors <i>and</i> 3FFh = 1023 (Cylinder)	Beyond 8.4 GB
1E2	83	Partition Type	Linux Ext2
1E3 - 1E5	FE FF FF	Last Sector (in CHS) [Same as for FE FF FF above!]	Beyond 8.4 GB
1E6 - 1E9	AB C0 92 01	Relative Sectors (or Offset) (0192C0ABh = 26,394,795)	26,394,795
1EA - 1ED	CD 2F 03 00	Total Sectors (or Length) (00032FCDh = 208,845)	208,845

Finally, the fourth entry in our example Table is that of an LBA Extended Partition (**type**

= **0Fh**). In the case of any Extended Partition, its Total Sectors (or Length) includes all of the Logical partitions that may exist inside of it, whether there's only one or many. Extended Partitions often run all the way to the end of a drive's useable sectors, but they don't have to. You should also **note** that even if all four entries of a Partition Table are being used, that does **not** necessarily mean there's no empty space left on a drive.

The Fourth 16-byte Entry in our Partition Table			
Byte(s) Offset	Value in this Example	Description	Meaning
1EE	00	Bootable? (80h = Yes; 00 = No)	NO
1EF - 1F1	FE FF FF	Starting Sector (in CHS) FF FF = (1111 1111) (1111 1111) <i>are regrouped as:</i> [11 1111] [11 1111 1111] 3Fh = 63 sectors <i>and</i> 3FFh = 1023 (Cylinder)	Beyond 8.4 GB
1F2	0F	Partition Type	Extended (LBA)
1F3 - 1F5	FE FF FF	Last Sector (in CHS) [Same as for FE FF FF above!]	Beyond 8.4 GB
1F6 - 1F9	78 F0 95 01	Relative Sectors (or Offset) (0195F078h = 26,603,640)	26,603,640
1FA - 1FD	83 AF CC 00	Total Sectors (or Length) (00CCAF83h = 13,414,275)	13,414,275

Last Update: **October 22, 2005**. (22.10.2005)

You can write to me using this: [online reply form](#). (It opens in a new window.)

 [MBR and Boot Records Index](#)

 [The Starman's Realm Index Page](#)

