Master boot record

This article is about a PC-specific type of boot sector on partitioned media. For the first sector on non-partitioned media, see volume boot record.

A **master boot record** (**MBR**) is a special type of boot sector at the very beginning of partitioned computer mass storage devices like fixed disks or removable drives intended for use with IBM PC-compatible systems and beyond. The concept of MBRs was publicly introduced in 1983 with PC DOS 2.0.

The MBR holds the information on how the partitions, containing file systems, are organized on that medium. The MBR also contains executable code to function as a loader for the installed operating system—usually by passing control over to the loader's second stage, or in conjunction with each partition's volume boot record (VBR). This MBR code is usually referred to as a boot loader.[1]

The organization of the partition table in the MBR limits the maximum addressable storage space of a 512-sector disk to 3.99 TiB ($2^{32} \times 512 + 2^{32} \times 512$ bytes).[2] Therefore, the MBR-based partitioning scheme is in the process of being superseded by the GUID Partition Table (GPT) scheme in new computers. A GPT can coexist with an MBR in order to provide some limited form of backward compatibility for older systems.

MBRs are not present on non-partitioned media such as floppies, superfloppies or other storage devices configured to behave as such.

1 Overview

Support for partitioned media, and thereby the master boot record (MBR), was introduced with IBM PC DOS 2.0 in March 1983 in order to support the 10 MB hard disk of the then-new IBM Personal Computer XT, still using the FAT12 file system. The original version of the MBR was written by David Litton of IBM in June 1982. The partition table supported up to four *primary partitions*, of which DOS could only use one. This did not change when FAT16 was introduced as a new file system with DOS 3.0. Support for an *extended partition*, a special primary partition type used as a container to hold other partitions, was added with DOS 3.2, and nested *logical drives* inside an extended partition came with DOS 3.30. Since MS-DOS, PC DOS, OS/2 and Windows were never enabled to boot off them, the MBR format and boot code remained almost unchanged in functionality, except for in some third-party implementations, throughout the eras of DOS and OS/2 up to 1996.

In 1996, support for logical block addressing (LBA) was introduced in Windows 95B and DOS 7.10 in order to support disks larger than 8 GB. *Disk timestamps* were also introduced, though the description of their actual purpose is not available.[3] This also reflected the idea that the MBR is meant to be operating system and file system independent. However, this design rule was partially compromised in more recent Microsoft implementations of the MBR, which enforce CHS access for FAT16B and FAT32 partition types 06_{hex}/$0B_{hex}$, whereas LBA is used for $0E_{hex}$/$0C_{hex}$.

Despite sometimes poor documentation of certain intrinsic details of the MBR format (which occasionally caused compatibility problems), it has been widely adopted as a de facto industry standard, due to the broad popularity of PC-compatible computers and its semi-static nature over decades. This was even to the extent of being supported by computer operating systems for other platforms. Sometimes this was in addition to other pre-existing or cross-platform standards for bootstrapping and partitioning.[4]

MBR partition entries and the MBR boot code used in commercial operating systems, however, are limited to 32 bits. Therefore, the maximum disk size supported on disks using 512-byte sectors (whether real or emulated) by the MBR partitioning scheme (without using non-standard methods) is limited to 2 TiB. Consequently, a different partitioning scheme must be used for larger disks, as they have become widely available since 2010. The MBR partitioning scheme is therefore in the process of being superseded by the GUID Partition Table (GPT). The official approach does little more than ensuring data integrity by employing a *protective MBR*. Specifically, it does not provide backward compatibility with operating systems that do not support the GPT scheme as well. In the meanwhile, multiple forms of *hybrid MBRs* have been designed and implemented by third parties in order to maintain partitions located in the first physical 2 TiB of a disk in both partitioning schemes "in parallel" and/or to allow older operating systems to boot off GPT partitions as well. The present non-standard nature of these solutions causes various compatibility problems in certain scenarios.

The MBR consists of 512 or more bytes located in the first sector of the drive.

It may contain one or more of:

- A partition table describing the partitions of a storage device. In this context the boot sector may also be called a *partition sector*.
- Bootstrap code: Instructions to identify the configured bootable partition, then load and execute its volume boot record (VBR) as a chain loader.
- Optional 32-bit *disk timestamp*.[5]
- Optional 32-bit *disk signature*.[6][7][8][9]

2 Disk partitioning

IBM PC DOS 2.0 introduced the FDISK utility to set up and maintain MBR partitions. When a storage device has been partitioned according to this scheme, its MBR contains a partition table describing the locations, sizes, and other attributes of linear regions referred to as partitions.

The partitions themselves may also contain data to describe more complex partitioning schemes, such as extended boot records (EBRs), BSD disklabels, or Logical Disk Manager metadata partitions.[10]

The MBR is not located in a partition; it is located at a first sector of the device (physical offset 0), preceding the first partition. (The boot sector present on a non-partitioned device or within an individual partition is called a volume boot record instead.) In cases where the computer is running a DDO BIOS overlay or boot manager, the partition table may be moved to some other physical location on the device; e.g., Ontrack Disk Manager often placed a copy of the original MBR contents in the second sector, then hid itself from any subsequently booted OS or application, so the MBR copy was treated as if it were still residing in the first sector.

2.1 Sector layout

By convention, there are exactly four primary partition table entries in the MBR partition table scheme, although some operating systems and system tools extended this to five (Advanced Active Partitions (AAP) with PTS-DOS 6.60[11] and DR-DOS 7.07), eight (AST and NEC MS-DOS 3.x[12][13]), or even sixteen entries (with Ontrack Disk Manager).

2.2 Partition table entries

An artifact of hard disk technology from the era of the PC XT, the partition table subdivides a storage medium using units of cylinders, heads, and sectors (CHS addressing). These values no longer correspond to their namesakes in modern disk drives, as well as being irrelevant in other devices such as solid-state drives which do not physically have cylinders or heads.

In the CHS scheme, sector indices have (almost) always begun with sector 1 rather than sector 0 by convention, and due to an error in all versions of MS-DOS/PC DOS up to including 7.10, the number of heads is generally limited to 255 instead of 256. When a CHS address is too large to fit into these fields, the tuple (1023, 254, 63) is typically used today, although on older systems, and with older disk tools, the cylinder value often wrapped around modulo the CHS barrier near 8 GB, causing ambiguity and risks of data corruption. (If the situation involves a "protective" MBR on a disk with a GPT, Intel's Extensible Firmware Interface specification requires that the tuple (1023, 255, 63) be used.) The 10-bit cylinder value is recorded within two bytes in order to facilitate making calls to the original/legacy INT 13h BIOS disk access routines, where 16 bits were divided into sector and cylinder parts, and not on byte boundaries.[15]

Due to the limits of CHS addressing,[18][19] a transition was made to using LBA or logical block addressing. Both the partition length and partition start address are sector values stored in the partition table entries as 32-bit quantities. The sector size used to be considered fixed at 512 (2^9) bytes, and a broad range of important components including chipsets, boot sectors, operating systems, database engines, partitioning tools, backup and file system utilities and other software had this value hard-coded. Since the end of 2009, disk drives using a new technology known as Advanced Format and employing 4,096-byte sectors (4Kn) have been available, although the size of the sector for some of these drives was still reported as 512 bytes to the host system through conversion in the hard drive firmware and referred to as 512 emulation drives (512e).

Since block addresses and sizes are stored in the partition table of an MBR using 32 bits, the maximum size as well as the highest start address of a partition using drives that have 512-byte sectors (actual or emulated) cannot exceed 2 TiB−512 bytes (2,199,023,255,040 bytes or 4,294,967,295 ($2^{32}-1$) sectors × 512 (2^9) bytes per sector). Alleviating this capacity limitation was one of the prime motivations for the development of the GPT.

Since partitioning information is stored in the MBR partition table using a beginning block address and a length, it may in theory be possible to define partitions in such a way that the allocated space for a disk with 512-byte sectors gives a total size approaching 4 TiB, if all but one partition are located below the 2 TiB limit and the last one is assigned as starting at or close to block $2^{32}-1$ and specify the size as up to $2^{32}-1$, thereby defining a partition which requires 33 rather than 32 bits for the sector address to be accessed. However, in practice, only certain LBA-48 enabled operating systems, including GNU/Linux, FreeBSD and Windows 7[20] that use 64-bit sector addresses internally actually support this.

Due to code space constraints and the nature of the MBR partition table to only support 32 bits, boot sectors, even if enabled to support LBA-48 rather than LBA-28, often use 32-bit calculations, unless they are specifically designed to support the full address range of LBA-48 or are intended to run on 64-bit platforms only. Any boot code or operating system using 32-bit sector addresses internally would cause addresses to wrap around accessing this partition and thereby result in serious data corruption over all partitions.

For disks that present a sector size other than 512 bytes, such as USB external drives, there are limitations as well. A sector size of 4,096 results in an eight-fold increase in the size of a partition that can be defined using MBR, allowing partitions up to 16 TiB ($2^{32} \times 4096$ bytes) in size.[21] Versions of Windows more recent than Windows XP support the larger sector sizes as well as Mac OS X, and Linux has supported larger sector sizes since 2.6.31[22] or 2.6.32,[23] but issues with boot loaders, partitioning tools and computer BIOS implementations present certain limitations,[24] since they are often hard-wired to reserve only 512 bytes for sector buffers, causing memory to become overwritten for larger sector sizes. This may cause unpredictable behaviour as well, and therefore should be avoided when compatibility and standard conformity is an issue.

Where a data storage device has been partitioned with the GPT scheme, the master boot record will still contain a partition table, but its only purpose is to indicate the existence of the GPT and to prevent utility programs which understand only the MBR partition table scheme from creating any partitions in what they would otherwise see as free space on the disk, thereby accidentally erasing the GPT.

3 System bootstrapping

On IBM PC-compatible computers, the bootstrapping firmware contained within the ROM BIOS loads and executes the master boot record.[25] The PC/XT (type 5160) used an Intel 8088 microprocessor. In order to remain compatible, all x86 architecture systems start with the microprocessor in an operating mode referred to as real mode. The BIOS reads the MBR from the storage device into physical memory, and then it directs the microprocessor to the start of the boot code. Since the BIOS runs in real mode, the processor is in real mode when the MBR program begins to execute, and so the beginning of the MBR is expected to contain real mode machine language instructions.[25]

Due to the restricted size of the MBR's code section, it typically contains only a small program that copies additional code (such as a boot loader) from the storage device into memory. Control is then passed to this code, which is responsible for loading the actual operating system. This process is known as chain loading.

Popular MBR code programs were created for booting PC DOS and MS-DOS, and similar boot code remains in wide use. These boot sectors expect the fdisk partition table scheme to be in use, and scans the list of partitions in the MBR's embedded partition table to find the only one that is marked with the *active flag*.[26] It then loads and runs the volume boot record (VBR) of the active partition.

There are alternative boot code implementations, some of which are installed by boot managers, which operate in a variety of ways. Some MBR code loads additional code for a boot manager 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. A MBR program may interact with the user to determine which partition on which drive should boot, and may transfer control to the MBR of a different drive. Other MBR code contains a list of disk locations (often corresponding to the contents of files in a filesystem) of the remainder of the boot manager code to load and to execute. (The first relies on behavior that is not universal across all disk partitioning utilities, most notably those which read and write GPTs. The last requires that the embedded list of disk locations be updated when changes are made that would relocate the remainder of the code.)

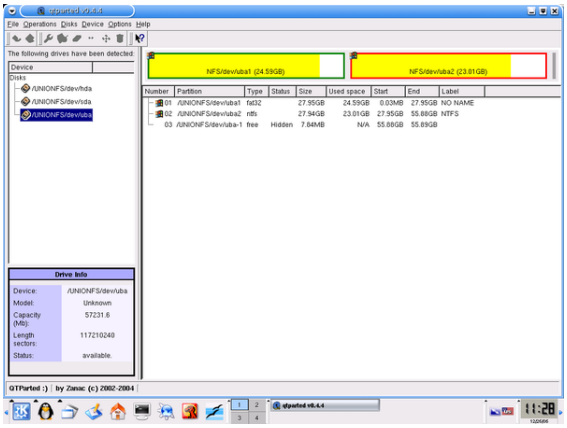
On machines that do not use x86 processors, or on x86 machines with non-BIOS firmware such as Open Firmware or Extensible Firmware Interface (EFI) firmware, this design is unsuitable, and the MBR is not used as part of the system bootstrap.[27] EFI firmware is instead capable of directly understanding the GPT partitioning scheme and the FAT filesystem format, and loads and runs programs held as files in the EFI System partition.[28] The MBR will be involved only insofar as it might contain a partition table for compatibility purposes if the GPT partition table scheme has been used.

There is some MBR replacement code that emulates EFI firmware's bootstrap, which makes non-EFI machines capable of booting from disks using the GPT partitioning scheme. It detects a GPT, places the processor in the correct operating mode, and loads the EFI compatible code from disk to complete this task.

4 Disk identity

In addition to the bootstrap code and a partition table, master boot records may contain a disk signature. This is a 32-bit value that is intended to identify uniquely the disk medium (as opposed to the disk unit—the two not necessarily being the same for removable hard disks).

The disk signature was introduced by Windows NT version 3.5, but it is now used by several operating systems, including the GNU/Linux version 2.6 and later. GNU/Linux tools can use the NT disk signature to de-



Information contained in the Partition Table of an external hard drive as it appears in the utility program QtParted, running under GNU/Linux

termine which disk the machine booted from.[29]

Windows NT (and later Microsoft operating systems) uses the disk signature as an index to all the partitions on any disk ever connected to the computer under that OS; these signatures are kept in Windows Registry keys, primarily for storing the persistent mappings between disk partitions and drive letters. It may also be used in BOOT.INI files (though most do not), to describe the location of bootable Windows NT (or later) partitions.[30] One key (among many) where NT disk signatures appear in a Windows 2000/XP registry is:

HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices\

If a disk's signature stored in the MBR was $A8_{hex}$ $E1_{hex}$ $B9_{hex}$ $D2_{hex}$ (in that order) and its first partition corresponded with logical drive C: under Windows, then the REG_BINARY data under the key value **\DosDevices\C:** would be:

A8 E1 B9 D2 00 7E 00 00 00 00 00 00_{hex}

The first four bytes are said disk signature. (*Note:* In other keys, these bytes may appear in reverse order from that found in the MBR sector.) These are followed by eight more bytes, forming a 64-bit integer, in little endian notation, which are used to locate the byte offset of this partition. In this case, 00_{hex} $7E_{hex}$ corresponds to the hexadecimal value $7E00_{hex}$ (32,256). If we assume the drive in question reports a sector size of 512 bytes, then dividing this byte offset by 512 results in 63, which is the physical sector number (or LBA) containing the first sector of the partition (unlike the *sector count* used in the sectors value of CHS tuples, which counts from **one**, the absolute or LBA sector value starts counting from **zero**).

If this disk had another partition with the values 00 F8 93 71 02_{hex} following the disk signature (under, e.g., the key value \DosDevices\D:), it would begin at byte offset $00027193F800_{hex}$ (10,495,457,280), which is also the first byte of physical sector 20,498,940.

Starting with Windows Vista, the disk signature is also stored in the Boot Configuration Data (BCD) store and the boot process depends on it.[31] If the disk signature changes, cannot be found or has a conflict, Windows is unable to boot.[32] Unless Windows is forced to use the overlapping part of the LBA address of the Advanced Active Partition entry as pseudo-disk signature, Windows' usage is conflictive with the Advanced Active Partition feature of PTS-DOS 7 and DR-DOS 7.07, in particular if their boot code is located outside the first 8 GB of the disk, so that LBA addressing must be used.

5 Programming considerations

The MBR originated in the PC XT.[33] IBM PC-compatible computers are little-endian, which means the processor stores numeric values spanning two or more bytes in memory least significant byte first. The format of the MBR on media reflects this convention. Thus, the MBR signature will appear in a disk editor as the sequence 55_{hex} AA_{hex}.[lower-alpha 1]

The bootstrap sequence in the BIOS will load the first valid MBR that it finds into the computer's physical memory at address 0000_{hex}:$7C00_{hex}$.[33] The last instruction executed in the BIOS code will be a "jump" to that address, to direct execution to the beginning of the MBR copy. The primary validation for most BIOSes is the signature at offset $+1FE_{hex}$, although a BIOS implementer may choose to include other checks, such as verifying that the MBR contains a valid partition table without entries referring to sectors beyond the reported capacity of the disk.

While the MBR boot sector code expects to be loaded at physical address 0000_{hex}:$7C00_{hex}$,[lower-alpha 7] all the memory from physical address 0000_{hex}:0501_{hex} (address 0000_{hex}:0500_{hex} is the last one used by a Phoenix BIOS)[15] to 0000_{hex}:$7FFF_{hex}$,[33] later relaxed to 0000_{hex}:$FFFF_{hex}$[34] (and sometimes[lower-alpha 8] up to 9000_{hex}:$FFFF_{hex}$)—the end of the first 640 KB—is available in real mode.[lower-alpha 9] The INT 12h BIOS interrupt call may help in determining how much memory can be allocated safely (by default, it simply reads the base memory size in KB from segment:offset location 0040_{hex}:0013_{hex}, but it may be hooked by other resident pre-boot software like BIOS overlays, RPL code or viruses to reduce the reported amount of available memory in order to keep other boot stage software like boot sectors from overwriting them).

The last 66 bytes of the 512-byte MBR are reserved for the partition table and other information, so the MBR boot sector program must be small enough to fit within 446 bytes of memory or less. The MBR code may communicate with the user, examine the partition table. Eventually, the MBR will need to perform its main task, and load the program that will perform the next stage of the boot process, usually by making use of INT 13h

BIOS calls. While it may be convenient to think of the MBR and the program that it loads as separate and discrete, a clear distinction between the MBR and the loaded OS is not technically required—the MBR, or parts of it,[lower-alpha 10] could stay resident in RAM and be used as part of the loaded program, after the MBR transfers control to that program. The same is true of a volume boot record, whether that volume is a floppy disk or a fixed disk partition. However, in practice, it is typical for the program loaded by a boot record program to discard and overwrite the RAM image of the latter, so that its only function is as the first link of the boot loader chain.

From a technical standpoint, it is important to note that the distinction between an MBR and a volume boot record exists only at the user software level, above the BIOS firmware. (Here, the term "user software" refers to both operating system software and application software.) To the BIOS, removable (e.g. floppy) and fixed disks are essentially the same. For either, the BIOS reads the first physical sector of the media into RAM at absolute address $7C00_{hex}$, checks the signature in the last two bytes of the loaded sector, and then, if the correct signature is found, transfers control to the first byte of the sector with a jump (JMP) instruction. The only real distinction that the BIOS makes is that (by default, or if the boot order is not configurable) it attempts to boot from the first removable disk before trying to boot from the first fixed disk. From the perspective of the BIOS, the action of the MBR loading a volume boot record into RAM is exactly the same as the action of a floppy disk volume boot record loading the object code of an operating system loader into RAM. In either case, the program that BIOS loaded is going about the work of chain loading an operating system. The distinction between an MBR and a volume boot record is an OS software-level abstraction, designed to help people to understand the operational organization and structure of the system. That distinction doesn't exist for the BIOS. Whatever the BIOS directly loads, be it an MBR or a volume boot record, is given total control of the system, and the BIOS from that point is solely at the service of that program. The loaded program owns the machine (until the next reboot, at least). With its total control, this program is not required to ever call the BIOS again and may even shut BIOS down completely, by removing the BIOS ISR vectors from the processor interrupt vector table, and then overwrite the BIOS data area. This is mentioned to emphasize that the boot program that the BIOS loads and runs from the first sector of a disk can truly do anything, so long as the program does not call for BIOS services or allow BIOS ISRs to be invoked after it has disrupted the BIOS state necessary for those services and ISRs to function properly.

As stated above, the conventional MBR bootstrap code loads and runs (boot loader- or operating system-dependent) volume boot record code that is located at the beginning of the "active" partition. A conventional volume boot record will fit within a 512-byte sector, but it is safe for MBR code to load additional sectors to accommodate boot loaders longer than one sector, provided they do not make any assumptions on what the sector size is. In fact, at least 1 KB of RAM is available at address $7C00_{hex}$ in every IBM XT- and AT-class machine, so a 1 KB sector could be used with no problem. Like the MBR, a volume boot record normally expects to be loaded at address 0000_{hex}:$7C00_{hex}$. This derives from the fact that the volume boot record design originated on unpartitioned media, where a volume boot record would be directly loaded by the BIOS boot procedure; as mentioned above, the BIOS treats MBRs and volume boot records (VBRs)[lower-alpha 11] exactly alike. Since this is the same location where the MBR is loaded, one of the first tasks of an MBR is to relocate itself somewhere else in memory. The relocation address is determined by the MBR, but it is most often 0000_{hex}:0600_{hex} (for MS-DOS/PC DOS, OS/2 and Windows MBR code) or 0060_{hex}:0000_{hex} (most DR-DOS MBRs). (Even though both of these segmented addresses resolve to the same physical memory address in real mode, for Apple Darwin to boot, the MBR must be relocated to 0000_{hex}:0600_{hex} instead of 0060_{hex}:0000_{hex}, since the code depends on the DS:SI pointer to the partition entry provided by the MBR, but it erroneously refers to it via 0000_{hex}:SI only.[35]) While the MBR code relocates itself it is still important not to relocate to other addresses in memory because many VBRs will assume a certain standard memory layout when loading their boot file.

The *Status* field in a partition table record is used to indicate an active partition. Standard-conformant MBRs will allow only one partition marked active and use this as part of a sanity-check to determine the existence of a valid partition table. They will display an error message, if more than one partition has been marked active. Some non-standard MBRs will not treat this as an error condition and just use the first marked partition in the row.

Traditionally, values other than 00_{hex} (not active) and 80_{hex} (active) were invalid and the bootstrap program would display an error message upon encountering them. However, the Plug and Play BIOS Specification and BIOS Boot Specification (BBS) allowed other devices to become bootable as well since 1994.[34][36] Consequently, with the introduction of MS-DOS 7.10 (Windows 95B) and higher, the MBR started to treat a set bit 7 as active flag and showed an error message for values 01_{hex}..$7F_{hex}$ only. It continued to treat the entry as physical drive unit to be used when loading the corresponding partition's VBR later on, thereby now also accepting other boot drives than 80_{hex} as valid, however, MS-DOS did not make use of this extension by itself. Storing the actual physical drive number in the partition table does not normally cause backward compatibility problems, since the value will differ from 80_{hex} only on drives other than the first one (which have not been bootable before, anyway). However, even with systems enabled to boot off other drives, the extension may still not work univer-

sally, for example, after the BIOS assignment of physical drives has changed, for example when drives are removed, added or swapped. Therefore, per the BIOS Boot Specification (BBS),[34] it is best practice for a modern MBR accepting bit 7 as active flag to pass on the DL value originally provided by the BIOS instead of using the entry in the partition table.

5.1 BIOS to MBR interface

The MBR is loaded at memory location 0000_{hex}:$7C00_{hex}$ and with the following CPU registers set up when the prior bootstrap loader (normally the IPL in the BIOS) passes execution to it by jumping to 0000_{hex}:$7C00_{hex}$ in the CPU's real mode.

- CS:IP = 0000_{hex}:$7C00_{hex}$ (fixed)

  Some Compaq BIOSes erroneously use $07C0_{hex}$:0000_{hex} instead. While this resolves to the same location in real mode memory, it is non-standard and should be avoided, since MBR code assuming certain register values or not written to be relocatable may not work otherwise.

- DL = boot drive unit (fixed disks / removable drives: 80_{hex} = first, 81_{hex} = second, ..., FE_{hex}; floppies / superfloppies: 00_{hex} = first, 01_{hex} = second, ..., $7E_{hex}$; values $7F_{hex}$ and FF_{hex} are reserved for ROM / remote drives and must not be used on disk).

  DL is supported by IBM BIOSes as well as most other BIOSes. The Toshiba T1000 BIOS is known to not support this properly, and some old Wyse 286 BIOSes use DL values greater or equal to 2 for fixed disks (thereby reflecting the logical drive numbers under DOS rather than the physical drive numbers of the BIOS). USB sticks configured as removable drives typically get an assignment of DL = 80_{hex}, 81_{hex}, etc. However, some rare BIOSes erroneously presented them under DL = 01_{hex}, just as if they were configured as superfloppies.

  A standard conformant BIOS assigns numbers greater or equal to 80_{hex} exclusively to fixed disk / removable drives, and traditionally only values 80_{hex} and 00_{hex} were passed on as physical drive units during boot. By convention, only fixed disks / removable drives are partitioned, therefore, the only DL value a MBR could see traditionally was 80_{hex}. Many MBRs were coded to ignore the DL value and work with a hard-wired value (normally 80_{hex}), anyway.

  The Plug and Play BIOS Specification and BIOS Boot Specification (BBS) allow other devices to become bootable as well since 1994.[34][36] The later recommends that MBR and VBR code should use DL rather than internally hardwired defaults.[34] This will also ensure compatibility with various non-standard assignments (see examples above), as far as the MBR code is concerned.

  Bootable CD-ROMs following the El Torito specification may contain disk images mounted by the BIOS to occur as floppy or superfloppies on this interface. DL values of 00_{hex} and 01_{hex} may also be used by Protected Area Run Time Interface Extension Services (PARTIES) and Trusted Computing Group (TCG) BIOS extensions in Trusted mode to access otherwise invisible PARTIES partitions, disk image files located via the Boot Engineering Extension Record (BEER) in the last physical sector of a hard disk's Host Protected Area (HPA). While designed to emulate floppies or superfloppies, MBR code accepting these non-standard DL values allows to use images of partitioned media at least in the boot stage of operating systems.

- DH bit 5 = 0: device supported through INT 13h; else: don't care (should be zero). DH is supported by some IBM BIOSes.

- Some of the other registers may typically also hold certain register values (DS, ES, SS = 0000_{hex}; SP = 0400_{hex}) with original IBM ROM BIOSes, but this is nothing to rely on, as other BIOSes may use other values. For this reason, MBR code by IBM, Microsoft, Digital Research, etc. never did take any advantage of it. Relying on these register values in boot sectors may also cause problems in chain-boot scenarios.

Systems with Plug-and-Play BIOS or BBS support will provide a pointer to PnP data in addition to DL:[34][36]

- DL = boot drive unit (see above)
- ES:DI = points to "$PnP" installation check structure

  This information allows the boot loader in the MBR (or VBR, if passed on) to actively interact with the BIOS or a resident PnP / BBS BIOS overlay in memory in order to configure the boot order, etc., however, this information is ignored by most standard MBRs and VBRs. Ideally, ES:DI is passed on to the VBR for later use by the loaded operating system, but PnP-enabled operating systems typically also have fallback methods to retrieve the PnP BIOS entry point later on so that most operating systems do not rely on this.

5.2 MBR to VBR interface

By convention, a standard conformant MBR passes execution to a successfully loaded VBR, loaded at memory location 0000_{hex}:$7C00_{hex}$, by jumping to 0000_{hex}:$7C00_{hex}$ in the CPU's real mode with the following registers maintained or specifically set up:

- CS:IP = 0000_{hex}:$7C00_{hex}$[lower-alpha 12] (constant)
- DL = boot drive unit (see above)

  MS-DOS 2.0-7.0 / PC DOS 2.0-6.3 MBRs do not pass on the DL value received on entry, but they rather use the boot status entry in the partition table entry of the selected primary partition as physical boot drive unit. Since this is, by convention, 80_{hex} in most MBR partition tables, it won't change things unless the BIOS attempted to boot off a physical device other than the first fixed disk / removable drive in the row. This is also the reason why these operating systems cannot boot off a second hard disk, etc. Some FDISK tools allow to mark partitions on secondary disks as "active" as well. In this situation, knowing that these operating systems cannot boot off other drives anyway, some of them continue to use the traditionally fixed value of 80_{hex} as active marker, whereas others use values corresponding with the currently assigned physical drive unit (81_{hex}, 82_{hex}), thereby allowing to boot off other drives at least in theory. In fact, this will work with many MBR codes, which take a set bit 7 of the boot status entry as active flag rather than insisting on 80_{hex}, however, MS-DOS/PC DOS MBRs are hard-wired to accept the fixed value of 80_{hex} only. Storing the actual physical drive number in the partition table will also cause problems, when the BIOS assignment of physical drives changes, for example when drives are removed, added or swapped. Therefore, for a normal MBR accepting bit 7 as active flag and otherwise just using and passing on to the VBR the DL value originally provided by the BIOS allows for maximum flexibility. MS-DOS 7.1 - 8.0 MBRs have changed to treat bit 7 as active flag and any values 01_{hex}..$7F_{hex}$ as invalid, but they still take the physical drive unit from the partition table rather than using the DL value provided by the BIOS. DR-DOS 7.07 extended MBRs treat bit 7 as active flag and use and pass on the BIOS DL value by default (including non-standard values 00_{hex}..01_{hex} used by some BIOSes also for partitioned media), but they also provide a special NEWLDR configuration block in order to support alternative boot methods in conjunction with LOADER and REAL/32 as well as to change the detail behaviour of the MBR, so that it can also work with drive values retrieved from the partition table (important in conjunction with LOADER and AAPs, see NEWLDR offset $+00C_{hex}$), translate Wyse non-standard drive units 02_{hex}..$7F_{hex}$ to 80_{hex}..FD_{hex}, and optionally fix up the drive value (stored at offset $+19_{hex}$ in the Extended BIOS Parameter Block (EBPB) or at sector offset $+1FD_{hex}$) in loaded VBRs before passing execution to them (see NEWLDR offset $+014_{hex}$)—this also allows other boot loaders to use NEWLDR as a chain-loader, configure its in-memory image on the fly and "tunnel" the loading of VBRs, EBRs, or AAPs through NEWLDR.

- The contents of DH and ES:DI should be preserved by the MBR for full Plug-and-Play support (see above), however, many MBRs, including those of MS-DOS 2.0 - 8.0 / PC DOS 2.0 - 6.3 and Windows NT/2000/XP, do not. (This is unsurprising, since those versions of DOS predate the Plug-and-Play BIOS standard, and previous standards and conventions indicated no requirements to preserve any register other than DL.) Some MBRs set DH to 0.

The MBR code passes additional information to the VBR in many implementations:

- DS:SI = points to the 16-byte MBR partition table entry (in the relocated MBR) corresponding with the activated VBR. PC-MOS 5.1 depends on this to boot if no partition in the partition table is flagged as bootable. In conjunction with LOADER, Multiuser DOS and REAL/32 boot sectors use this to locate the boot sector of the active partition (or another bootstrap loader like IBMBIO.LDR at a fixed position on disk) if the boot file (LOADER.SYS) could not be found. PTS-DOS 6.6 and S/DOS 1.0 use this in conjunction with their Advanced Active Partition (AAP) feature. In addition to support for LOADER and AAPs, DR-DOS 7.07 can (sometimes) use this to determine the necessary INT 13h access method when using its dual CHS/LBA VBR code and it will update the boot drive / status flag field in the partition entry according to the effectively used DL value. Darwin bootloaders (Apple's boot1h, boot1u, and David Elliott's boot1fat32) depend on this pointer as well, but additionally they don't use DS, but assume it to be set to 0000_{hex} instead.[35] This will cause problems if this assumption is incorrect. The MBR code of OS/2, MS-DOS 2.0 to 8.0, PC DOS 2.0 to 7.10 and Windows NT/2000/XP provides this same interface as well, although these systems do not use it. The Windows Vista/7 MBRs no longer provide this DS:SI pointer. While some extensions only depend on the 16-byte partition table entry itself, other extensions may re-

quire the whole 4 (or 5 entry) partition table to be present as well.

- DS:BP = optionally points to the 16-byte MBR partition table entry (in the relocated MBR) corresponding with the activated VBR. This is identical to the pointer provided by DS:SI (see above) and is provided by MS-DOS 2.0-8.0, PC DOS 2.0-7.10, Windows NT/2000/XP/Vista/7 MBRs. It is, however, not supported by most third-party MBRs.

Under DR-DOS 7.07 an extended interface may be optionally provided by the extended MBR and in conjunction with LOADER:

- AX = magic signature indicating the presence of this NEWLDR extension ($0EDC_{hex}$)
- DL = boot drive unit (see above)
- DS:SI = points to the 16-byte MBR partition table entry used (see above)
- ES:BX = start of boot sector or NEWLDR sector image (typically $7C00_{hex}$)
- CX = reserved

In conjunction with GPT, an *Enhanced Disk Drive Specification* (EDD) 4 Hybrid MBR proposal recommends another extension to the interface:[37]

- EAX = 54504721_{hex} ("!GPT")
- DL = boot drive unit (see above)
- DS:SI = points to a Hybrid MBR handover structure, consisting of a 16-byte dummy MBR partition table entry (with all bits set except for the boot flag at offset $+0_{hex}$ and the partition type at offset $+4_{hex}$) followed by additional data. This is partially compatible with the older DS:SI extension discussed above, if only the 16-byte partition entry, not the whole partition table is required by these older extensions.

  Since older operating systems (including their VBRs) do no support this extension nor are they able to address sectors beyond the 2 TiB (cca 2.2 TB) barrier, a GPT-enabled hybrid boot loader should still emulate the 16-byte dummy MBR partition table entry if the boot partition is located within the first 2 TiB.[lower-alpha 13]

- ES:DI = points to "$PnP" installation check structure (see above)

6 Editing/replacing MBR contents

Though it is possible to manipulate the bytes in the MBR sector directly using various disk editors, there are tools to write fixed sets of functioning code to the MBR. Since MS-DOS 5.0, the program FDISK has included the switch /MBR, which will rewrite the MBR code.[38] Under Windows 2000 and Windows XP, the Recovery Console can be used to write new MBR code to a storage device using its fixmbr command. Under Windows Vista and Windows 7, the Recovery Environment can be used to write new MBR code using the BOOTREC /FIXMBR command. Some third-party utilities may also be used for directly editing the contents of partition tables (without requiring any knowledge of hexadecimal or disk/sector editors), such as MBRWizard.[lower-alpha 14]

dd is also a commonly used POSIX command to read or write to any location on a storage device, MBR included. In Linux, ms-sys may be used to install a Windows MBR. The GRUB and LILO projects have tools for writing code to the MBR sector, namely grub-install and lilo -mbr. The GRUB Legacy interactive console can write to the MBR, using the setup and embed commands, but GRUB2 currently requires grub-install to be run from within an operating system.

Various programs are able to create a "backup" of both the primary partition table and the logical partitions in the extended partition.

Linux sfdisk (on a SystemRescueCD) is able to save a backup of the primary and extended partition table. It creates a file that can be read in a text editor, or this file can be used by sfdisk to restore the primary/extended partition table. An example command to back up the partition table is sfdisk -d /dev/hda > hda.out and to restore is sfdisk /dev/hda < hda.out. It is possible to copy the partition table from one disk to another this way, useful for setting up mirroring, but it should be noted that sfdisk executes the command without prompting/warnings using sfdisk -d /dev/sda | sfdisk /dev/sdb.[39]

7 See also

- Extended boot record (EBR)
- Volume boot record (VBR)
- GUID Partition Table (GPT)
- BIOS Boot partition
- EFI System partition
- Boot engineering extension record (BEER)
- Host protected area (HPA)
- Device configuration overlay (DCO)

- Apple partition map (APM)
- Amiga rigid disk block (RDB)
- BSD disklabel
- Boot loader
- Disk cloning
- Recovery disc
- GNU Parted
- Partition alignment

8 Notes

[1] The signature at offset $+1FE_{hex}$ in boot sectors is 55_{hex} AA_{hex}, that is 55_{hex} at offset $+1FE_{hex}$ and AA_{hex} at offset $+1FF_{hex}$. Since little-endian representation must be assumed in the context of IBM PC compatible machines, this can be written as 16-bit word $AA55_{hex}$ in programs for x86 processors (note the swapped order), whereas it would have to be written as $55AA_{hex}$ in programs for other CPU architectures using a big-endian representation. Since this has been mixed up numerous times in books and even in original Microsoft reference documents, this article uses the offset-based byte-wise on-disk representation to avoid any possible misinterpretation.

[2] In order to ensure the integrity of the MBR boot loader code, it is important that the bytes at $+0DA_{hex}$ to $+0DF_{hex}$ are never changed, unless either *all* six bytes represent a value of 0 or the whole MBR bootstrap loader code (except for the (extended) partition table) is replaced at the same time as well. This includes resetting these values to $00\ 00\ 00\ 00\ 00\ 00_{hex}$ unless the code stored in the MBR is known. Windows adheres to this rule.

[3] Originally, status values other than 00_{hex} and 80_{hex} were invalid, but modern MBRs treat the bit 7 as active flag and use this entry to store the physical boot unit.

[4] The starting sector fields are limited to 1023+1 cylinders, 255+1 heads, and 63 sectors; ending sector fields have the same limitations.

[5] The range for sector is 1 through 63; the range for cylinder is 0 through 1023; the range for head is 0 through 255 inclusive.[15]

[6] The number of sectors is an index field; thus, the zero value is invalid, reserved and must not be used in normal partition entries. The entry is used by operating systems in certain circumstances; in such cases the CHS addresses are ignored.[17]

[7] The address 0000_{hex}:$7C00_{hex}$ is the first byte of the 32nd KB of RAM. As a historical note, the loading of the boot program at this address was the obvious reason why, while the minimum RAM size of an original IBM PC (type 5150) was 16 KB, 32 KB were required for the disk option in the IBM XT.

[8] If there is an EBDA, the available memory ends below it.

[9] Very old machines may have less than 640 KB ($A0000_{hex}$ or 655,360 bytes) of memory. In theory, only 32 KB (up to 0000_{hex}:$7FFF_{hex}$) or 64 KB (up to 0000_{hex}:$FFFF_{hex}$) are guaranteed to exist; this would be the case on an IBM XT-class machine equipped with only the required minimum amount of memory for a disk system.

[10] such as a routine to do a primitive block move, user I/O, or parse a file system directory

[11] This applies when BIOS handles a VBR, which is when it is in the first physical sector of unpartitioned media. Otherwise, BIOS has nothing to do with the VBR. The design of VBRs is such as it is because VBRs originated solely on unpartitioned floppy disk media—the type 5150 IBM PC originally had no hard disk option—and the partitioning system using an MBR was later developed as an adaptation to put more than one volume, each beginning with its own VBR as-already-defined, onto a single fixed disk. By this design, essentially the MBR emulates the BIOS boot routine, doing the same things the BIOS would do to process that VBR and set up the initial operating environment for it if the BIOS found that VBR on an unpartitioned medium.

[12] IP is set as a result of the jump. CS may be set to 0 either by making a far jump or by loading it explicitly before making a near jump. (It is impossible for jumped-to x86 code to detect whether a near or far jump was used to reach it [unless the code that made the jump separately passes this information in some way].)

[13] This is not part of the above mentioned proposal, but a natural consequence of pre-existing conditions.

[14] For example, *PowerQuest's Partition Table Editor* (PTEDIT32.EXE), which runs under Windows operating systems, is still available here: Symantec's FTP site.

9 References

[1] Denis Howe (May 19, 2009). "master boot record". *FOLDOC*. Retrieved 2 May 2015.

[2] "Windows support for hard disks that are larger than 2 TB". Microsoft. 2013-06-26. Retrieved 2013-08-28.

[3] "Mystery Bytes of the Win95B/98/SE/Me MBR". Thestarman.pcministry.com. 2004-09-04. Retrieved 2014-04-17.

[4] Lucas, Michael (2003). *Absolute OpenBSD: Unix for the practical paranoid*. p. 73. ISBN 9781886411999. Retrieved 2011-04-09. Every operating system includes tools to manage MBR partitions. Unfortunately, every operating system handles MBR partitions in a slightly different manner.

[5] Sedory, Daniel B. (2004). "The Mystery Bytes (or the Drive/Timestamp Bytes) of the MS-Windows 95B, 98, 98SE and Me Master Boot Record (MBR)". *Master Boot Records*. thestarman.pcministry.com. Retrieved 2012-08-25.

[6] Norton, Peter; Clark, Scott (2002). *Peter Norton's New Inside the PC*. Sams Publishing. pp. 360–361. ISBN 0-672-32289-7.

[7] Graves, Michael W. (2004). *A+ Guide To PC Hardware Maintenance and Repair*. Thomson Delmar. p. 276. ISBN 1-4018-5230-0.

[8] Andrews, Jean (2003). *Upgrade and Repair with Jean Andrews*. Thomson Course Technology. p. 646. ISBN 1-59200-112-2.

[9] Boswell, William (2003). *Inside Windows Server 2003*. Addison-Wesley Professional. p. 13. ISBN 0-7357-1158-5.

[10] Smith, Roderick W. (2000). *The Multi-Boot Configuration Handbook*. Que Publishing. pp. 260–261. ISBN 0-7897-2283-6.

[11] Brouwer, Andries Evert. "Properties of partition tables". *Partition types*. Matthias Paul: "PTS-DOS [uses] a special fifth partition entry in front of the other four entries in the MBR and corresponding AAP-aware MBR bootstrap code."

[12] Brouwer, Andries Evert. "Properties of partition tables". *Partition types*. Some OEM systems, such as AST DOS (type 14_{hex}) and NEC DOS (type 24_{hex}) had 8 instead of 4 partition entries in their MBR sectors. (Matthias Paul). (NB. NEC MS-DOS 3.30 and AST MS-DOS partition tables with eight entries are preceded with a signature $A55A_{hex}$ at offset $+17C_{hex}$.)

[13] Sedory, Daniel B. "Notes on the Differences in one OEM version of the DOS 3.30 MBR". *Master Boot Records*. When we added partitions to this NEC table, the first one was placed at offsets $+1EE_{hex}$ through $+1FD_{hex}$ and the next entry was added just above it. So, the entries are inserted and listed backwards from that of a normal Table. Thus, looking at such a Table with a disk editor or partition listing utility, it would show the first entry in a NEC eight-entry table as being the last one (fourth entry) in a normal Partition Table. Shows an 8-entry partition table and where its boot code differs from MS-DOS 3.30.

[14] "Partition Table". osdev.org. Retrieved 2013-11-15.

[15] *System BIOS for IBM PC/XT/AT Computers and Compatibles*. Phoenix technical reference. Addison-Wesley. 1989. ISBN 0-201-51806-6.

[16] Brouwer, Andries Evert. "List of partition identifiers for PCs". *Partition types*.

[17] Wood, Sybil (2002). *Microsoft Windows 2000 Server Operations Guide*. Microsoft Press. p. 18. ISBN 9780735617964.

[18] "An Introduction to Hard Disk Geometry". *Tech Juice*. 2011-08-08. Retrieved 2013-04-19.

[19] Kozierok, Charles M. (2001-04-17). "BIOS and the Hard Disk". *The PC Guide*. Retrieved 2013-04-19.

[20] Smith, Robert (2011-06-26). "Working Around MBR's Limitations". *GPT fdisk Tutorial*. Retrieved 2013-04-20.

[21] "More than 2 TiB on a MBR disk". superuser.com. 2013-03-07. Retrieved 2013-10-22.

[22] "Transition to Advanced Format 4K Sector Hard Drives". *Tech Insight*. Seagate Technology. Retrieved 2013-04-19.

[23] Calvert, Kelvin (2011-03-16). "WD AV-GP Large Capacity Hard Drives" (PDF). Western Digital. Retrieved 2013-04-20.

[24] Smith, Roderick W. (2010-04-27). "Linux on 4KB-sector disks: Practical advice". *DeveloperWorks*. IBM. Retrieved 2013-04-19.

[25] "MBR (x86)". *OSDev Wiki*. OSDev.org. 2012-03-05. Retrieved 2013-04-20.

[26] Sedory, Daniel B. (2003-07-30). "IBM DOS 2.00 Master Boot Record". The Starman's Realm. Retrieved 2011-07-22.

[27] Singh, Amit (2009-12-25). "Booting Mac OS X". Mac OS X Internals: The Book. Retrieved 2011-07-22.

[28] de Boyne Pollard, Jonathan (2011-07-10). "FGA: The EFI boot process". *Frequently Given Answers*. Jonathan de Boyne Pollard. Retrieved 2011-07-22.

[29] Domsch, Matt. "Re: RFC 2.6.0 EDD enhancements". *Linux Kernel Mailing List*.

[30] "Windows may use Signature() syntax in the BOOT.INI file". *KnowledgeBase*. Microsoft.

[31] "Vista's MBR Disk Signature". *Multibooters: Dual and Multibooting with Vista*. January 2007. Retrieved 2013-04-19.

[32] Russinovich, Mark (2011-11-08). "Fixing Disk Signature Collisions". *Mark Russinovich's Blog*. Microsoft. Retrieved 2013-04-19.

[33] Sakamoto, Masahiko (2010-05-13). "Why BIOS loads MBR into 0x7C00 in x86?". Glamenv-Septzen.net. Retrieved 2011-05-04.

[34] Compaq; Phoenix Technologies; Intel (1996-01-11). "BIOS Boot Specification 1.01" (PDF). ACPICA. Retrieved 2013-04-20.

[35] Elliott, David (2009-10-12). "Why does the "standard" MBR set SI?". *tgwbd.org*. Retrieved 2013-04-20.

[36] Compaq; Phoenix Technologies; Intel (1994-05-05). "Plug and Play BIOS Specification 1.0A" (PDF). Intel. Retrieved 2013-04-20.

[37] Elliott, Robert (2010-01-04). "e09127r3 EDD-4 Hybrid MBR boot code annex" (PDF). T13 Technical Committee. Retrieved 2013-04-20.

[38] "FDISK /MBR rewrites the Master Boot Record". *Support*. Microsoft. 2011-09-23. Retrieved 2013-04-19.

[39] "sfdisk(8) – Linux man page". *die.net*. Retrieved 2013-04-20.

10 Further reading

- H. Gilbert. "Partitions and Volumes". *PC Lube & Tune*.
- Ray Knights. "Ray's Place". *MBR and Windows Boot Sectors (includes code disassembly and explanations of boot process)*.
- Hale Landis. "Master Boot Record". *How It Works*.
- Daniel B. Sedory. "MBRs (Master Boot Records)". *Boot Records Revealed!*. (Mirror site) (Another mirror)

11 External links

- Article on master boot record

12 Text and image sources, contributors, and licenses

12.1 Text

- **Master boot record** *Source:* https://en.wikipedia.org/wiki/Master_boot_record?oldid=714323313 *Contributors:* LC~enwiki, Mav, Uriyan, SolKarma, Ben-Zin~enwiki, Tempel, Hirzel, AdSR, Haakon, Mac, Ronz, Plop, TraxPlayer, Grin, Smaffy, HPA, Greenrd, WhisperToMe, Furrykef, Saltine, Fibonacci, Wernher, Shizhao, AnonMoos, Jeffq, Robbot, Tomchiukc, Antoinel~enwiki, Techtonik, Mattflaschen, Tobias Bergemann, DocWatson42, Jhf, Tom harrison, Rchandra, AlistairMcMillan, Pgan002, LiDaobing, Wzwz, Peter bertok, Now3d, Abdull, DmitryKo, Jarsyl, Sietse Snel, PatrikR, R. S. Shaw, Giraffedata, Jhertel, Ynhockey, Kbolino, Jonathan de Boyne Pollard, Uncle G, Paul Mackay~enwiki, Stefan h~enwiki, Kbdank71, Rjwilmsi, Jamie Kitson, Phantom784, FlaBot, Mirror Vax, Intersofia, Preslethe, Skierpage, Chobot, DustWolf, HoCkEy PUCK, WriterHound, YurikBot, Borgx, Laurentius, Hede2000, Wgungfu, Jrideout, DragonHawk, Długosz, Dogcow, PhilipO, Neil.steiner, Voidxor, Zwobot, Bucketsofg, Jeremy Visser, Xpclient, Speculatrix, Jan Vlug, Triskelios, SmackBot, The Monster, Eskimbot, Deminy, Rotemliss, Chris the speller, Bluebot, DStoykov, QTCaptain, EncMstr, OrangeDog, Nbarth, Ryan Roos, Luigi.a.cruz, AThing, Peyre, Kvng, Pathosbot, FatalError, Requestion, No1lakersfan, Marc W. Abel, Ntsimp, ColdShine, BruceEwing, Thijs!bot, Konradek, Al Lemos, Mojo Hand, Electron9, AntiVandalBot, Darklilac, 01001, Magioladitis, Bongwarrior, Soulbot, Robomojo, Gwern, Jav72, Alfe, Threetorts, PrestonH, Lanchon, Public Menace, Bigdumbdinosaur, Alan012, Asymmetric, TXiKiBoT, M gol, Milan Keršláger, Insanity Incarnate, Jimmi Hugh, TheStarman, Vs49688, Fnagaton, Digisus, ClueBot, Mild Bill Hiccup, AlptaBot, MicroVirus, Kin kad, Jensanthegreat, M4gnum0n, Leonard^Bloom, Aleksd, Callmejosh, DumZiBoT, Lumenos, C. A. Russell, Dsimic, Addbot, Mortense, Sergei, Nawcom, A:-)Brunuś, Rbeede, Yobot, Mklbtz, The Earwig, Crispmuncher, Knockwood, AnomieBOT, Materialscientist, Citation bot, ArthurBot, Alfred1520, Vineel567, Control.valve, FrescoBot, BenzolBot, Ndaiju, Van Rijn, Babahu, RedBot, Orenburg1, Maxipes Fik, Tbhotch, PsychicShades, EmausBot, WikitanvirBot, Angrytoast, Xmm0, DesertGeek, Ida Shaw, Bomazi, Zabacad, ClueBot NG, Bgcaus, Matthiaspaul, Friecode, Be..anyone, Helpful Pixie Bot, BG19bot, Jaxhop, Eidab, Blurybury, BattyBot, Jimw338, Codename Lisa, Mogism, Rosslagerwall, Linush100, PaulCuttler, Jodosma, Devon Sean McCULLOUGH, Jianhui67, ScotXW, MacGyves, BlueFenixReborn, UnitTwo, Ereecson and Anonymous: 213

12.2 Images

- **File:Qtparted-usb-hdd-snapshot.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/87/Qtparted-usb-hdd-snapshot.png *License:* Public domain *Contributors:* my own screen capture *Original artist:* Tomchiukc

12.3 Content license

- Creative Commons Attribution-Share Alike 3.0