Atari Hard Disk Partitioning Technical Information

Jean Louis-Guérin – V1.0 November 2009

Table of content

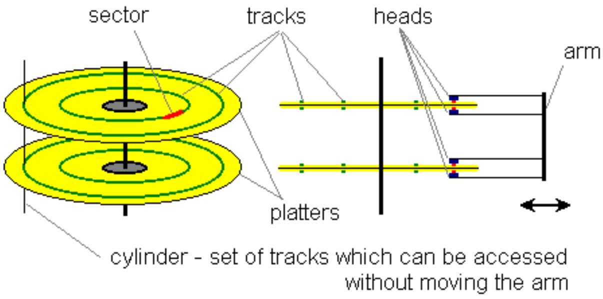
1 Introduction 3
1.1 Hard Disk Primer 3
1.2 Hard Disk Preparation Steps 3
1.3 TOS Partition Size 4
1.4 FAT Partition Type and Size 4
2 TOS Hard Disk Partitioning 5
2.1 Atari Hard Disk Layout 5
2.2 Root Sector 5
2.3 Standard Partition 6
2.3.1 Regular Partition (GEM) Limits 6
2.3.2 Big Partition (BGM) Limits 6
2.3.3 Example of layout with standard partitions 7
2.4 Extended Partition 7
2.4.1 Extended Root Sector 7
2.4.2 Example of layout with extended partitions 8
2.5 TOS Partition Structures 8
2.5.1 Boot sector 8
2.5.2 File Allocation Table 9
2.5.3 Root Directory 10
2.5.4 Position of the different Structures in a TOS Partition 11
2.6 The Boot Sequence 12
3 DOS/FAT Hard Disk Partitioning 13
3.1 Master Boot Record 13
3.2 Primary Partition 13
3.2.1 Example of layout with standard partitions 14
3.3 Extended Partition 14
3.3.1 Extended Master Boot Record 15
3.3.2 Example of layout with extended partitions 16
3.4 FAT Partition Structures 17
3.4.1 Boot sector 17
3.4.2 FS Information Sector 18
3.4.3 File Allocation Table 18
3.4.4 Directory Table 19
3.4.5 Position of the Structures in a DOS Partition 21
3.5 The Boot Sequence 21
4 Atari Hard Disk Partitioning Tests 22
4.1 TOS Partitions 22
4.1.1 Partitioning Example Using HDDRIVER 8.23 22
4.1.2 Atari Bootable TOS Partitions 24
4.1.3 Accessing the TOS Partitions 24
4.1.4 Summary of the tests with TOS partitions 24
4.2 TOS & DOS Partitions 25
4.2.1 Partitioning Example using PPTOSDOS 25
4.2.2 Example using HDDRIVER 8.23 27
4.2.3 Accessing TOS & DOS partition with other Atari hard disk driver 28
4.2.4 Accessing Mixture of DOS and TOS & DOS partitions on Atari 29
4.2.5 Atari Bootable TOS & DOS Partition 29
4.2.6 Summary of the test with TOS & DOS Partition 29
4.3 DOS/FAT Partitions 29
4.3.1 Problems with DOS Partition 29
4.3.2 Accessing Small DOS/FAT Partitions on Atari 31
4.3.3 Accessing Large DOS/FAT Partitions on Atari 31
4.3.4 Accessing Huge DOS/FAT Partitions on Atari 32
4.3.5 Atari Bootable DOS/FAT Partitions 32
4.3.6 Creating DOS Partitions with HDDRIVER 32
4.3.7 Summary of the Tests with DOS partitions 32
5 References 33

1 Introduction

This goal of this document is to provide in-depth technical information about Atari hard disks partitioning. For that matter I describe in detail the TOS File System as well as the DOS/FAT File System as both of them are used on the Atari platform. However the DOS/FAT File System study is limited to what is useful in the Atari platform context. In order to explain the compatibilities and limitations of the different types of partitioning several practical examples are analyzed.

1.1 Hard Disk Primer

First disks had a simple design. They had one or more rotating platters and a moving arm with read/write heads attached to it - one head on each side of the platter. The arm could move and stop at the certain number of positions. When it stopped each head could read or write data on the underlying track. Every read or write had to be done in blocks of bytes, called *Sectors*. Sectors were usually 512 bytes long and there were fixed number of sectors on each track.



When IDE (Integrated Drive Electronics) disks came out the disk space was used more efficiently. Engineers had placed more sectors on the outer tracks, but still provided software writers with a convenient "cubical" look of the disk by doing internal translation of CHS (cylinders, heads, and sectors). Variable sector/cylinder count by early IDE drives is called Zone-bit recording. For example an old 340MB disk has only two platters = 4 heads (sides), but it reports 665 cylinders, 16 heads, and 63 sectors. In reality it, probably, has more then 4*63 sectors on each outer track and a little less than 4*63 on the most inner tracks, but we could never know for sure. With the early IDE disks CPU only has to tell the CHS of the sector that it wants to read and drive's electronics will position the heads to start data transfer.

The maximum allowable values for CHS addressing mode are: 0 to 1023 for cylinders, 0 to 255 for heads, and 1 to 63 for sector. If you multiply these values you will see that the largest hard disk that could be addressed with CHS addressing mode is 8GB.

The newest drives have a simpler interface. Instead of addressing sectors by their CHS (cylinder, head, and sector) address they use LBA (Logical Block Addressing) mode. In LBA mode a program has only to tell the number of the sector from the beginning of the disk (all sectors on disk are numbered 0, 1, 2, 3 ...). Virtually all modern operating systems use LBA addressing, but the CHS notation is still around. First of all, MS-DOS, which is about 20 years old, uses only CHS. Also some programs, like Partition Magic, would not work if partitions do not start at the cylinder or side boundary.

1.2 Hard Disk Preparation Steps

Before a hard disk can be used to store data it must be "prepared". This is done in three steps:

- The first step is called ***low-level formatting*** (often referred as ***formatting*** in Atari world): It is used to create the actual structures on the surface of the media that are used to hold the data. The magnetic medium on the surfaces must be divided into tracks that contain numbered sectors that the controller can find. Once the disk has been formatted, the locations of the tracks and sectors on the disk are fixed in place.

  *Note: With modern SCSI / IDE drives and with drives using SD card this operation **is not required anymore** and therefore is not described in this document. However, if by mistake, you format an already formatted drive in most cases it should not hurt, but you should avoid it if you do not understand exactly the consequences.*

- The second step is called ***partitioning***: Hard drives can be divided into smaller logical drive units called *partitions*. In this way a single hard drive can appear to be two or more drives to the computer. Besides simply keeping drive sizes under the file system limits, dividing a drive also allows partitions to be used for specific purposes, keeping the drive organized. The maximum size of a partition depends on the OS, the Hard Disk Drivers, and the Host Adapter. The partition information is stored in the first physical sector of the disk called the Root Sector for the TOS file system and the Master Boot Record for the DOS/FAT file system.

- The third step is called ***high-level formatting*** (often referred as ***Formatting*** in the PC world and ***Initialization*** in the Atari world): This is the process of creating the basic disk's logical structures: In order for the OS to use a disk it has to know about the number of tracks, the number of sectors per tracks, the size of the sectors and the number of heads. This information is defined in the Boot Sector. Beyond that it is necessary for the OS to find information (e.g. location of all the sectors belonging to this file, attributes ...) about any files stored on the diskette as well as global information (e.g. the space still available on the diskette). This information is kept in the File Allocation Tables (FATs) and the in the Root Directory structure.

1.3 TOS Partition Size

The following table indicates the minimum sector size based on TOS partition sizes:

Partition Size[1]	**Sector Size**
Up to 32MB	512
32MB – 64MB	1024
64 MB – 128MB	2048
128 MB – 256MB	4096
256MB – 512MB	8192
512MB – 1GB[2]	16384

With most of the partitioning programs you only need to specify the size of the partition you want to create and the driver will compute for you the optimum Sector Size. With some hard disk drivers it is possible to modify the sector size (for example with HDDRIVER). In that case you have to make sure that you specify a value greater or equal to the one specified in the table above. Using larger value results in fewer FAT clusters allocation for big files, but with the drawback that small files will occupy more space on the disk.

The maximum size of a partition depends on the TOS version, the Hard Disk drivers, and the capability of the host adapter. With recent hard disk drivers and host adapters, that support the ICD extended command set, the partitions sizes may be:

- Up to 256 megabytes for TOS < 1.04,
- Up to 512 megabytes with TOS ≥ 1.4, and
- Up to 1GB with TOS ≥ 4.0 (Falcon).

1.4 FAT Partition Type and Size

The following table summarizes the characteristic of several types of DOS/FAT partition that are useful to know in the context of the Atari platform:

Partition Type	Fdisk	Size	Fat Type	Version
01	PRI DOS	0-15 MB	12 bits (FAT12)	MS-DOS 2.0
04	PRI DOS	16-32 MB	16 bits (FAT16A)	MS-DOS 3.0
05	EXT DOS	0-2 GB	n/a	MS-DOS 3.3
06	PRI DOS	32 MB-2 GB	16 bits (FAT16B)	MS-DOS 4.0
0E	PRI DOS	32 MB-2 GB	16 bits (FAT16B)	Windows 95[3]
0F	EXT DOS	0-2 GB	n/a	Windows 95
0B	PRI DOS	512 MB - 2 TB	32 bits (FAT32)	OSR2
0C	EXT DOS	512 MB - 2 TB	32 bits (FAT32)	OSR2

[1] Partition size is given for TOS ≥ 1.04. Prior to this version the maximum partition size should be divided by 2
[2] Only supported in TOS 4.x
[3] Type 0x0E and 0x0F forces usage of LBA addressing instead of CHS addressing.

2 TOS Hard Disk Partitioning

In this chapter we will describe the layout and various information concerning the Atari Hard Disks partitioning as defined in the AHDI 3.00 specification. We also look at some enhancements like the one defined by the HDDRIVER, ICD, and PPTOSDOS… Drivers.

Compared to initial Atari specification, AHDI 3.00 adds support for hard disks with more than four partitions, and for partitions of size greater or equal to 32 MB (16 MB if TOS < 1.04).

2.1 Atari Hard Disk Layout

Partitioning and Initialization of the disk write information that defines the layout of the disk:

- The Root Sector (RS) defines the number of partitions and their positions on the disk.
- The optional Bad Sector List contains the list of bad sectors detected on the disk. This is not used anymore on “modern” drive.
- One or up to 4 partitions. There are two kinds of partitions defined in AHDI 3.0: standard partitions and extended partitions:
  - A standard partition contains a number of control structures, necessary to describe the partitions, but most of its content is the actual data. AHDI defines two types of standard partitions: regular partition or big partition (a partition whose size is ≥ 32MB).
  - An extended partition is a special partition that is subdivided into standard partitions.

2.2 Root Sector

The **Root Sector** (RS) of a TOS File System is always the first 512-byte sector (Physical Sector 0) of a partitioned data storage device such as a hard disk. This is equivalent to the Master Boot Record in the FAT file System. The **Root Sector** contains:

- The disk's primary partition table, with one or several entries (up to 4) for the standard partitions. This partition table may also contain one entry for an extended partition.
- And eventually some *bootstrapping* code (also called IPL).

By definition, there are exactly four entries in the primary partition table of the **Root Sector**. The partition size and the partition start address are stored as 32-bit quantities. Because the physical sector size is always 512 bytes, this implies that neither the maximum size of a partition nor the maximum start address (both in bytes) can exceed 2^32 * 512 bytes, or 2 TB.

Offset	Length	Description
$0000		The boot loader code for a boot disk. Not used and filled with 0 for a non bootable disk
$1B6	2	Cylinders
$1B8	1	Heads
$1B9	1	■ $00 = SASI ■ $FF = SCSI
$1BA	2	Write precomp cylinder
$1BC	2	Reduced write current cylinder
$1BE	1	Parking cylinder offset
$1BF	1	Step rate
$1C0	1	Interleave
$1C1	1	Sectors per track
$01C2	4	Hard Disk Size in number of physical (512 bytes) sectors
$01C6 $01D2 $01DE $01EA	4 * 12	Table for the 4 possible partitions described by four 12-byte partitions entry (described below)
$01F6	4	Bad sectors list offset from beginning of disk. Specified in number of physical sectors.
$01FA	4	Bad sectors count in number of physical sectors
$01FE	2	Reserved

The grayed information is considered optional, and is used by a **very few** applications.

Each partition (standard or extended) is defined by a Partition Entry:

Offset	Length	Description	Locations
$00	1	Flag: indicate the state of the partition ■ bit 0 when set partition *exist*, ■ bit 1-6 reserved ■ bit 7 when set partition *bootable* The BIOS will boot the first partition that has bit 7 set	$1C6, $1D2, $1DE, $1EA
$01	3	Id: a 3-bytes ASCII field that identifies the type of partition ■ GEM for regular (< 32MB) partition ■ BGM for big (≥ 32MB) partition ■ XGM for extended partition	$1C7, $1D3, $1DF, $1EB
$04	4	Offset to the beginning of the partition from the beginning of the hard disk. Specified in number of physical (512 bytes) sectors	$1CA, $1D6, $1E2, $1EE
$08	4	Size of the partition in number of physical sectors	$1CE, $1DA, $1E6, $1F2

2.3 Standard Partition

The following is an overview of the order of the structures in standard TOS file system partition:

	Boot Sector	Bad sectors list (optional)	File Allocation Table #1	File Allocation Table #2	Root Directory	Data Region for files and directories... (To end of partition or disk)
size in sectors	(number of reserved sectors)		(number of FATs) * (sectors per FAT)		(number of root entries * 32) / 512	NumberOfClusters * SectorsPerCluster

A TOS file system is therefore composed of these four different regions:

- The **Boot Sectors region** located at the very beginning of a partition: The first logical sector of a standard partition (logical sector 0) is the Boot Sector. It includes an area called the *BIOS Parameter Block* (with some basic file system information, in particular its type, and pointers to the location of the other sections) and may contain some *boot loader* code. The total count of reserved sectors is indicated by a field inside the **Boot Sector**. Important information from the **Boot Sector** is accessible through a TOS structure called the *BIOS Parameter Block* (**BPB**).
- The **FAT region**: This typically contains two copies (may vary) of the File Allocation Table for the sake of redundancy checking, although the extra copy is rarely used, even by disk repair utilities. These are maps of the **Data region**, indicating which clusters are used by files and directories.
- The **Root Directory region**: It contains the Root Directory that stores information about the files and directories located in the **Root Directory**. The **Root Directory** has a fixed size which is pre-allocated at creation of the volume.
- The **Data Region**: This is where the actual file and directory data is stored and takes up most of the partition. The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT

The Atari AHDI 3.00 specifies two types of standard partition:

- The *regular partition* (GEM Partition) and,
- The *big partition* (BGM Partition)

2.3.1 Regular Partition (GEM) Limits

- Size of a physical sector in number of bytes = 512
- Maximum number of sectors = 2^15 = 32768 (< TOS 1.04[4])
- Maximum number of sectors = 2^16 = 65536 (>= TOS 1.04)
- Maximum size of a partition in number of bytes = 32768 * 512 = 16 MB (< TOS 1.04)
- Maximum size of a partition in number of bytes = 65536 * 512 = 32 MB (≥ TOS 1.04)

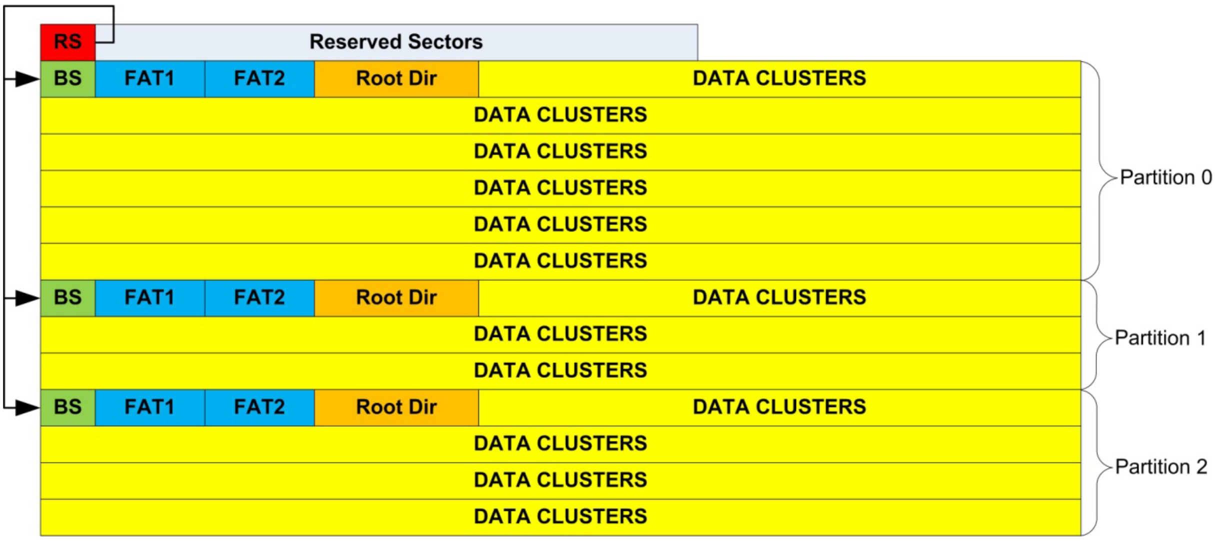
2.3.2 Big Partition (BGM) Limits

- Maximum size of a cluster in number of bytes = 2^14 = 16384
- Size of a cluster in number of logical sectors = 2
- Maximum size of a logical sector in number of bytes = 16384 / 2 = 8192
- Maximum number of logical sectors = 2^15 = 32768 (< TOS 1.04)
- Maximum number of logical sectors = 2^16 = 65536 (>= TOS 1.04)
- Maximum size of a partition in number of bytes = 32768 * 8192 = 256 MB (< TOS 1.04)
- Maximum size of a partition in number of bytes = 65536 * 8192 = 512 MB (≥ TOS 1.04)

4 Note that prior to TOS 1.04 the number of sectors is stored as a signed integer resulting in a maximum of 32768 sectors. Starting with TOS 1.04 the number of sectors is stored as an unsigned integer resulting in a maximum of 65536 sectors

2.3.3 Example of layout with standard partitions

In the following example we have a hard disk with 3 standard partitions. The **Root Sector** contains 3 pointers to the 3 partitions. These partitions can be either regular or big partitions.



2.4 Extended Partition

Extended partition enables a hard disk to contain more than 4 partitions. Only one entry in the Atari partition table can contain an extended partition. The extended partition is identified by the ASCII characters “XGM” in the **id** field of the partition entry. Since an extended partition is not bootable, it must be preceded by at least one standard partition, so the hard disk can be made bootable. This requirement makes it impossible for the first partition to be an extended partition.

An extended partition is subdivided into smaller ones. Each subdivision consists of an **Extended Root Sector** (**ERS**), and a Standard Partition. Conceptually, each subdivision is like a stand-alone hard disk with only one partition on it. These subdivisions are *linked* together by a pointer in the **Extended Root Sector**.

2.4.1 Extended Root Sector

The layout of an **Extended Root Sector** resembles that of the Root Sector, except that it only contains the partition table. Only two of the four partition table entries can be used, but not necessarily the first two. One of them is used to describe the *Standard Partition* in the current subdivision; the other one provides eventually a link to the next subdivision. The link should occupy the entry that follows the entry for the description of the standard partition. The other two unused entries should be filled with zeroes.

For the standard partition description, the definitions of the fields in the partition table entry are:

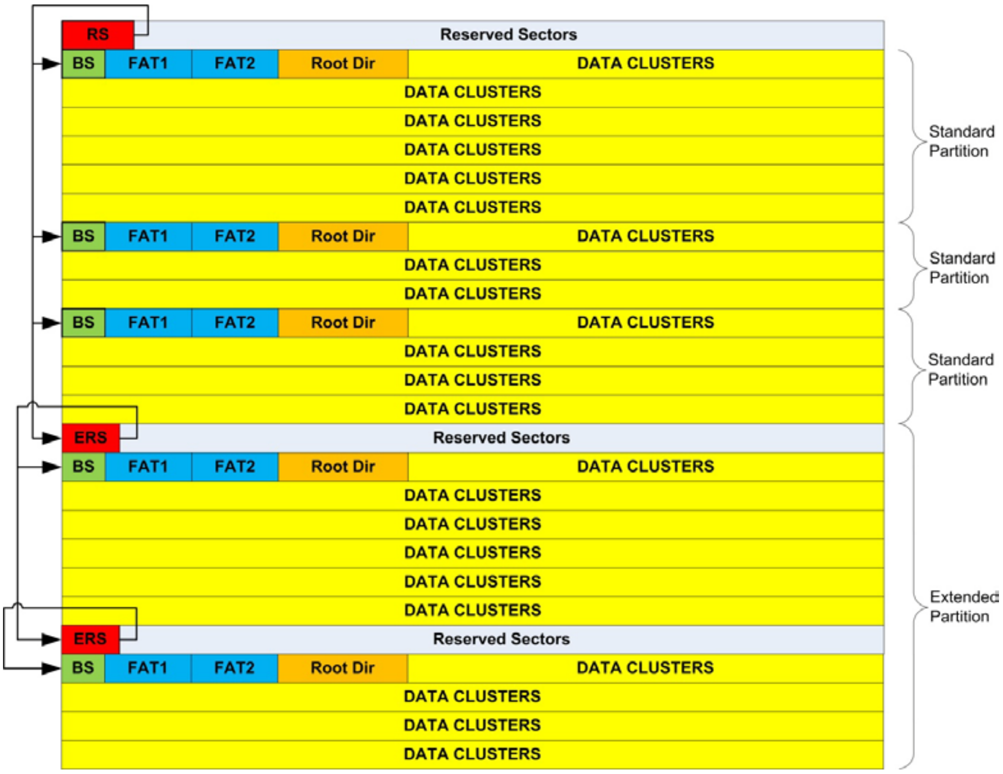
Offset	Length	Description
$00	1	Flag: indicate the state of the partition ■ bit 0 when set partition exist, ■ bit 1-7 reserved
$01	3	Id: a 3-bytes ASCII field that identifies the type of partition ■ GEM for regular (< 32MB) partition ■ BGM for big (≥ 32MB) partition
$04	4	Offset to the beginning of the standard partition from the beginning of the extended root sector that this structure reside in. In number of physical (512 bytes) sectors
$08	4	Size of the partition in number of physical sectors

For the link to the next partition, the definitions of the fields in the partition table entry are:

Offset	Length	Description
$00	1	Flag: indicate the state of the partition ■ bit 0 when set partition exist, ■ bit 1-7 reserved
$01	3	Id: a 3-bytes ASCII field that identifies the type of partition ■ XGM must be used
$04	4	Offset to the beginning of the next subdivision from the beginning of the entire extended partition. In number of physical (512 bytes) sectors
$08	4	Size of the partition in number of physical sectors

2.4.2 Example of layout with extended partitions

In the following example we have a hard disk with 3 standard partitions and an extended partition that contains two standard partitions. The **Root Sector** contains 3 pointers to the 3 standard partitions and a pointer to the extended partition that starts with the first **Extended Root Sector**. This ERS contains a pointer to a standard partition and a pointer to the next **Extended Root Sector**. The second ERS contains only a pointer to the second standard partition as it is the last in the chain.



2.5 *TOS Partition Structures*

2.5.1 Boot sector

The **Boot Sector** is located in the first logical sector of a **logical drive** (standard partition) and it occupies one logical sector. When a logical sector contains more than one physical (512-byte) sectors, the **Boot Sector** will be bigger than 512 bytes. However, only the first 512 bytes of a **Boot Sector** are used, no matter how big the **Boot Sector** might be. The rest of the **Boot Sector** is zero-filled.

This sector is read by the **TOS** to find important information about the disk. Some parameters are loaded from this sector to be used by the BIOS and are stored in a TOS structure called the **BPB**[5] (Bios Parameter Block). Eventually the **Boot Sector** also contains a *bootstrap routine* that allow starting a relocatable program at boot time.

[5] The Atari BPB is based on MS-DOS version 2.x BPB.

The structure of the **Boot Sector**:

Name	Offset	Bytes	Contents
BRA	$00	2	This word contains a 680x0 BRA.S instruction to the bootstrap code in this sector if the disk is executable, otherwise it is unused.
OEM	$02	6	These six bytes are reserved for use as any necessary filler information.
SERIAL	$08	4	The low 24-bits of this long represent a unique disk serial number.
BPS	$0B	2	This is an Intel format word (low byte first) which indicates the number of bytes per logical sector on the disk.
SPC	$0D	1	This is a byte which indicates the number of sectors per cluster (must be a power of 2). The only value supported by GEMDOS is 2.
RES	$0E	2	This is an Intel format word which indicates the number of reserved logical sectors at the beginning of the logical drive, including the boot sector itself. It is usually one.
NFATS	$10	1	This is a byte indicating the number of File Allocation Table's (FAT's) stored on the disk. Usually the value is two.
NDIRS	$11	2	This is an Intel format word indicating the total number of file name entries that can be stored in the root directory of the volume.
NSECTS	$13	2	This is an Intel format word indicating the number of sectors on the disk (including those reserved).
MEDIA	$15	1	This byte is the media descriptor. For hard disks this value is set to $F8. It is not used by the ST BIOS.
SPF	$16	2	This is an Intel format word indicating the size occupied by each of the FATs on the volume[6].
SPT	$18	2	This is an Intel format word indicating the number of sectors per track. This field is not used for hard drive
NHEADS	$1A	2	This is an Intel format word indicating the number of heads on the media. Not used for Hard Disk
NHID	$1C	2	This is an Intel format word indicating the number of hidden sectors. Not used by Atari hard drive.
	$1E		Boot Code if Any
	$1FE	2	Checksum

The grayed areas are read from the boot sector and stored in the BPB.

The last word in the boot sector (at offset $1FE) is reserved to “evening out” the sector checksum. To be bootable a **Boot Sector** checksum must be equal to the magic number $1234.

2.5.2 File Allocation Table

The File Allocation Table structures (**FAT**) is an array used by the TOS to keep track of which clusters on a drive have been allocated for each file or directory. As a program creates a new file or adds to an existing one, the system allocates sectors for that file, writes the data to the given sectors, and keeps track of the allocated sectors by recording them in the FAT. To conserve space and speed up record-keeping, each record in the FAT corresponds to two or more consecutive sectors (called a cluster). The number of sectors in a cluster depends on the type and capacity of the drive but is always a power of 2 (the only value supported by GEMDOS is 2). Every logical drive has at least one FAT, and most drives have two, one serving as a backup should the other fail. The FAT immediately follows the Boot Sector and any other reserved sectors.

Depending on the number of clusters on the drive, the FAT consists of an array of either 12-bit or 16-bit entries. Drives with more than 4086 clusters have a 16-bit FAT; those with 4086 or fewer clusters have a 12-bit FAT (typically only used by Floppy disks).

The first two entries in a FAT are reserved. In most cases the first byte contains the media descriptor (usually $F8) and the additional reserved bytes are set to $FF. Each FAT entry represents a corresponding cluster on the drive. If the cluster is part of a file or directory, the entry contains either a marker specifying the cluster as an index pointing to the next cluster in the file or directory, or the last in that file or directory. If a cluster is not part of a file or directory, the entry contains a value indicating the cluster's status. The **SCLUSTER** field in the Root Directory corresponding to the file or directory specifies the index of the first FAT entry for the file or directory.

[6] Given this information, together with the number of FATs and reserved sectors listed above, we can compute where the root directory begins. Given the number of entries in the root directory, we can also compute where the user data area of the disk begins.

The following table shows possible FAT entry values:

FAT12 Value	FAT16 Value	Meaning
$000	$0000	Available cluster.
$002-$FEF	$0002-$FFEF	Index of entry for the next cluster in the file or directory. Note that $001 does not appear in a FAT, since that value corresponds to the FAT's second reserved entry. Index numbering is based on the beginning of the FAT
$FF0-$FF6	$FFF0-$FFF6	Reserved
$FF7	$FFF7	Bad sector in cluster; do not use cluster.
$FF8-$FFF	$FFF8-$FFFF	Last cluster of file or directory. (usually the value $FFF is used)

For example, the following segment of a 12-bit FAT shows the FAT entries for a file consisting of four clusters:

- $000 $F8 $FF $FF (2 reserved entries)
- $003 Cluster 2 points to cluster 3
- $005 Cluster 3 points to cluster 5
- $FF7 Cluster 4 contains a bad sector
- $006 Cluster 5 points to cluster 6
- $FFF Cluster 6 is the last cluster for the file
- $000 Clusters 7 is available
- ...

Note: If a cluster contains $000 this does not mean that it is empty but that it is available. This is due to the fact that when a file is deleted the data are not erased but only the first letter of the name of the file in the directory structure is set to $E5 and all clusters used by the deleted file are set to $000.

2.5.3 Root Directory

The TOS arranges and stores file-system contents in directories. Every file system has at least one directory, called the **Root Directory** (also referred as the **Catalog** in Atari), and may have additional directories either in the **Root Directory** or ordered hierarchically below it. The contents of each directory are described in individual directory entries. The TOS strictly controls the format and content of directories.

The **Root Directory** is always the topmost directory and it is created during initialization of a partition. The **Root Directory** can hold information for only a fixed number of files or other directories, and the number cannot be changed without reformatting the partition. A program can identify this limit by examining the **NDIRS** field in the **BPB** structure described in the Boot Sector. This field specifies the maximum number of root-directory entries for the partition.

A user or a program can add new directories within the current directory, or within other directories. Unlike the **Root Directory**, the new directory is limited only by the amount of space available on the medium, not by a fixed number of entries. The TOS initially allocates only a single cluster for the directory, allocating additional clusters only when they are needed. Every directory except the **Root Directory** has two entries when it is created. The first entry specifies the directory itself, and the second entry specifies its parent directory (the directory that contains it). These entries use the special directory names "." (An ASCII period) and ".." (Two ASCII periods) respectively.

The TOS gives programs access to files in the file system. Programs can read from and write to existing files, as well as create new ones. Files can contain any amount of data, up to 4GB or the limits of the limit of the data region on a partition. Apart from its contents, every file has a name (possibly with an extension), access attributes, and an associated date and time. This information is stored in the file's directory entry, not in the file itself.

The **Root Directory** is located just after the FATs. Each entry in the **Root Directory** is described by the following 32 bytes long structure:

Name	Bytes	Contents
FNAME	8	Specifies the name of the file or directory. If the file or directory was created by using a name with fewer than eight characters, space characters (ASCII $20) fill the remaining bytes in the field. The first byte in the field can be a character or one of the following values: ■ $00: The directory entry has never been used. The TOS uses this value to limit the length of directory searches. ■ $05: The first character in the name has the value $E5. ■ $2E: The directory entry is an alias for this directory or the parent directory. If the remaining bytes are space characters (ASCII 20h), the **SCLUSTER** field contains the starting cluster for this directory. If the second byte is also $2E (and the remaining bytes are space characters), **SCLUSTER** contains the starting cluster number of the parent directory, or zero if the parent is the root directory. ■ $E5: The file or directory has been deleted.
FEXT	3	Specifies the file or directory extension. If the extension has fewer than three characters, space characters (ASCII $20) fill the remaining bytes in this field.
ATTRIB	1	Specifies the attributes of the file or directory. This field can contain some combination of the following values: ■ $01: Specifies a read-only file. ■ $02: Specifies a hidden file or directory. ■ $04: Specifies a system file or directory. ■ $08: Specifies a volume label. The directory entry contains no other usable information (except for date and time of creation) and can occur only in the root directory. ■ $10: Specifies a directory. ■ $20: Specifies a file that is new or has been modified. ■ All other values are reserved. (The two high-order bits are set to zero.) If no attributes are set, the file is a normal file.
RES	10	Reserved; do not use.
FTIME	2	Specifies the time the file or directory was created or last updated. The field has the following form: ■ Bits 0-4: Specifies two-second intervals. Can be a value in the range 0 - 29. ■ Bits 5-10: Specifies minutes. Can be a value in the range 0 - 59. ■ Bits 11-15: Specifies hours. Can be a value in the range 0 - 23.
FDATE	2	Specifies the date the file or directory was created or last updated. The field has the following form: ■ Bits 0-4: Specifies the day. Can be a value in the range 1 through 31. ■ Bits 5-8: Specifies the month. Can be a value in the range 1 through 12. ■ Bits 9-15: Specifies the year, relative to 1980.
SCLUSTER	2	Specifies the starting cluster of the file or directory (index into the FAT)
FSIZE	4	Specifies the maximum size of the file, in bytes.

2.5.4 Position of the different Structures in a TOS Partition

- The position of the **Boot Sector** P_{BS}(the beginning of a logical partition) is directly given in the **Root Sector** or in an **Extended Root Sector**
- The position of the first **FAT** P_{FAT1} is equal to the position of the **boot sector** plus the number of reserved sector:
  $P_{FAT1} = P_{BS} + RES * (BPS/512)$
- The position of the second **FAT** P_{FAT2} is equal to the position of the P_{FAT1} plus the size of the **FAT**:
  $P_{FAT2} = P_{FAT1} + SPF * (BPS/512)$
- The position of the **Root Directory** P_{RD} is equal to the position of P_{FAT2} plus the size of the **FAT**:
  $P_{RD} = P_{FAT2} + SPF * (BPS/512)$
- The position of the first data cluster P_{DATA} is equal to the position of the **Root Directory** plus the size of the **Root Directory**:
  $P_{DATA} = P_{RD} + NDIRS * (32/512)$

The Size of the data region = Number of Clusters * Sectors per Cluster

2.6 The Boot Sequence

In this section we describe a *typical* sequence to load a hard disk driver from a bootable hard disk partition. The actual implementation may differ.

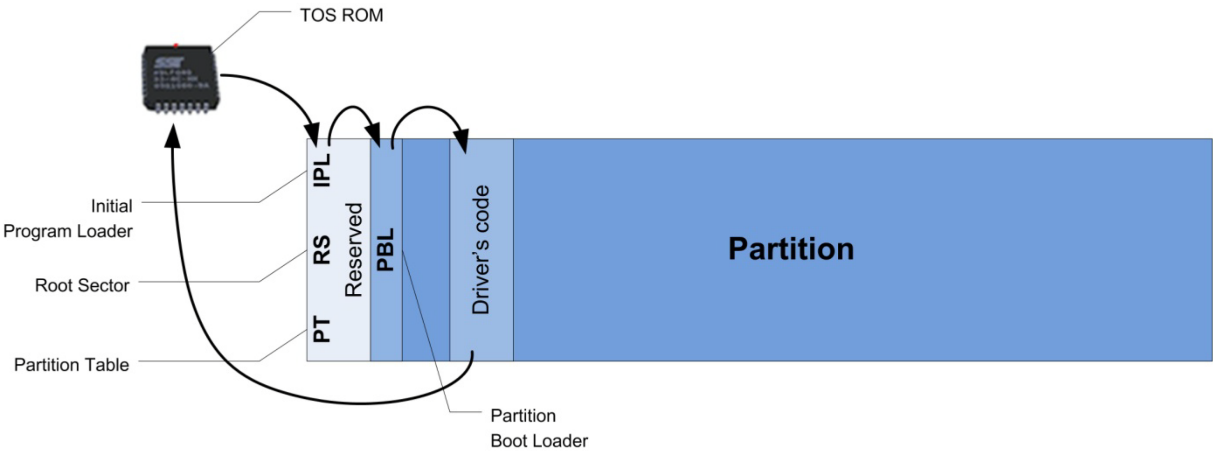
During the TOS ROM System initialization an attempt is made to load Root Sector (RS) from the DMA bus. For each of the eight DMA bus devices the TOS attempts a read operation on the first physical sector 0.

If the read is successful, and if the sector checksums is equal to $1234, then the **Root Sector** program is executed. Note that Root Sector Program is a misleading name because the **Root Sector** is actually composed of a boot-loader program and a partition table. The most common name for the boot-loader program part of the RS is the **Initial Program Loader** (IPL). The IPL is very small and quite limited. Its main job is just to find and start the next program in the chain. For that matter it usually looks at the partition table to see if any of the entries there has a flag to indicate a *bootable partition*. If it found one then it usually goes to the very first byte of that partition (the Boot Sector) and starts the program that it finds there (If several partition are bootable the IPL select the first).

The next program in the chain is at the very beginning of the partition in the Boot Sector. This program is often called the **Partition Boot Loader** (PBL). The PBL will do its job and then start the next program: usually the hard disk driver loader. The location of the next program will be different for various hard disk drivers. During the installation of a hard disk driver the PBL will be written with the information necessary to find the driver file location. The boot loader code can perform a variety of tasks. In some cases it can, for example, load the hard disk driver from the first track of the disk, which it assumes to be “free” space (that is not allocated to any disk partition), and executes it. In others cases, it uses a table of embedded disk locations to locate the hard disk driver loader and execute it.

The last program in the chain is the actual hard disk driver loader. This program loads in memory the necessary code to handle the disk drives and finally returns to the TOS program to start GEMDOS.

For Atari the boot sequence is: → TOS → IPL → PBL → HD Driver → GEMDOS



In these graphic the RS is shown as a separate section at the very start of the hard drive. It is indeed separate and not connected in any way to the following partitions. Convention is to reserve a small section of the drive specifically for the RS to reside on. I've shown the PBL as a separate section but it is actually a part of the partition it is in.

Note that the PPTOSDOS hard disk driver uses a slightly different boot sequence. It does not use (and therefore does not write) a PBL in any of the partitions. Instead the IPL call directly the HD driver loader code. This code is located in the reserved area at the beginning of the disk (starting at sector 2). This allows an easy selection at boot time of the boot partition. You can therefore switch between different configurations by selecting a specific partition with the required AUTO, ACC, Desktop Settings…

3 DOS/FAT Hard Disk Partitioning

In this chapter I describe the layout and various information concerning the DOS/FAT Hard Disks partitioning. PC hard disk partitioning is a vast subject and I will only present here information that can be useful in the context of its usage on Atari.

The layout of PC DOS hard disks is similar but not identical to layout of Atari hard disks.

Partitioning and Initialization of the disk write information that defines the layout of the disk:

- The Master Boot Record (MBR) defines the number of partitions and their positions on the disk.
- The Reserved Sectors is optional. However, for historical reason, a partition on a FAT file system is aligned on a cylinder boundary (Cylinder 0, head 1, Sector 1 in CHS notation). The 62 sectors gap between them is left unused. This is not required with LBA drives, but we need to follow this rule in order to make happy old software (MS-DOS for example).
- One or several partitions.

There are two types of partitions: *primary* partitions and *extended partitions*:

- A *primary partition* contains a number of control structures, necessary to describe the partitions, but most of its content is the actual data.
- An *extended partition* is a special kind of partition which itself is subdivided into *primary partitions*.

3.1 Master Boot Record

The first physical sector 0 on a disk on a hard disk contains the *Master Boot Record* structure (this equivalent to the Atari Root Sector):

Offset	Length	Description
0x0000	440	Boot loader code. Filled with zero if disk is not bootable.
0x01B8	4	Optional Disk signature
0x01BC	2	Usually Nulls; 0x0000
0x01BE 0x01CE 0x01DE 0x01EE	4 * 16	Table of partitions: Four 16-byte entries, IBM Partition Table scheme ■ $1BE, $1BF, $1C2, $1C3, $1C6 ■ $1CE, $1CF, $1D2, $1D3, $1D6 ■ $1DE, $1DF, $1E2, $1E3, $1E6 ■ $1EE, $1EF, $1F2, $1F3, $1F6
0x01FE	2	MBR Signature $AA55 in little-endian format

There are four 16-byte structures to describe each partition:

Offset	Length	Description
0x00	1	state (0x80 = bootable (*active*), 0x00 = non-bootable, other = invalid)
0x01	3	CHS address of first block in partition described in the next 3 bytes.
0x01	1	■ Head (0-254)
0x02	1	■ Sector is in bits 5–0; bits 9–8 of Cylinder are in bits 7–6
0x03	1	■ bits 7–0 of Cylinder
0x04	1	partition type
0x05	3	CHS address of last block in partition described in the next 3 bytes.
0x05	1	■ Head
0x06	1	■ Sector is in bits 5–0; bits 9–8 of Cylinder are in bits 7–6
0x07	1	■ bits 7–0 of Cylinder
0x08	4	LBA of first sector in the partition
0x0C	4	number of blocks in partition, in little-endian format

3.2 Primary Partition

A primary partition contains one FAT file system. The “partition type” code for a primary partition describes the type of the file system. The FAT file systems have made use of quite a number of partition type codes over time due to the limits of various DOS and Windows OS versions. Please refer to FAT Partition Type and Size for a short summary of partition types useful in the context of the Atari platform.

The following is an overview of the order of the structures in a primary FAT file system partition:

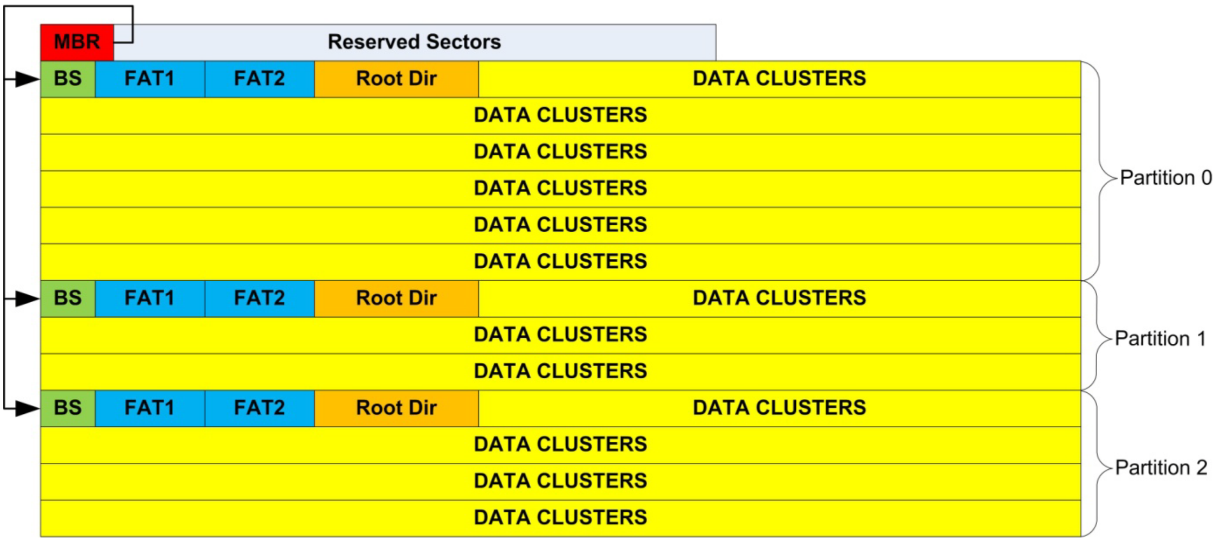
	Boot Sector	FS Information Sector (FAT32 only)	More reserved sectors (optional)	File Allocation Table #1	File Allocation Table #2	Root Directory (FAT12/16 only)	Data Region (for files and directories) ... (To end of partition or disk)
size in sectors	(number of reserved sectors)			(number of FATs) * (sectors per FAT)		(number of root entries * 32) / Bytes per sector	NumberOfClusters * SectorsPerCluster

A FAT file system is therefore composed of these four sections:

- The **Boot sectors region**, located at the very beginning of the partition: The first reserved sector (logical sector 0) is the Boot Sector. It includes an area called the *BIOS Parameter Block* (with some basic file system information, in particular its type, and pointers to the location of the other sections) and usually it contains the operating system's *boot loader* code. The total count of reserved sectors is indicated by a field inside the **Boot Sector**. Important information from the **Boot Sector** is accessible through a DOS structure called the *BIOS Parameter Block* (**BPB**). For FAT32 file systems, the reserved sectors include a File System Information *Sector,* usually at sector 1, and a **Backup Boot Sector,** usually at Sector 6. The exact location of these two sectors is specified in the extended FAT32 BPB.
- The **FAT region**: This typically contains two copies (may vary) of the File Allocation Table for the sake of redundancy checking, although the extra copy is rarely used, even by disk repair utilities. These are maps of the **Data region**, indicating which clusters are used by files and directories.
- The **Root Directory region**: This is the Directory Table that stores information about the files and directories located in the **Root Directory**. It imposes on the **Root Directory** a fixed maximum size which is pre-allocated at creation of this volume.
- The **Data region**: This is where the actual file and directory data is stored and takes up most of the partition. The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the **FAT**

3.2.1 Example of layout with standard partitions

In the following example we have a hard disk with 3 primary partitions. The Master Boot Record contains 3 pointers to the 3 partitions. These partitions can be either regular or big partitions.



3.3 Extended Partition

An extended partition is a special partition which contains *secondary partition(s)*. A hard disk may contain only one extended partition; which can then be sub-divided into many *logical drives*.

3.3.1 Extended Master Boot Record

The first sector of the Extended Partition contains an **Extended Master Boot Record** (EMBR). It is very similar to the Master Boot Record.

The **Extended Master Boot Record** contains the following information:

Offset	Length	Description
$0000	455	Normally unused and filled with 0.
$01BE	16	Partition Table's First entry
$01CE	16	Partition Table's Second entry
$01DE	32	Unused, but should be filled with zero-bytes
$01FE	2	MBR Signature $AA55 in little-endian format

Where a Partition Table entry contains:

Offset	Length	Description
0x00	1	state (0x80 = bootable (*active*), 0x00 = non-bootable, other = invalid)
0x01	3	CHS address of first block in partition described in the next 3 bytes.
0x01	1	■ Head (0-254)
0x02	1	■ Sector is in bits 5–0; bits 9–8 of Cylinder are in bits 7–6
0x03	1	■ bits 7–0 of Cylinder
0x04	1	partition type
0x05	3	CHS address of last block in partition described in the next 3 bytes.
0x05	1	■ Head
0x06	1	■ Sector is in bits 5–0; bits 9–8 of Cylinder are in bits 7–6
0x07	1	■ bits 7–0 of Cylinder
0x08	4	LBA of first sector in the partition
0x0C	4	number of blocks in partition, in little-endian format

The **first entry** of an EMBR partition table points to the logical partition belonging to that EMBR:

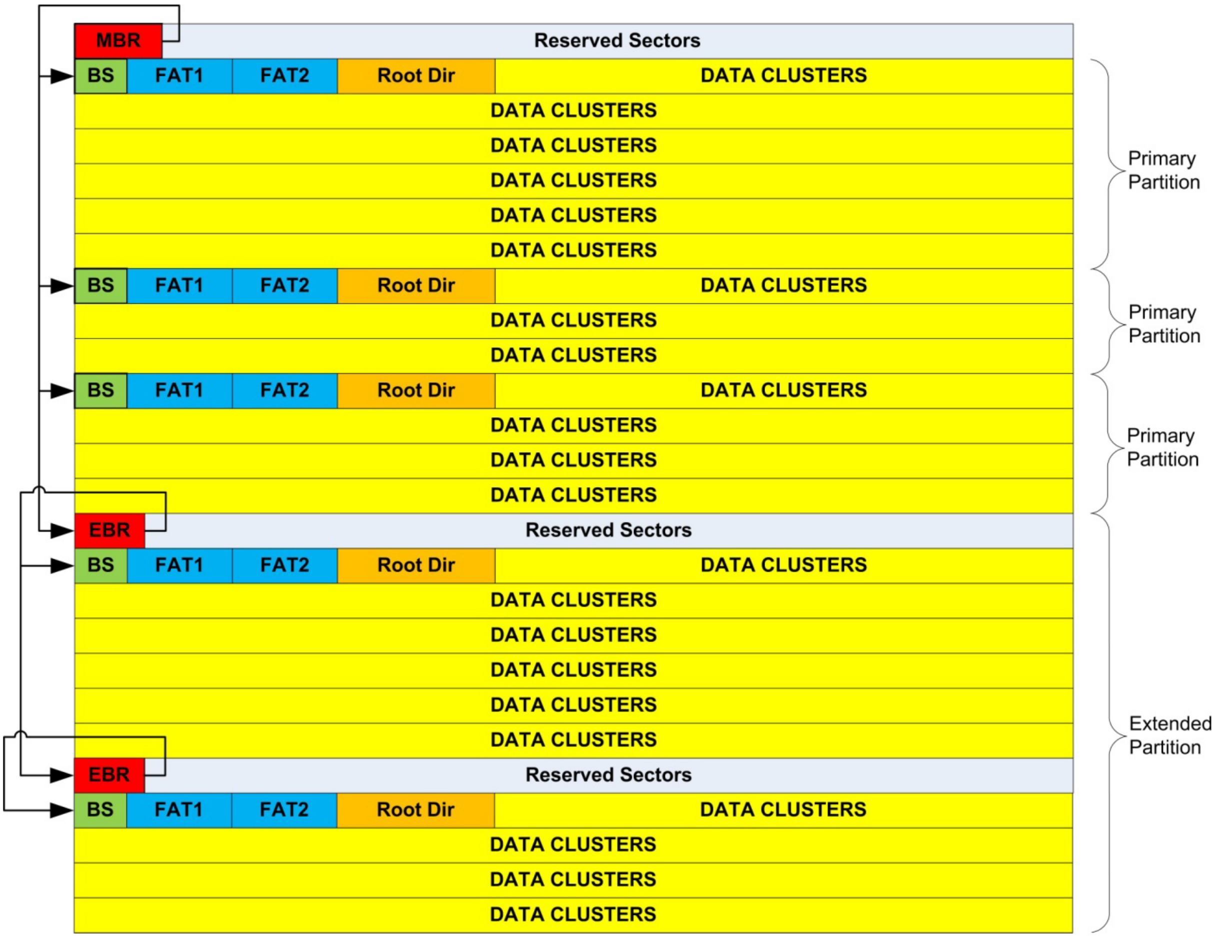
- Starting Sector = relative offset between this EMBR sector and the first sector of the logical partition
  Note: This will be the same value for each EMBR on the same hard disk; usually 63.
- Number of Sectors = total count of sectors for this logical partition
  Note: The unused sectors in the same track as the EMBR are not considered part of the logical partition for this count value.

The **second entry** of an EMBR partition table will contain zero-bytes if it's the last EMBR in the extended partition; otherwise, it points to the next EMBR in the EMBR chain:

- Starting Sector = relative address of next EMBR within extended partition
  in other words: Starting Sector = LBA address of next EMBR minus LBA address of extended partition's first EMBR
- Number of Sectors = total count of sectors for next logical partition, but count starts from the next EMBR sector
  Note: Unlike the first entry in an EMBR's partition table, this Number of Sectors count includes the next logical partition's EMBR sector along with the other sectors in its otherwise unused track.

3.3.2 Example of layout with extended partitions

In the following example we have a hard disk with 3 primary partitions and an extended partition that contains two embedded primary partitions. The Master Boot Record contains 3 pointers to the 3 *standard partitions* and a pointer to the *extended partition.* The first logical sector of the *extended partition* contains the first **Extended Master Boot Record**. This EMBR contains in turn a pointer to the primary partition and a pointer to the next **Extended Master Boot Record**. This second EMBR contains in turn a pointer to a primary partition.



3.4 FAT Partition Structures

3.4.1 Boot sector

The boot sector is the first logical sector of a logical drive and it occupies one logical sector. The grayed areas are read from the boot sector and stored in the BIOS Parameter Block (BPB).

Name	Offset	Length	Description
BRA	0x00	3	Jump instruction. This instruction will be executed and will skip past the rest of the (non-executable) header if the partition is booted from.
OEM	0x03	8	OEM Name (padded with spaces). This value determines in which system disk was formatted. MS-DOS checks this field to determine which other parts of the boot record can be relied on.
BPS	0x0b	2	Bytes per Sector. A common value is 512, especially for file systems on IDE (or compatible) disks. The *BIOS Parameter Block* starts here.
SPC	0x0d	1	Sectors per Cluster. Allowed values are powers of two from 1 to 128. However, the value must not be such that the number of bytes per cluster becomes greater than 32KB.
RES	0x0e	2	Reserved sector count. The number of sectors before the first FAT in the file system image (including boot sector). Typically 1 for FAT12/FAT16.
NFATS	0x10	1	Number of file allocation table following the reserved sectors. Almost always 2. The second FAT is used by recovery program if the first FAT is corrupted.
NDIRS	0x11	2	Maximum number of root directory entries. This value should always be such that the root directory ends on a sector boundary (i.e. such that its size becomes a multiple of the sector size). 0 for FAT32
NSECTS	0x13	2	Total number of sectors on the drive. If the size of the drive is greater than 32MB, this field is set to zero and the number of sectors is specified in the huge number of sectors field at offset 0x20 (HSECTS). 0 for FAT32
MEDIA	0x15	1	Media descriptor: Usually 0xF8 for Hard disk. Same value of media descriptor should be repeated as first byte of each copy of FAT.
SPF	0x16	2	Number of Sectors per File Allocation Table
SPT	0x18	2	Number of Sectors per single Track
NHEADS	0x1a	2	Number of heads on the drive
NHID	0x1c	4	Number of Hidden sectors
HSECTS	0x20	4	Huge number of Sectors (when more than 65535 sectors) otherwise, see NSECTS at offset 0x13. This field allow support for drives larger than 32MB

3.4.1.1 Extended BIOS Parameter Block used by FAT12 and FAT16:

Name	Offset	Length	Description
DRNUM	0x24	1	Drive ID: Specifies whether the drive is the first hard disk drive (value 0x80) or not (value 0x00). Used internally by MS-DOS
	0x25	1	Reserved
EBSIG	0x26	1	Extended boot signature. Value is 0x29 (or 0x28).
VOLID	0x27	4	Volume serial number
VLAB	0x2b	11	Volume Label, padded with blanks (0x20).
FSTYPE	0x36	8	FAT file system type, padded with blanks (0x20), e.g.: "FAT12 ", "FAT16 ". This is not meant to be used to determine drive type; however, some utilities use it in this way.
	0x3E	448	Operating system boot code
	0x1FE	2	Boot sector signature (0x55 0xAA)

3.4.1.2 Extended BIOS Parameter Block used by FAT32:

Offset	Length	Description
0x24	4	Big Sectors per FAT
0x28	2	Extended FAT Flags
0x2a	2	FS Version
0x2c	4	First Cluster number of root directory
0x30	2	Sector number of FS Information Sector
0x32	2	Sector number of a copy of this boot sector
0x34	12	Reserved
0x40	1	Physical Drive Number (Drive ID)
0x41	1	Reserved for NT
0x42	1	Extended boot signature.
0x43	4	ID (serial number)
0x47	11	Volume Label
0x52	8	FAT file system type: "FAT32 "
0x5a	420	Operating system boot code
0x1FE	2	Boot sector signature (0x55 0xAA)

3.4.2 FS Information Sector

The **FS Information Sector** was introduced in FAT32 for speeding up access times of certain operations (in particular, getting the amount of free space). It is located at a sector number specified in the boot record at position 0x30 (usually sector 1, immediately after the boot record).

Byte Offset	Length (bytes)	Description
0x00	4	FS information sector signature (0x52 0x52 0x61 0x41 / "RRaA")
0x04	480	Reserved (byte values are 0x00)
0x1e4	4	FS information sector signature (0x72 0x72 0x41 0x61 / "rrAa")
0x1e8	4	Number of free clusters on the drive, or -1 if unknown
0x1ec	4	Number of the most recently allocated cluster
0x1f0	14	Reserved (byte values are 0x00)
0x1fe	2	FS information sector signature (0x55 0xAA)

3.4.3 File Allocation Table

A partition is divided up 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 partition, typically cluster sizes lie somewhere between 2 KB and 32 KB. Each file may occupy one or more of these clusters depending on its size; thus, a file is represented by a chain of these clusters (referred to as a singly linked list). However these clusters are not necessarily stored adjacent to one another on the disk's surface but are often instead *fragmented* throughout the Data Region.

The **file allocation table** (**FAT**) is a list of entries that map to each cluster on the partition. Each entry records one of five 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 special entry to mark a reserved cluster
- a zero to note that the cluster is unused

Each version of the FAT file system uses a different size for FAT entries. Smaller numbers result in a smaller FAT table, 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 you consider the 3 bytes as one little-endian 24-bit number, the 12 least significant bits are the first entry and the 12 most significant bits are the second.

FAT entry values:

FAT12	FAT16	FAT32	Description
0x000	0x0000	0x00000000	Free Cluster
0x001	0x0001	0x00000001	Reserved value; do not use
0x002–0xFEF	0x0002–0xFFEF	0x00000002–0x0FFFFFEF	Used cluster; value points to next cluster
0xFF0–0xFF6	0xFFF0–0xFFF6	0x0FFFFFF0–0x0FFFFFF6	Reserved values; do not use.
0xFF7	0xFFF7	0x0FFFFFF7	Bad sector in cluster or reserved cluster
0xFF8–0xFFF	0xFFF8–0xFFFF	0x0FFFFFF8–0x0FFFFFFF	Last cluster in file

The first cluster of the Data Region is cluster #2. That leaves the first two entries of the FAT unused. In the first byte of the first entry a copy of the media descriptor is stored (usually 0xF8). The remaining 8 bits if FAT16 or 20 bits if FAT32 of this entry are 1. In the second entry the end-of-cluster-chain marker is stored. The high order two bits of the second entry are sometimes, in the case of FAT16, used for dirty volume management: high order bit 1: last shutdown was clean; next highest bit 1: during the previous mount no disk I/O errors were detected. Note that FAT32 uses only 28 bits of the 32 possible bits. The upper 4 bits are usually zero (as indicated in the table above) but are reserved and should be left untouched.

3.4.4 Directory Table

A **Directory Table** is a special type of file that represents a directory (also known as a **folder**). Each file or directory stored within it is represented by a 32-byte entry in the table. Each entry records the name, extension, attributes (archive, directory, hidden, read-only, system and volume), the date and time of creation, the address of the first cluster of the file/directory's data and finally the size of the file/directory. Aside from the **Root Directory Table** in FAT12 and FAT16 file systems, which occupies the special **Root Directory region** location, all **Directory Tables** are stored in the **Data region**. The actual number of entries in a directory stored in the **Data region** can grow by adding another cluster to the chain in the FAT.

Legal characters for DOS file names include the following:

- Upper case letters A–Z
- Numbers 0–9
- Space (though trailing spaces in either the base name or the extension are considered to be padding and not a part of the file name, also filenames with space in them could not be used on the DOS command line prior to Windows 95 because of the lack of a suitable escaping system)
- ! # $ % & ' ( ) - @ ^ _ ` { } ~
- Values 128–255

This excludes the following ASCII characters:

- " * / : < > ? \ |
  Windows/MSDOS has no shell escape character
- + , . ; = [ ]
  They are allowed in long file names only.
- Lower case letters a–z
  Stored as A–Z. Allowed in long file names.
- Control characters 0–31
- Value 127 (DEL)

Directory entries, both in the **Root Directory** region and in subdirectories, are of the following format:

Byte Offset	Length	Description
0x00	8	DOS file name (padded with spaces). The first byte can have the following special values: ■ 0x00 Entry is available and no subsequent entry is in use ■ 0x05 Initial character is actually 0xE5. ■ 0x2E 'Dot' entry; either '.' or '..' ■ 0xE5 Entry has been previously erased and is available.
0x08	3	DOS file extension (padded with spaces)
0x0b	1	File Attributes **Bit** **Mask** **Description** ■ 0 0x01 Read Only ■ 1 0x02 Hidden ■ 2 0x04 System ■ 3 0x08 Volume Label ■ 4 0x10 Subdirectory ■ 5 0x20 Archive ■ 6 0x40 Device (internal use only, never found on disk) ■ 7 0x80 Unused An attribute value of 0x0F is used to designate a long file name entry.
0x0c	1	Reserved
0x0d	1	Create time, fine resolution: 10ms units, values from 0 to 199.
0x0e	2	Create time. The hour, minute and second are encoded according to the following bitmap: **Bits** **Description** ■ 15-11 Hours (0-23) ■ 10-5 Minutes (0-59) ■ 4-0 Seconds/2 (0-29) Note that the *seconds* is recorded only to a 2 second resolution. Finer resolution for file creation is found at offset 0x0d.
0x10	2	Create date. The year, month and day are encoded according to the following bitmap: **Bits** **Description** ■ 15-9 Year (0 = 1980, 127 = 2107) ■ 8-5 Month (1 = January, 12 = December) ■ 4-0 Day (1 - 31)
0x12	2	Last access date; see offset 0x10 for description.
0x14	2	EA-Index in FAT12 and FAT16
0x16	2	Last modified time; see offset 0x0e for description.
0x18	2	Last modified date; see offset 0x10 for description.
0x1a	2	First cluster in FAT12 and FAT16. Entries with the Volume Label flag, subdirectory ".." pointing to root, and empty files with size 0 should have first cluster 0.
0x1c	4	File size in bytes. Entries with the Volume Label or Subdirectory flag set should have a size of 0.

Clusters are numbered from a cluster offset as defined above and the FilestartCluster is in 0x1a. This means the first data segment can be calculated:

FileStartSector = reservedSectors + (noofFAT * sectors2FAT) +
(maxRootEntry * 32 / bytes2sector) +
((FileStartCluster − 2) * sectors2cluster)

3.4.5 Position of the Structures in a DOS Partition

- The position of the **Boot Sector** P_{BS}(the beginning of a logical partition) is directly given in the **Master Boot Record** or in an **Extended Master Boot Record**
- The position of the first **FAT** P_{FAT1} is equal to the position of the **boot sector** plus the number of reserved sector:
  $P_{FAT1} = P_{BS} + RES$
- The position of the second **FAT** P_{FAT2} is equal to the position of the P_{FAT1} plus the size of the **FAT**:
  $P_{FAT2} = P_{FAT1} + SPF$
- The position of the **Root Directory** P_{RD} is equal to the position of P_{FAT2} plus the size of the **FAT**:
  $P_{RD} = P_{FAT2} + SPF$
- The position of the first data cluster P_{DATA} is equal to the position of the **Root Directory** plus the size of the **Root Directory**:
  $P_{DATA} = P_{RD} + NDIRS * (32/512)$

The Size of the data region = Number of Clusters * Sectors per Cluster

3.5 The Boot Sequence

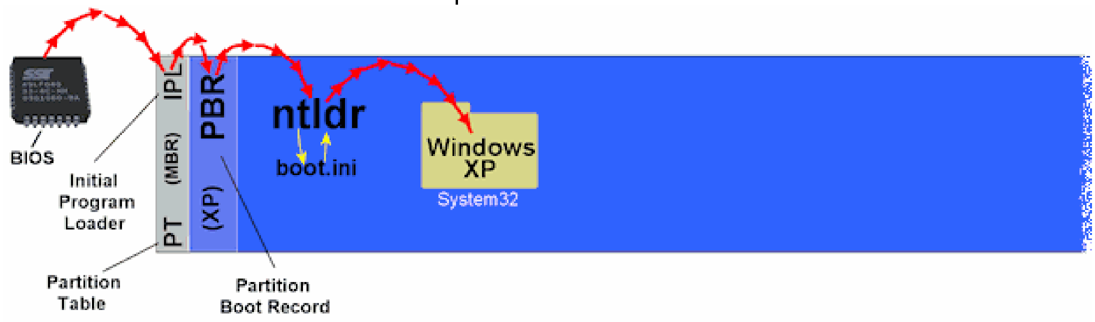
The first program executed in the boot sequence is built into your computer's motherboard. This program is called the BIOS. The BIOS search for the next program to execute. It will look in the place you want it to – Floppy, CD, hard drive, etc.

If you are booting an OS on the hard drive then the next program is, unsurprisingly, on the hard drive. It is always right at the very beginning of the drive, starting on the very first byte of the very first sector. This program is commonly called the MBR (Master Boot Record), but this is a little misleading because the MBR is actually the boot-program and the partition table. The most common name for the boot-program part of the MBR is the **Initial Program Loader** (IPL). Just like the BIOS program the IPL is not usually specific to any OS. The Microsoft IPL is very small and quite limited and its main job is just to find and start that next program in the chain. It looks at the partition table to see if any of the entries there has a flag to indicate an active partition. If it found one then it goes to the very first byte of that partition and starts the little program that it finds there.

The third little program in the chain is at the very beginning of the partition. This one is called the **Partition Boot Record** (PBR). Now the PBR will do its job and then start the next program. However, unlike the BIOS and IPL, the PBR is operating system specific and needs to know the name and location of the file it has to start. This next file will be different for various operating systems, so during the install of an OS the PBR will be written with the information necessary to find the correct file. For WinNT before Vista this will be **ntldr**, which will always just be in the root of the partition. That is it will not be inside any folder or directory, but just right there on its own, next to the Windows and Program Files folders.

For all WinNT before Vista the **ntldr** will be the 4th and last program in the boot sequence chain. It's called the boot-loader and it is the one that does the actual job of starting Windows from the System32 folder.

For Win 2K/XP etc the boot sequence is: - BIOS - IPL - PBR - ntldr - Windows



In these graphic the MBR is shown as a separate section at the very start of the hard drive. It is indeed separate and not connected in any way to the following partitions. Convention is to reserve a small section of the drive specifically for the MBR to reside on. I've shown the PBR as a separate section but it is actually a part of the partition it is in. Windows reserves the first 16 sectors of its partition to be used exclusively for the partition boot record.

4 Atari Hard Disk Partitioning Tests

In this chapter we will examine in detail some examples of hard disk partitioning usable on the Atari platform. We will introduce a hybrid type of partition called the TOS & DOS partitions. And we will look at the compatibilities and limitations of the different types of partitioning.

For these tests I have used several Atari Hard Disk drivers packages: HDDRIVER 8.23, HDDRIVER 7.80, ICD AdSCSI 6.5.5, CBHD 5.02, SCSI Tools 6.5.2, and PPTOSDOS 0.9…

4.1 TOS Partitions

For each of the hard disk driver packages I have first used the partitioning/initialization utility provided to verify that the content of partitions produced follow the AHDI 3.0 specification.

Some of the partitioning programs are easier to use than others to use (better GUI and more options). For a detail description of the partitioning procedures using the different Atari hard disk packages please refer to my document: **UltraSatan Partitioning Guide**.

4.1.1 Partitioning Example Using HDDRIVER 8.23

The results displayed in this section have been obtained using the HDDRIVER V8.23 commercial package. In the following example I have partitioned a 2GB SD Card plugged into an UltraSatan drive into six 300 MB TOS partitions. With the partitioning defined above the following values should be set automatically during initialization to : Logical Sector Size = 8192, Sector Per Cluster = 2, Files in the root directory = 256

4.1.1.1 Analysis of the Root Sector and the Extended Root Sectors

After partitioning we examine the content of the SD card on a PC using a disk editor like **WinHex** or **HxD**. The **Root Sector** at physical location 0 contains:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
...
000001B0   00 00 00 00 00 00 00 00  00 00 00 00 CF 31 00 00   ............Ï1..
000001C0   00 00 00 3A A0 00 01 42  47 4D 00 00 00 02 00 09   ...: ..BGM......
000001D0   60 00 01 42 47 4D 00 09  60 02 00 09 60 00 01 42   `..BGM..`...`..B
000001E0   47 4D 00 12 C0 02 00 09  60 00 01 58 47 4D 00 1C   GM..À...`..XGM..
000001F0   20 02 00 1C 20 00 00 00  00 01 00 00 00 01 00 00    ... ...........
```

It can be interpreted as:

- At offset $01C2 = (00 3A A0 00): The hard disk size is 3 842 048 sectors or 1 967 128 576 bytes
- The first partition structure is located at offset $1C6
  - $1C6 Flag = 1 existing partition
  - $1C7 Id = BGM big partition
  - $1CA partition starting at physical sector 2 (00 00 00 02)
  - $1CE partition size of 614400 (00 09 60 00) physical sectors or 314 572 800 bytes
- The second partition structure is located at offset $1D2
  - $1D2 Flag = 1 existing partition
  - $1D3 Id = BGM partition
  - $1D6 partition starting at physical sector 614402 (00 09 60 02)
  - $1DA partition size of 614400 (00 09 60 00) physical sectors
- The third partition structure is located at offset $1DE
  - $1DE Flag = 1 existing partition
  - $1DF Id = BGM partition
  - $1E2 partition starting at physical sector 1228802 (00 12 C0 02)
  - $1E6 partition size of 614400 (00 09 60 00) physical sectors
- The fourth partition (extended) structure is located at offset $1D2
  - $1EA Flag = 1 existing partition
  - $1EB Id = XGM partition
  - $1EE partition starting at physical sector 1843202 (1C 20 02 00)
  - $1F2 not meaningful
- at offset $1F6 (00 00 00 01): The Bad sector list offset is equal to 1
- at offset $1FA (00 00 00 01): The Bad Sector count is equal to 1

As defined in the **Root Sector** we have an *extended partition* with an **Extended Root Sector** located at sector 1843202.

At this location the first **Extended Root Sector** contains:

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
38400400  00 00 00 52 4F 4F 54 58  00 00 00 00 00 00 00 00   ...ROOTX........
...
384005B0  00 00 00 00 00 00 00 00  00 00 00 00 BC ED 00 00   ............¼í..
384005C0  00 00 00 00 00 00 01 42  47 4D 00 00 00 01 00 09   .......BGM......
384005D0  5F FF 01 58 47 4D 00 09  60 00 00 09 60 00 00 00   _ÿ.XGM..`...`...
384005E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
384005F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

- The first partition entry is located at offset $384005C6
  - Flag = 1 existing partition
  - Id = BGM big partition
  - partition starting at physical sector 1 (00 00 00 01)
  - partition size of 614399 (00 09 5F FF) physical sectors = 314 572 288 bytes
- The second partition entry
  - Flag = 1 existing partition
  - Id = XGM partition
  - Next extended partition starting at physical sector 614400 (00 09 60 00)
  - not meaningful

Note that the position of the partition and the next ERS are given relative to the extended partition.

As defined in the first **Extended Root Sector** we have another partition starting with another **Extended Root Sector** located at sector 2457602 (1843202 + 614399)

At this location the second **Extended Root Sector** contains:

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
4B000400  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
...
4B0005B0  00 00 00 00 00 00 00 00  00 00 00 00 00 DD 00 00   .............Ý..
4B0005C0  00 00 00 00 00 00 01 42  47 4D 00 00 00 01 00 09   .......BGM......
4B0005D0  5F FF 01 58 47 4D 00 12  C0 00 00 09 60 00 00 00   _ÿ.XGM..À...`...
4B0005E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
4B0005F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

- The first partition structure is located at offset $4B0005C6
  - Flag = 1 existing partition
  - Id = BGM big partition
  - partition starting at physical sector 1 (00 00 00 01)
  - partition size of 614399 (00 09 5F FF) physical sectors = 314 572 288 bytes
- The second partition structure
  - Flag = 1 existing partition
  - Id = XGM partition
  - Next extended partition starting at physical sector 1228800 (00 12 C0 00)
  - not meaningful

As defined in the second **Extended Root Sector** we have another partition starting with another **Extended Root Sector** located at sector 3072002 (1843202 + 1228800)

At this location the third **Extended Root Sector** contains:

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
5DC00400  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
...
5DC005B0  00 00 00 00 00 00 00 00  00 00 00 00 69 9D 00 00   ............i□..
5DC005C0  00 00 00 00 00 00 01 42  47 4D 00 00 00 01 00 09   .......BGM......
5DC005D0  5F FF 00 00 00 00 00 00  00 00 00 00 00 00 00 00   _ÿ..............
5DC005E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
5DC005F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

- The first partition structure is located at offset $5DC005C6
  - Flag = 1 existing partition
  - Id = BGM big partition
  - partition starting at physical sector 1 (00 00 00 01)
  - partition size of 614399 (00 09 5F FF) physical sectors = 314 572 288 bytes
- All the fields in the second partition structure are null indicating no more partitions.

4.1.1.2 Analysis of the Boot Sector

At the beginning of each standard partition (logical sector 0) we first find a **Boot Sector**. For example if we look at the boot sector of the first partition located at physical sector 2 we have:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000400   E9 00 90 4D 53 44 4F 53  5A 82 E6 00 20 02 01 00   é.□MSDOSZ,æ. ...
00000410   02 00 01 00 96 F8 05 00  00 00 00 00 02 00 00 00   ....–ø.........
00000420   00 00 00 00 00 00 29 E6  00 86 12 4E 4F 20 4E 41   ......)æ.†.NO NA
00000430   4D 45 20 20 20 20 46 41  54 31 36 20 20 20 00 00   ME    FAT16   ..
00000440   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
...
```

This **Boot Sector** can be interpreted as follow:

```
$0B  BPS     8192    16 Phys sectors
$0D  SPC     2       1 cluster = 16384 bytes
$0E  RES     1       1 logical sector = 18 Phys sect
$10  NFATS   2       2 FATs
$11  NDIRS   256     256 Directory entries
$13  NSECTS  38400   38400 * 8192 = 314 572 800 bytes
$15  MEDIA   $F8     Hard Disk
$16  SPF     5       5 logical sector = 80 phys sect
```

The **Boot Sector** plus the reserved sectors are immediately followed by the two **FATs**, a **Root Directory**, and the **Data**. The location of the different regions can be computed from **BPB** information located in the **Root Sector.** For example for the first partition:

- **Boot Sector** starts at sector 2 as specified in the **Root Sector**
- First **FAT** starts a sector 18 = (2 + 1x16),
- Second **FAT** starts a sector 98 = 18 + 5*16
- **Root Directory** starts at sector 178 = 96 + 5*16
- Data starts at sector 194 = 178 + 256*32/512

The same analysis could be done for all the partitions (regular or extended).

4.1.2 Atari Bootable TOS Partitions

As we have seen in The Boot Sequence section it is possible to render any of the TOS primary partitions bootable on an Atari. This results in loading the hard disk driver in memory from the HD.

The procedures to render a TOS partition bootable differ for every package but use the same mechanism. It results in writing some **IPL** code in the in the **Root Sector** as well as some boot-loader code (PBL) in the **Boot Sector** of the chosen partition. The code in the boot sector can either load the rest of its code from a file (for example **HDDRIVER.SYS**) or from sectors in the reserved region (for example SCSI Tools).

4.1.3 Accessing the TOS Partitions

With all the hard disk drivers I was able to access the TOS partitions created by any other packages. The maximum size of the partitions follows the AHDI specification as described in the TOS Partition Size section. However it is interesting to note that with the SCSI Tools 6.5.2 and AHDI 6.0.6.1 packages the maximum size of the boot partition is limited to 32MB (16MB for TOS < 1.04).

4.1.4 Summary of the tests with TOS partitions

- All the TOS partitioning tools strictly follow the AHDI specification and therefore you should not have any problem using a TOS partitioned drive with any Atari AHDI compliant hard disk driver. For example it is possible to read and write the partitions created with the **HDDRIVER** partitioning utility using the **ICD** hard disk driver.
- It is possible to render any of the TOS primary partitions bootable on the Atari. For example it is possible with the HDUTIL.PRG from ICD to install the ICD boot loader onto a partition of a drive partitioned with HDDRIVER.
- Bootable TOS partition is limited to 32MB with SCSI TOOLS and AHDI packages.
- All TOS Partitions works correctly with a maximum size of:
  - up to 256MB with TOS 1.0 & 1.02,
  - up to 512MB for TOS ≥ 1.04
  - up to 1GB with TOS ≥ 4.0
- Of course the partitioned disk cannot be accessed directly on a PC running DOS/Windows (although some specific PC applications can read TOS partitions on a PC).

4.2 TOS & DOS Partitions

Two Atari hard disk drivers (namely PPTOSDOS and HDDRIVER) use a hybrid type of partition called TOS & DOS partition. This technique allows creating partitions on a drive that can be seen by PC DOS-Windows computers as a DOS/FAT partition and by Atari computers as a TOS partition. The HDDRIVER and PPTOSDOS packages use similar technique but different implementation and therefore the two solutions are **not compatible**. For each TOS & DOS partition two boot sectors are written in the partition: one for the DOS file system and one for the TOS file system. The maximum size of a TOS & DOS partitions follows the TOS file system limitation of 512MB (for TOS ≥ 1.04).

As the TOS & DOS partition are accessible on both platforms, can be made bootable, and can have a relatively large size (512MB) they are very well suited for data transfer between Atari and PC computers (for example using SD cards plugged into Satan or UltraSatan Drives).

4.2.1 Partitioning Example using PPTOSDOS

The PPTOSDOS hard disk driver allows creating multiple DOS & TOS partition on a drive.

For the test I have used a 1GB SD Card plugged in an UltraSatan drive and I have partitioned the drive into three 300MB partition using the PPTOSDOS TOS & DOS compatibility mode.

4.2.1.1 Analysis of the Partition from a DOS point of view

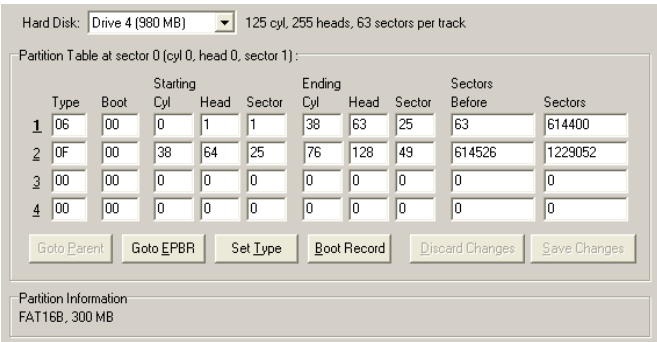
We now examine the content of the SD card on a PC using the disk **WinHex** editor.

The MBR format is follow exactly the standard PC format:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
...
000001A0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000001B0   00 00 00 00 00 00 00 00  7D B1 86 57 1C 57 00 01   ........}±†W.W..
000001C0   01 00 06 3F 19 26 3F 00  00 00 00 60 09 00 00 40   ...?.&?....`...@
000001D0   19 26 0F 80 31 4C 7E 60  09 00 FC C0 12 00 00 00   .&.€1L~`..üÀ....
000001E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA   ..............Uª
```

Relevant information in the partition table:

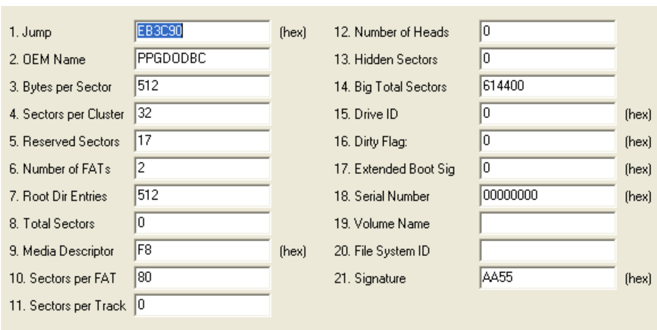
- The first partition entry is located at offset $1BE
  - $1BE State = 00 (non bootable)
  - $1BF Starting CHS = 0, 1, 1
  - $1C2 Type = 06 (FAT16B with size > 32MB)
  - $1C3 Ending CHS = 38, 63, 25
  - $1C6 LBA of first sector = 63
  - 1CA Size of sector = 614400
- The second partition entry is located at offset $1DE
  - $1DE State = 00 (non bootable)
  - $1DF Starting CHS = 38, 64, 25
  - $1C2 Type = 0F (FAT16B with size > 32MB)
  - $1C3 Ending CHS = 76, 128, 49
  - $1E6 partition size 1229052
- The third and fourth partition entries are empty



To get details information on PC I use the PowerQuest Partition Table Editor 1.0 program.

As we can see the first partition entry is declared as a **FAT16B** partition (type=06) starting at sector 63 with 1012032 sectors.

Now if we look at the **Boot sector** for the first DOS partition (at sector 63) we find the following values:



- BPS = 512
- SPC = 32
- Reserved = 17
- NSECTS = 0
- HSECTS = 614400

The next partition entry is an extended partition. We will look at the detail here it as this will be done in the section on DOS/FAT partitions. However it is interesting to note that the extended partition is specified by using a file type=0F (instead of type=05). This is done on purpose to force the usage of LBA addressing instead of CHS addressing. See also FAT Partition Type and Size paragraph.

As we can see these values are all correct for DOS file system to access the partition.

The **boot sector** plus the reserved sectors are immediately followed by the two **FATs**, a **root directory**, and the **data**.

The location of the different control structures can be computed from this DOS **BPB**:

- **Boot Sector** starts at sector 63 as specified in the **MBR**
- first **FAT** starts a sector 80 = (63 + 17),
- Second **FAT** starts a sector 160 = 80 + 80
- **Root Directory** starts at sector 240 = 160 + 80
- **Data** starts at sector 272 = 240 + 512*32/512

4.2.1.2 Analysis of the Partition from a TOS point of View

If we now look at sector 64 (the very next sector after the DOS Boot Sector specified in the partition table) we find a hidden TOS boot sector. Now let's analyze with **WinHex** the content of this sector:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00008000   EB 3C 90 50 50 47 44 4F  44 42 43 00 20 02 01 00   ë<□PPGDODBC. ...
00008010   02 00 02 06 95 F8 05 00  00 00 00 00 00 00 00 00   ....•ø..........
00008020   60 50 09 00 00 00 00 00  00 00 00 00 00 00 00 00   `P..............
00008030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

The TOS boot sector can be interpreted as:

Name	Offset	Length	Values
BRA	0x00	3	EB 3C 90
OEM	0x03	8	PPGDODBC
BPS	0x0b	2	8192
SPC	0x0d	1	2
RES	0x0e	2	1
NFATS	0x10	1	2
NDIRS	0x11	2	512
NSECTS	0x13	2	38150
MEDIA	0x15	1	F8
SPF	0x16	2	5
SPT	0x18	2	0
NHEADS	0x1a	2	0
NHID	0x1c	4	0
HSECTS	0x20	4	610400

As we can see these values are all correct for TOS file system to access the partition.

The location of the different control structures can be computed from the **BPB**:

- **Boot Sector** starts at sector 64 (computed as 63+1 by the PPTOSDOS driver)
- First **FAT** starts a sector 80 = 64 + 1*16
- Second **FAT** starts a sector 160 = 80 + 5*16
- **Root Directory** starts at sector 240 = 160 + 5*16
- **Data** starts at sector 272 = 240 + 512*32/512

As we can see the two **FATs**, the **Root Directory**, and the **Data** are have the exact same location when interpreted by the FAT file system and the TOS file system.

Therefore both the Atari (when using PPTOSDOS) and the PC DOS/Windows have all the necessary information to access, at the same location, the data correctly.

For test purpose I have filled about 160MB of data on the disk and the data could be accessed on both platforms correctly.

4.2.2 Example using HDDRIVER 8.23

The HDDRIVER solution allows creating one and only one DOS & TOS partition on a drive. This is a severe limitation for large drive. If for example you use a 2GB SD card on an UltraSatan drive you will be able to create one 512MB partition and you will lose the remaining 1250MB!

For the test I have therefore used a 512MB SD Card plugged in an UltraSatan drive. I have partitioned the drive into one 500MB partition using the HDDRIVER TOS & DOS compatibility mode. The sector per track has been set to 63 and the number of heads to 255 (used to compute CHS values).

4.2.2.1 Analysis of the Partition from a DOS point of view

We now examine the content of the SD card on a PC using the disk **WinHex** editor.

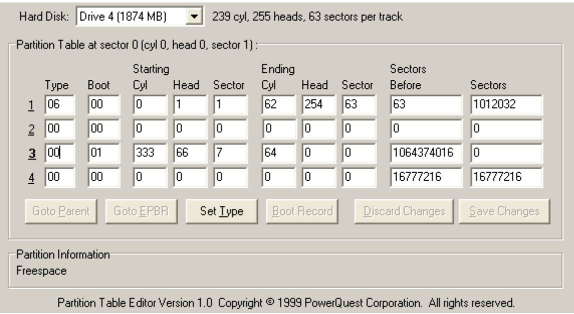
The MBR format is similar to the standard PC format but add a special entry in the partition table:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
...
000001A0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000001B0   00 00 00 00 00 00 00 00  F3 B7 71 F9 82 68 00 01   ........ó·qù‚h..
000001C0   01 00 06 FE 3F 3E 3F 00  00 00 40 71 0F 00 00 00   ...þ?>?...@q....
000001D0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 01 42   ...............B
000001E0   47 4D 00 00 00 40 00 0F  71 3F 00 00 00 00 00 00   GM...@..q?......
000001F0   00 00 00 00 00 00 00 00  00 01 00 00 00 01 55 AA   ..............Uª
```

The relevant information is:

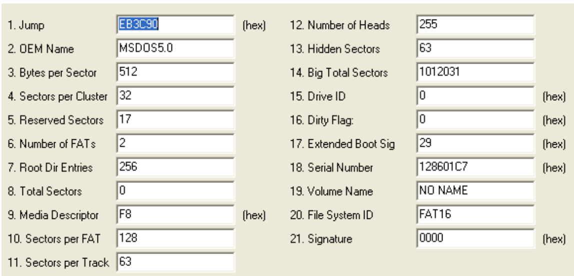
- The first partition entry is located at offset $1BE
  - $1BE State = 00 (non bootable)
  - $1BF Starting CHS = 0, 1, 1
  - $1C2 Type = 06 (FAT16B with size > 32MB)
  - $1C3 Ending CHS = 62, 254, 63
  - $1C6 LBA of first sector = 63 (small indian format)
  - 1CA Size of sector = 1012032 (small indian)
- The second partition entry is located at offset $1CE and is empty
- The third partition entry is located at offset $1DE. It contains a combination of DOS and TOS information and can (probably) be interpreted as:
  - $1DE Flag = 1 existing partition (for TOS)
  - $1DF Id = BGM partition (for TOS)
  - $1E2 Type = 0 empty partition (for DOS to ignore this partition)
  - $1E2 Starting address 64 (00 00 00 40) in big indian format (for TOS)
  - $1E6 partition size 1012031 (00 0F 71 3F) sectors in big indian (for TOS)
- The fourth partition entry is located at offset $1EE and is empty

To get details information on PC I use the PowerQuest Partition Table Editor 1.0 program.



As we can see the first partition is declared as a **FAT16B** partition (type=06) starting at sector 63 with 1012032 sectors. The third partition is an empty partition for DOS (type=00) but with some nonsense values in it. This partition is ignored by DOS/Windows as the partition is declared as type=00. This is the hidden partition that will be used by the TOS file system.

Now if we look at the **Boot sector** for the DOS partition (at sector 63) we find the following values:



- BPS = 512
- SPC = 32
- Reserved = 17
- NSECTS = 0
- HSECTS = 1012031

These values are all correct for DOS file system to access the partition.

The **boot sector** plus the reserved sectors are immediately followed by the two **FATs**, a **root directory**, and the **data**.

The location of the different regions can be computed from the **BPB** located in the **root sector**:

- **Boot Sector** starts at sector 63 as specified in the **Root Sector**
- first **FAT** starts a sector 80 = 63 + 17
- Second **FAT** starts a sector 208 = 80 + 128
- **Root Directory** starts at sector 336 = 208 + 128
- **Data** starts at sector 352 = 336 + 256*32/512

4.2.2.2 Analysis of the Partition from a TOS point of View

Now let's analyze with **WinHex** the content of sector 64: the TOS "hidden partition" as specified in the **MBR**. Here we can find a standard TOS **Boot Sector**:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00008000   E9 00 90 4D 53 44 4F 53  5A 82 E6 00 20 02 01 00   é.□MSDOSZ,æ. ...
00008010   02 00 01 13 F7 F8 08 00  00 00 00 00 40 00 00 00   ....÷ø......@...
00008020   00 00 00 00 00 00 29 23  00 86 12 4E 4F 20 4E 41   ......)#.†.NO NA
00008030   4D 45 20 20 20 20 46 41  54 31 36 20 20 20 00 00   ME    FAT16   ..
00008040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

This **boot sector** can be interpreted as follow:

Name	Offset	Length	Value
BRA	0x00	3	E9 00 90
OEM	0x03	8	MSDOSZ
BPS	0x0b	2	8192 → 16 Phys sectors
SPC	0x0d	1	2 →1 logical sector = 18 Phys sect
RES	0x0e	2	1 logical sector = 18 Phys sect
NFATS	0x10	1	2 FATs
NDIRS	0x11	2	256 Directory entries
NSECTS	0x13	2	63251 → 63251* 8192 = 518 152 192 bytes
MEDIA	0x15	1	F8 → Hard Disk
SPF	0x16	2	8 logical sector = 128 phys sect

These values are all correct for TOS file system to access the partition.

The **Boot Sector** plus the reserved sectors are immediately followed by the two **FATs**, a **Root Directory**, and the **Data**.

The location of the different regions can be computed from **BPB** located in the **root sector**:

- **Boot Sector** starts at sector 64 as specified in the **Root Sector**
- First **FAT** starts a sector 80 = 64 + 1x16
- Second **FAT** starts a sector 208 = 80 + 8*16
- **Root Directory** starts at sector 336 = 208 + 8*16
- **Data** starts at sector 352 = 336 + 256*32/512

As we can see the two **FATs**, the **Root Directory**, and the **Data** are have the exact same location when interpreted by the FAT file system (partition 0) and the TOS file system (hidden partition 3).

Therefore both the Atari (using HDDRIVER) and the PC DOS/Windows have all the necessary information to access, at the same location, the data correctly.

For test purpose I have filled about 160MB of data on the disk and the data could be accessed on both platforms correctly.

4.2.3 Accessing TOS & DOS partition with other Atari hard disk driver

I have tried to access TOS & DOS partitioned drive with the ICD driver (results have been similar with any driver other than HDDRIVER and PPTOSDOS). All the files are correctly displayed in the Atari disk browser and I was able to read some of the files located at the beginning of the partition without problem. However when I tried to access some of the files located at the end of the partition (beyond the 32MB limit), the return data was **totally incorrect** and writing would corrupt the partition.

This behavior can easily be explained: The ICD driver is obviously not aware of the "hidden" TOS boot sector used by either the PPDOSTOS or by the HDDRIVER drivers. The ICD driver only sees the DOS boot sector and therefore only interprets the DOS BPB. As we will see in the next section DOS partitions when used on an Atari, with TOS and GEMDOS, are limited to 32MB and this explain why we get incorrect result if we try to access data beyond this limit.

4.2.4 Accessing Mixture of DOS and TOS & DOS partitions on Atari

As we have seen the HDDRIVER only allow only one TOS & DOS partition and therefore do not allow the mixture of DOS and TOS & DOS partitions. However PPTOSDOS allow using a mixture of DOS and TOS & DOS partitions. But in order to avoid the problem of DOS partitions larger than 32MB (that we will see in the next section) the “pure” DOS partitions are filtered by the PPTOSDOS and are therefore only seen on a PC.

4.2.5 Atari Bootable TOS & DOS Partition

It is possible to render a DOS & TOS partition bootable by using the driver’s specific utility (provided with the PPDOSTOS or HDDRIVER packages). However it is not possible to render this partition bootable with any other hard disk driver utility. Therefore a DOS & TOS bootable partition **should only be created and accessed** by using the specific tools provided with specific hard disk driver package.

4.2.6 Summary of the test with TOS & DOS Partition

- The PPTOSDOS can create **many** medium partitions (up to 512 MB for TOS ≥ 1.04) that can be accessed correctly on both the Atari and the PC. These partitions can be used to transfer data between the Atari and the PC.
- The HDDRIVER can create **one and only one** medium partition (up to 512 MB for TOS ≥ 1.04) that can be accessed correctly on both the Atari and the PC. This partition can be used to transfer data between the Atari and the PC.
- Any other hard disk drivers **are not capable** to use correctly the TOS & DOS partition created by PPTOSDOS or HDDRIVER.
- PPTOSDOS is the only hard disk driver that allows a mixture of DOS and DOS&TOS partitions on a single drive. However to avoid the problem of the 32MB DOS partition limit the DOS/FAT partitions are not seen on the Atari platform.
- A TOS & DOS partition can be made bootable using the appropriate utility.

4.3 DOS/FAT Partitions

Several Atari hard disk packages can directly access DOS/FAT partitions. But we will see, in this section, that accessing DOS partitions is only possible with some limitations. Also note that most of the Atari hard disk utilities do not provide any capability to create DOS partitions. HDDRIVER and PPTOSDOS are the only two exceptions.

4.3.1 Problems with DOS Partition

We first try to partition a 2GB SD card plugged into an UltraSatan drive into six 300MB DOS partitions using the HDDRIVER utility.

We use the ***Partition*** command of HDUTIL program to partition the drive:

- The size for each of the 6 partitions is set to 300MB (leaving 76MB of free space).
- The compatibility mode is set to DOS. In this mode we also have to specify the *Sector per Track* and the *Number of Heads*. This information is used to compute the **CHS** values in the Partition table entries of the MBR. Although, in modern PCs, these values are not used anymore it is recommended to set them to their maximum values (see HD background information). Therefore we set SPT = 63 and NHEADS = 255.

Then we use the ***Initialize Partition*** command[7] on all the partitions. During initialization the program asks for the *Logical Sector Size* as well as the *Sector per Cluster*. We set the LSS to the minimum value that allow access the complete 300MB partition. Therefore we set LSS to 8192 (8192 * 65536 > 300MB) and we set the SPC to 2 as this is the only value recognized by GEMDOS.

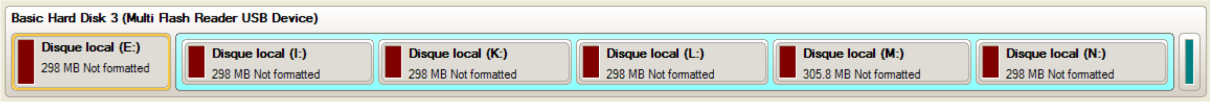
After these operations all the DOS partitions are correctly accessible on the Atari with the HDDRIVER without limitations.

Now we plug the SD card to a card reader on the PC. The partitions are recognized by Windows[8] but they are not accessible (they are reported as not formatted).

[7] It is mandatory to reboot the system before **each** initialize command execution!
[8] Accessing multiple partitions on Windows requires a special hard disk driver for the card reader

4.3.1.1 Analysis of the Partition Layout and MBR

We first look at the HD layout with an application like Paragon Partition Manager:

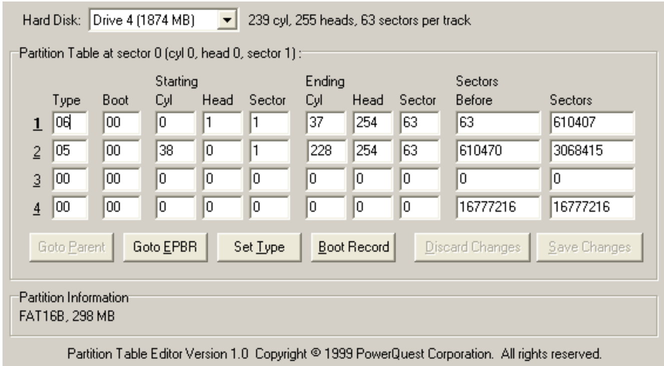


As we can see there is only one *Primary Partition* and five *Secondary Partitions* located in an *Extended Partition*.

If we examine the MBR we find that it correctly follows the DOS/FAT scheme.

To display detailed information I have used the PowerQuest Partition Table Editor 1.0 program.

As we can see the first partition is declared as a type=06 (**FAT16B)** partition starting at sector 63. The next partition is a type=05 (**Extended FAT16B)** partition starting at sector 610470. If we look at the next EPRB we find the same kind of structure: one type=06 partition and a type=05 link to the next EPRB…



4.3.1.2 Analysis of the Problems with the Boot Sector

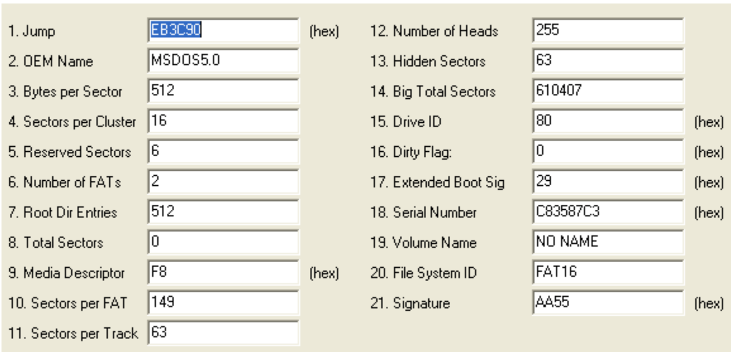
We now look at the values in the **Boot Sector**. As we can see the number of bytes per sector is equal to 8192 and the Sectors per Cluster is equal to 2 (this is what we have defined during initialization). We also see that the computed number of sectors is equal to 38150 and that huge sector number = 0.

While all these values make sense in the Atari environment they are totally unacceptable on a DOS system because the only supported number of bytes per sector is 512.

This explain why these partitions are accessible on an Atari, but not accessible on a PC.

Therefore I have repartitioned the drive on the PC as six 300MB partitions using FAT16 and the default values.

Now the content **Boot Sector** is changed to: BPS is set to 512, and SPC is set to 16 (16 * 512 = 8192 bytes per logical sector). The total number of sectors is 610407. This value is written in the HSECS field (as the number of sectors is > 65536) and the NSECTS is set to 0. Other values are also modified but they are not relevant in this discussion.



With these BPB values we can now access correctly the partition under DOS/Windows. However if we try to use this card on the Atari none of the hard disk drivers are capable of accessing the drive! This is mainly due to the fact that the SPC is now equal to16 and this value is not supported by the GEMDOS.

Therefore we can see that we have identified two constraints for DOS partitions:

- On Atari the SPC has to be set to 2,
- And on PC-DOS the BPS has to be set to 512.

Consequently if we want to create partitions that can be accessed correctly on **both platforms** we have to follow these two constrained values at the **same** time. This imply a logical sector size of 1024 (512 * 2) and as a result the maximum partition size is now 32MB (65536 * 512).

We will see later a solution to overcome this limitation.

4.3.2 Accessing Small DOS/FAT Partitions on Atari

Based on the constraints identified above we can now define DOS/FAT partitions that can be accessed on DOS/Windows platform as well as on Atari platform.

As an example we create the several partitions using the Windows Disk Management console under Windows XP. For the test I have created on an SD card three primary partitions each with a size of 31MB and one extended partition of size 31MB. The partitions are created using: FAT for the File system, and 1204 for the Allocation unit size (do not use default).

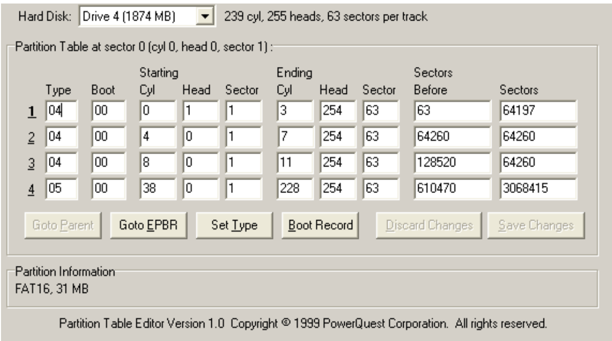
> *Note: In order to be able to create multiple partitions on an SD card, plugged into a PC card reader, you need to install a special card reader driver (like the **Hitachi Microfilter**). This allows seeing the PC card reader as a hard disk. The procedure is not describe here but you can refer to my document **UltraSatan Partitioning Guide** (see references)*

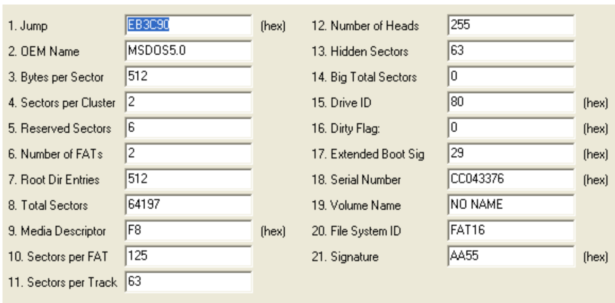
4.3.2.1 Analysis of the Master Boot Record and Boot Sector

Now if we examine the drive with PowerQuest Partition Table Editor we can see that three primary partitions of type=04 (**FAT16A** with a max size < 32 MB) have been created.

And if we look at the corresponding **Boot Sectors** they all have a BPS of 512 and a SPC of 2:

With the special hard disk driver for the SD card reader we can access the three "small" partitions on the PC.





These same three partitions can be accessed correctly on the Atari using either the HDDRIVER or the ICD hard disk drivers.

Therefore we see that it is possible to create "small" partitions of size ≤ 32MB that can be used on DOS/Windows as well as on Atari (granted that the Atari hard disk driver support DOS partitions). These small type=04 DOS partitions can be used to transfer data between the PC and the Atari.

4.3.3 Accessing Large DOS/FAT Partitions on Atari

By large DOS partition I mean partitions with a size ≥ 32MB. These partitions are referred as:

- Type $06 or $0E (aka **FAT-16B**) with a size in the range 32MB – 2GB
- Type $05 or $0F (aka **Extended FAT-16B**) with a size in the range 32MB – 2GB

> *Remember also that in order to access data beyond 1GB you need to have a HD driver that support the ICD extended command set (SCSI Group 1) as well as a host adapter that also support this extended command set. For example an UltraSatan disk drive.*

As we have seen due to the constraints imposed by the TOS file systems and the DOS file systems it seems that it is only possible to access Small (**FAT12** and **FAT16A** ≤ 32 MB) DOS/TOS partitions with an Atari.

> ***Warning**: Beware that FAT16B and even FAT32 partitions are recognized by many Atari hard disk drivers and therefore on the surface they look fine: Partitions seems to be accessible and even report correct size. However when you try to access data beyond 32MB the driver returns **incorrect** values. Even worse if you write beyond this 32MB limit the driver writes the data at the beginning of the partition resulting in a **totally corrupted partition**.*

The **BIGDOS** freeware allows access to Large DOS partitions with some restriction (for more information please read the BIGDOS documentation). Most of the problems (for example the fix value of SPC=2) comes from some code inside GEMDOS. BIGDOS replaces GEMDOS at boot time and removes some of its limitations. More specifically it allows the support of SPC values of up to 64, and uses of the HSECTS parameter (32-bit number of sectors) instead of the NSECTS parameter (16-bit number of sectors). This allows more than 65536 sectors and therefore removes the 32MB limitation.

BIGDOS supports many large DOS/Fat partitions on the same drive. For example you can partition a 2GB drive into four 512MB partitions.

To work with BIGDOS you need to use a hard disk driver that complies with XHDI 1.20 (or above). BIGDOS has been tested successfully by the author with HDDRIVER version 4.51 (or above) and with CBHD version 4.5 (or above). In practice it works with many hard disk drivers, but unfortunately it does not work with some others like the ICD AdSCSI 6.5.5 hard disk driver.

With BIGDOS loaded, I have tested a 2 GB FAT16B partition. All the hard disk drivers that support BIGDOS where able to access correctly the information on these partitions and do not exhibit the 32MB problem explained above. Note that these 2GB partitions where used for test purpose, but such big partitions are not recommended for performance reason.

When using BIGDOS it is recommended to partition the drive on a PC. For example you can use the Windows Disk Management tool. In that case specify the type FAT for the file system and use the default parameter for allocation unit size. For more information on the exact procedure please refer to my document **UltraSatan Partitioning Guide**.

There are other solutions to access large DOS partitions on an Atari (for example by using Mint) but they are not covered here.

4.3.4 Accessing Huge DOS/FAT Partitions on Atari

By Huge DOS partition I mean partitions with a size ≥ 2GB. These partitions are referred as:

- Type $0B (aka **FAT32**) with a size in the range 512MB – 2TB
- Type $0C (aka **Extended FAT32**) with a size in the range 512MB – 2GB

I have not been able to access huge DOS/FAT partitions, with any of the Atari HD drivers that I have tried, on Atari. This is true even if BIGDOS is loaded.

There are some solutions to access huge DOS partitions on an Atari (for example by using Mint) but they are not covered here.

4.3.5 Atari Bootable DOS/FAT Partitions

I was not able to render a DOS/FAT partition bootable with any of the Atari HD drivers that I have tried.

4.3.6 Creating DOS Partitions with HDDRIVER

HDDRIVER is supposed to be able to create DOS partitions. However I have found the following problems (using v8.23):

- For FAT16B partitions: The number of hidden sector (at BPB $1C) is set incorrectly for partition other than first. HSEC is not used very often and therefore it might not be a problem?
- For FAT32 partitions: The Extended BPB has totally wrong values.

4.3.7 Summary of the Tests with DOS partitions

- Many hard disk drivers can access “Small” DOS/FAT partition (FAT16A with size ≤ 32MB).
- It is possible to partition a drive with many small DOS partitions and to access all these partitions on the Atari and the PC (requires an appropriate card reader driver on PC).
- Large DOS partitions (size > 32MB) created on the PC are not directly compatible with Atari hard disk drivers. These partitions are therefore either not accessible or incorrectly accessed in the Atari environment.
- However it is possible to access correctly large DOS partitions (FAT16B) by loading at boot time the BIGDOS replacement of GEMDOS. But in order to use BIGDOS the hard disk driver need to follow XHDI 1.2 (or above).
- Huge DOS partitions (size ≥ 2GB) created on the PC are not accessible on Atari.
- To create one or several large DOS/FAT partitions on a dive, it is recommended to create them on a PC using for example the Windows Disk Management tool. Please refer to my document **UltraSatan Partitioning Guide** for detail procedure.

5 References

Master boot record from Wikipedia
CHS conversion from Wikipedia
Partition types - Andries Brouwer
Disk partitioning from Wikipedia
File Allocation Table From Wikipedia, the free encyclopedia
MS-DOS Partitioning Summary – Microsoft 2007
Hard Disk Partitioning Primer - Mikhail Ranish
AHDI 3.0 Release Notes – Atari, April 1980
XHDI 1.30 Specifications – J.F. Reschke, 1999
A Hitchhiker's Guide to the BIOS – Atari 1998, 1989, 1990
MS-DOS Programmer's Reference V5 – Microsoft Press 1991
UltraSatan Partitioning Guide – Jean Louis-Guérin November 2009
Driver for ASCI/SCSI Atari disks – Pera Putnik & Jean Louis-Guerin
Multibooters - Dual and Multibooting with Vista
BIGDOS – Rainer Seitel 2000