From Blocks to Filesystems to Booting

How OpenBSD makes bags of blocks useful

K. Westerback, krw@openbsd.org

17 September 2022

EuroBSDCon 2022

Table of Contents

Introduction

Introduction

Things that will be discussed ...

Introduction

Data Structures used to tame block devices

- struct disklabel
- GUID Partition Table, a.k.a. GPT
- Master Boot Record, a.k.a. MBR
- Partition Boot Record, a.k.a. PBR

Introduction

Kernel functions using the data structures

- MI readdoslabel() and checklabel() in /usr/src/sys/kern/subr_disk.c
- MD [read | write]disklabel() in /usr/src/sys/arch/.../disk_subr.c
- Device entry points XXopen(), XXgetdisklabel() and ioctl's

Introduction

Userland programs using the data structures and kernel functions

- fdisk(8) manipulates GPT and MBR
- disklabel(8) manipulates struct disklabel
- installboot(8) sprinkles pixie dust necessary to boot OpenBSD

Introduction

Things that will **not** be discussed

- Extended MBR partitions
- Booting other/multiple operating systems
- Booting from the network, CD or DVD
- Peculiarities of sparc64, macppc, hppa and alpha

Introduction

A few important definitions:

block 512 bytes, a. k. a. DEV_BSIZE

daddr_t int64_t block offset

sector minimum number of bytes in an i/o, usually 512 or 4096

partition contiguous sequence of sectors

Degrees of Usefulness

Blocks

If the kernel finds a block device, userland can (ab)use it without any further configuration.

```
sysctl hw.disknames
```

lists the block devices the kernel has found, and their DUIDs if present. e. g.

```
hw.disknames=sd0:2a1a01275f0cbc1b,sd1:ac4d478b606f7154,sd2:
```

The DUID can be used in most place a device name is required, but the most common use is in fstab(5) entries. e. g.

```
2a1a01275f0cbc1b.l /home ffs rw,nodev,nosuid,softdep 1 2
```

Blocks

- Block devices (cd(4), fd(4), rd(4), sd(4), vnd(4), wd(4)) provide the kernel with enough information to construct i/o's
  - the number of sectors on the device
  - the size of a sector
  - the "raw" partition, a. k. a. 'c', covering all sectors
- this information is provided in a struct disklabel, generated when XXopen() calls XXgetdisklabel()
  - struct disklabel is one block, exactly 512 bytes
  - can describe up to 16 partitions
- Userland programs use ioctl's to obtain this information
  - DIOC**GP**DINFO returns the default information for the device
  - DIOC**G**DINFO returns the information currently cached by the kernel
  - DIOC**RL**DINFO reloads the kernel's cached information

Blocks

procedure SETUP

pledge(stdio disklabel unveil rpath wpath)

unveil(/dev/rsdNc)

open(/dev/rsdNc)

ioctl(DIOCGPDINFO)

pledge(stdio)

procedure WORK

while not done **do**

lseek(); *read*()

do stuff

lseek(); *write*()

Blocks

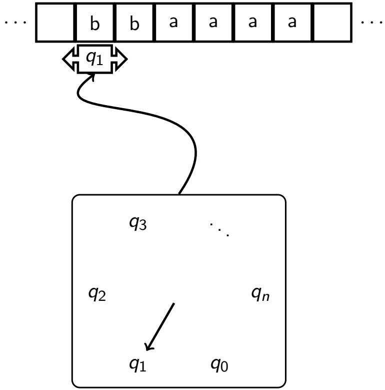

Figure 1: Turing Machine

Filesystems

When directly manipulating blocks becomes too cumbersome, blocks can be abstracted into a filesystem. Creating a single filesystem utilizing all the sectors on the device is straightforward.

1. newfs -t [type] /dev/rsdNc
2. mount -t [type] /mountpoint
3. add entry to fstab(5)

Job done?

Filesystems

Well . . . the device may have a GPT, MBR or PBR with useful partition information

Filesystems

- XXgetdisklabel() calls the MD function readdisklabel(), which calls the MI function readdoslabel() to check the device for a GPT, MBR or PBR
- readdoslabel() will add ("spoof") up to 8 partitions into the default struct disklabel
- spoofing is useful for media you want to be portable
- OpenBSD partitions (a. k. a. "A6" on MBR, "824cc7a0-36a8-11e3-890a-952519ad3f61" on GPT) are **not** spoofed

Spoofing

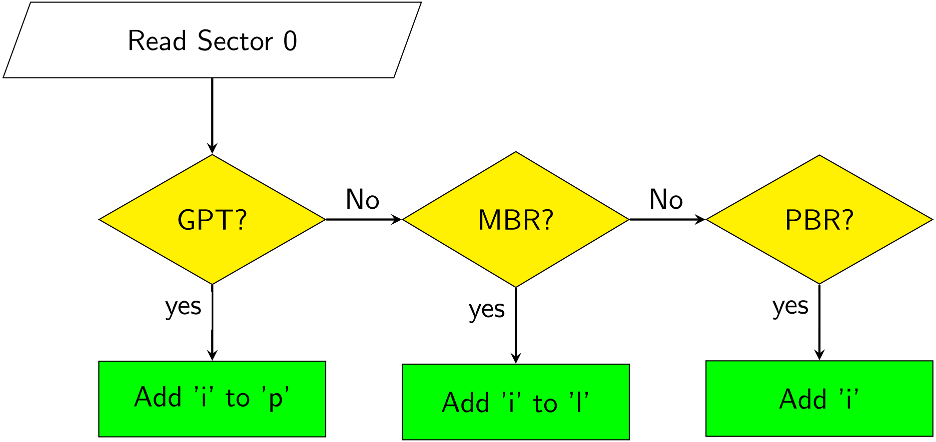

Figure 2: Spoofing

Spoofing

p_fstype (e. g. FS_BSDFFS)	uint8_t
$p\_offseth << 32 + p\_offset$	48-bit sector values
$p\_sizeh << 32 + p\_size$	

Figure 3: Disklabel spoofed partition

- DL_GETPOFFSET() and DL_GETPSIZE() compose values
- DL_SETPOFFSET() and DL_SETPSIZE() decompose them
- 48 bits allows 281,474,976,710,656 sectors
- for 512-byte sectors that works out to be 144PB
- kernel can address INT64_MAX (daddr_t) blocks, i. e. more than can currently be represented by a disklabel partition entry

Spoofing

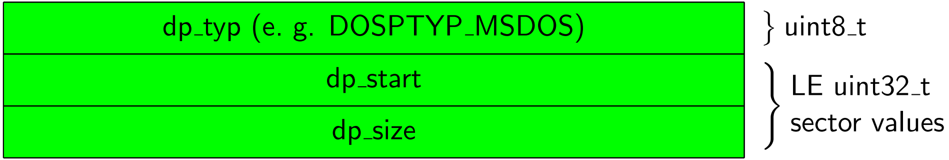

Figure 4: MBR partition info

Figure 5: GPT partition info

Spoofing

- initialize GPT or MBR with ‘fdisk -g’ or ‘fdisk -i’
- display GPT or MBR with ‘fdisk [-v]’
- edit GPT or MBR with ‘fdisk -e’

Spoofing

Recent fdisk(8) changes

- recognize more GPT partition types (BIOS Boot, High5 BBL, Apple APFS, etc.)
- protect some GPT partition types from editing
- more permissive GPT validation vs device size
- display “Micrsoft Basic Data” instead of “FAT12”
- always write GPT checksum fields as LE
- remove MBR-only partition types from GPT help
- remove GPT-only partition types from MBR help
- remove geometry editing
- recognize and display GPT partition attributes
- -b and -l are block instead of sector values, wasting less space

Spoofing

Recent readdoslabel() changes

- don't spoof GPT partitions with "Required" attribute

Job done?

OpenBSD Takes Control

Well . . .

- you may want OpenBSD functionality, e. .g.
  - softraid(4)
  - swap space
  - OpenBSD FFS
- you may want more than 8 partitions
- you may want a set of partitions different from the 8 that spoofing chooses

These things require that a disklabel with the desired partition configuration is written to disk. Historically OpenBSD took a straightforward approach when asked to do this.

All Your Sector Are Belong to Us!

OpenBSD Takes Control

1. fdisk(8)
   1.1 created a default GPT (-g) or MBR (-i)
   1.2 put all sectors, give or take some rounding and the GPT/MBR, into a single OpenBSD partition
   1.3 wrote the GPT or MBR to disk, obliterating any existing GPT or MBR
2. disklabel(8)
   2.1 obtained the default disklabel
   2.2 initialized the partition configuration with -A and -T
   2.3 wrote the disklabel into the DOS_LABELSECTOR block of the OpenBSD partition
3. kernel
   3.1 used the GPT or MBR to find the OpenBSD partition
   3.2 read the disklabel from the OpenBSD partition
   3.3 validated the disklabel with checkdisklabel()
   3.4 **ignored** any other GPT/MBR information

OpenBSD Takes Control

disklabel(8) is used to create, examine and modify the on-disk struct disklabel

- uses DIOC**W**DINFO (also used by newfs(8) and growfs(8))
- up to 15 user defined partitions
- the 16th partition ('c') is managed by the kernel and cannot be modified
- fstab(5) entries specify which partitions the kernel mounts at startup
- fstab(5) entries can be generated with -F or -f
- boundstart (DL_GETBSTART(), boundend (DL_GETBEND()) are default limits for partitions

Recent disklabel(8) changes

- template (-T) files have new keyword “raid”
- garbage collected struct disklabel fields d_bbsize and d_sbsize
- no longer display or maintain struct disklabel field d_drivedata
- default partition sizes updated

Job done?

OpenBSD Takes Control

Well . . .

- the modern world has become complicated
- UEFI booting
- new platforms (e. g. arm64, riscv64)
  - provide disk images to initialize hardware
  - have proprietary **required** partitions
  - assume GPT information on the disk size does not matter
  - store information in the EFI Sys partition, e. g. firmware updates

OpenBSD Takes Control

These new constraints drove many recent changes.

- fdisk(8)
  - add -A to auto-allocate GPT free space while preserving "protected" partitions
  - add -b to create a "boot" partition in addition to the OpenBSD partition
- readdoslabel()
  - GPT validity checks relaxed
  - GPT OpenBSD partitions treated like MBR OpenBSD partition, i. e. size doesn't matter
  - prevents overwriting in-use data with the disklabel
- install scripts
  - create larger EFI Sys partitions where 960 blocks are no longer enough
  - can create more softraid(4) configurations
  - consistently use fdisk -b

Job done?

Booting

Well . . . you may want to boot OpenBSD from the device

Booting

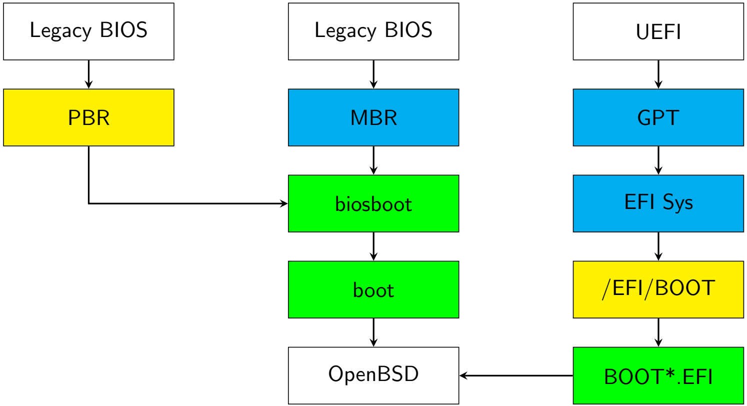

Figure 6: Booting

Booting – Legacy BIOS

PBR

- installboot(8) copies it into place
- BIOS executes PBR code which invokes biosboot(8)

MBR

- fdisk(8) installs boot code into the MBR
- BIOS executes boot code
- boot code loads biosboot(8) from the OpenBSD partition

Booting – Legacy BIOS

biosboot(8)

- /usr/mdec/biosboot patched by installboot(8) to know where the file /boot is **at the time installboot(8) is run**
- written by installboot(8) into **the first block of the OpenBSD partition**
- executes /boot

/boot

- loads the kernel

Booting - Legacy BIOS

Recent /usr/mdec/mbr changes

- only installed when MBR boot code is required (i386, amd64 and landisk)
- partition information removed
- remove “shift to force CHS” mode

Booting – UEFI

- fdisk(8)
  - allocates EFI Sys partition with -b
- installboot(8)
  - formats EFI Sys partition with -p
  - creates /EFI/BOOT directory
  - copies BOOT*.EFI file(s) to /EFI/BOOT/
- BOOT*.EFI
  - loads the kernel
  - is the *default* EFI executable
  - OpenBSD does *not* insert a Bootloader entry into the NVRAM array

Booting – UEFI

Recent fdisk(8) changes

- safely auto-allocates space with -A
- safely allocates boot partition with -b

Recent install scripts changes

- create larger EFI Sys partition when required
- improved support for softraid(4) installations

Booting – UEFI

Recent installboot(8) changes

- preserves contents of existing EFI Sys partition
- prepares the MD “boot” partition with -p
- adopting more EFI smarts
- softraid(4) installations

Booting

Job done!

Miscellaneous Tricks and Traps

Writing a disklabel to a disk **without** an OpenBSD partition

- GPT sector gh_lba_start + DOS_LABELSECTOR blocks
- non-GPT sector 0 + DOS_LABELSECTOR blocks
- readdoslabel() WON'T allow writedisklabel() to write in a non-OpenBSD partition
- readdoslabel() will look there **after** checking for an OpenBSD partition

Miscellaneous Tricks and Traps

Writing a disklabel to a disk **with** an OpenBSD partition

- written to block DOS_LABELSECTOR of the OpenBSD partition
- the DOS_LABELSECTOR block *must* **NOT** *be otherwise used!*
- FFS filesystems have at least BBSIZE (8K) bytes reserved for that purpose

Miscellaneous Tricks and Traps

Kill a GPT

- use ‘fdisk -i’
- dd’ing zeros into the first few sectors is **not** sufficient

Miscellaneous Tricks and Traps

Rediscovering a disklabel by adding/removing OpenBSD partition

- changing the block addresses readdoslabel() checks, by adding, moving or removing an OpenBSD partition will **not** remove the previous disklabel

Miscellaneous Tricks and Traps

The softraid(4) hack – 225 partitions

1. create disklabel with 15 RAID partitions
2. configure each partition as a RAID0 device
3. each RAID0 device has its own disklabel with 15 configurable partitions

Future Development

Time for something new and improved?

Random selection of ideas that have been proposed

- More partitions
- 64-bit offset/size values
- More spoofed partitions
- Move MBR code insertion into installboot(8)
- More EFI magic
- separate in-kernel vs on-disk disk information
- eliminate mixing of sector and block values in user input and display
- multiple OpenBSD partitions
- replace list of protected GPT partitions with list of editable partitions
- fixed endian for fields
- nuke "expert" mode(s)
- stop supporting old 32-bit partition descriptors

New horizons await!

Conclusion

Conclusion

With a little care and meticulous planning OpenBSD can turn those bags of blocks into whatever type of useful device you need.

Thank you for listening.

Questions?

Appendix

disklabel -d

```
# disklabel -d
# /dev/rsd2c:
type: SCSI
disk: SCSI disk
label: SD/MMC 7MKHS
duid: 0000000000000000
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 255
sectors/cylinder: 16065
cylinders: 7620
total sectors: 122419200
boundstart: 64
boundend: 122419167
drivedata: 0

16 partitions:
#                size           offset  fstype [fsize bsize   cpg]
  c:        122419200                0  unused
```

Raw blocks

```
#include <sys/param.h>  /* DEV_BSIZE */
#include <sys/ioctl.h>
#include <sys/disklabel.h>
#include <sys/dkio.h>

#include <err.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void usefulwork(int, uint8_t *, size_t);

void
usefulwork(int f, uint8_t *sec, size_t sz)
{
        if (lseek(f, sz, SEEK_SET) == -1 ||
            read(f, sec, sz) == -1 ||
            lseek(f, 0, SEEK_SET) == -1 ||
            write(f, sec, sz) == -1)
                err(1, "No useful work accomplished");
}
```

Raw blocks

```
int
main(void)
{
        struct disklabel dl;
        uint8_t *sec;
        int f;

        if (pledge("stdio disklabel unveil rpath wpath", NULL) == -1 ||
            unveil("/dev/rsd2c", "rw") == -1 ||
            (f = open("/dev/rsd2c", O_RDWR)) == -1 ||
            ioctl(f, DIOCGPDINFO, &dl) == -1 ||
            (sec = malloc(dl.d_secsize)) == NULL) ||
            pledge("stdio", NULL) == -1)
                err(1, "setup failed");

        usefulwork(f, sec, dl.d_secsize);

        free(sec);
        close(f);
}
```

Disklabel Contents

struct disklabel is defined in /usr/src/sys/sys/disklabel.h, which is installed into /usr/include/sys/disklabel.h.

LABELOFFSET bytes
78 bytes of label information
70 bytes of device information
256 bytes of partition information
$512 - (LABELOFFSET + 404)$ bytes

Figure 7: Disklabel Contents

Disklabel Format

Figure 8: Disklabel Format

Disklabel Partition

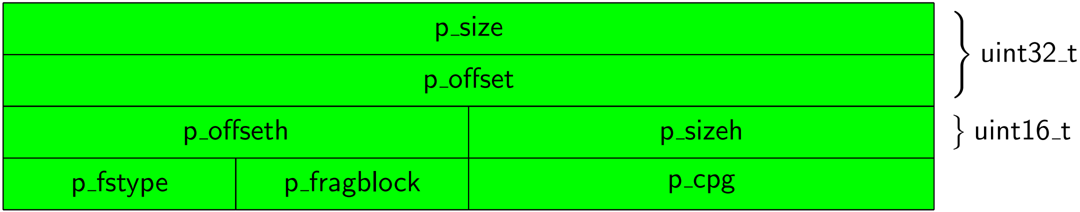

Figure 9: Disklabel Partition

GPT Contents

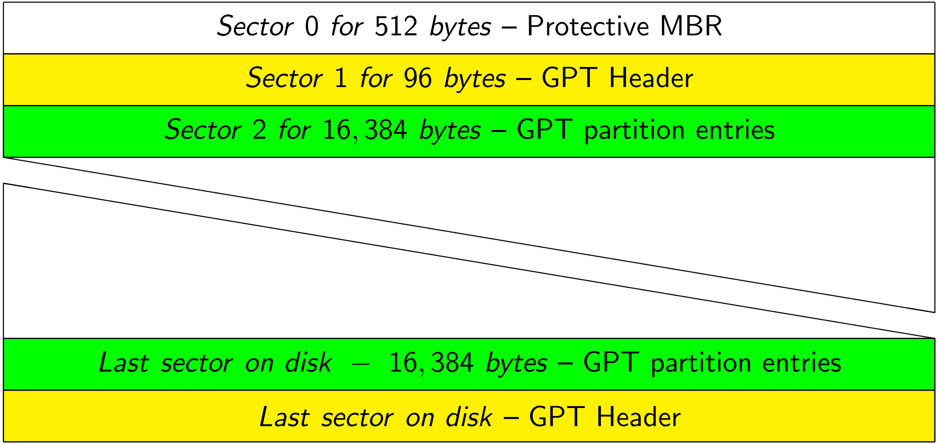

Figure 10: GPT

GPT Header Format

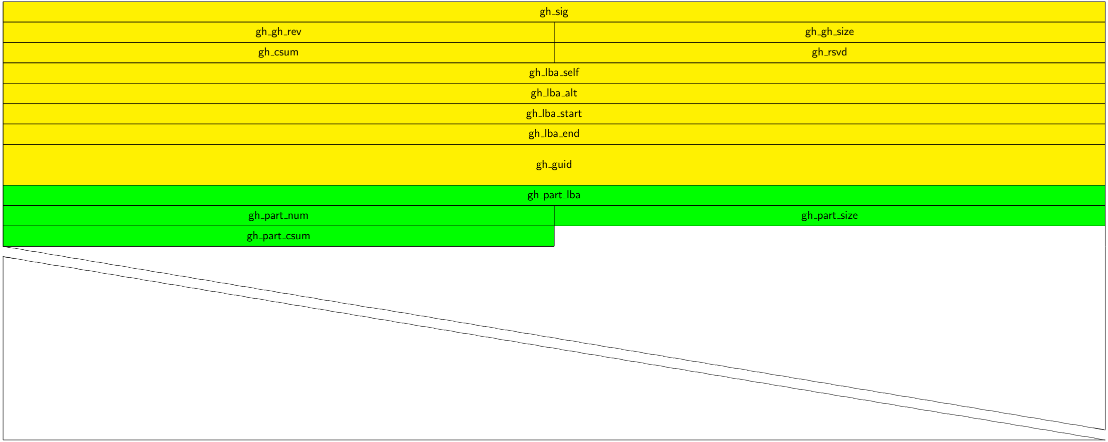

Figure 11: GPT Header format

GPT Partition Entry Format

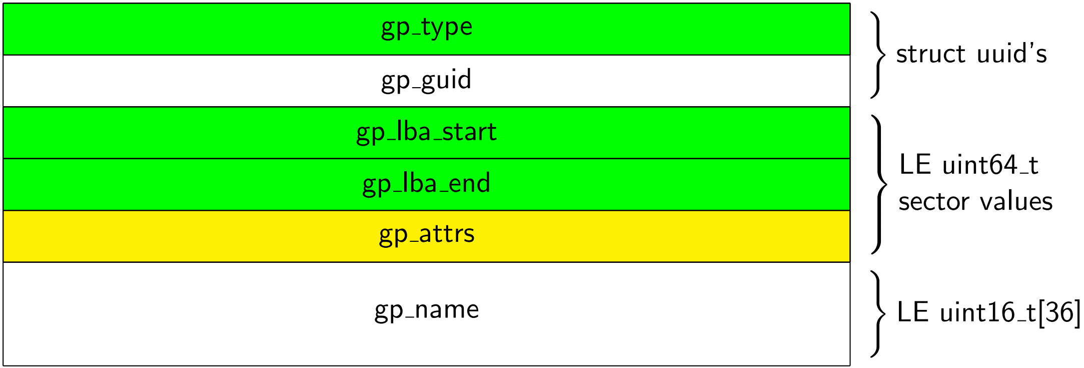

Figure 12: GPT Partition Format

MBR Contents

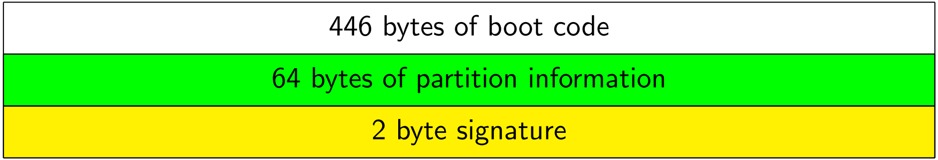

Figure 13: MBR Contents

MBR Format

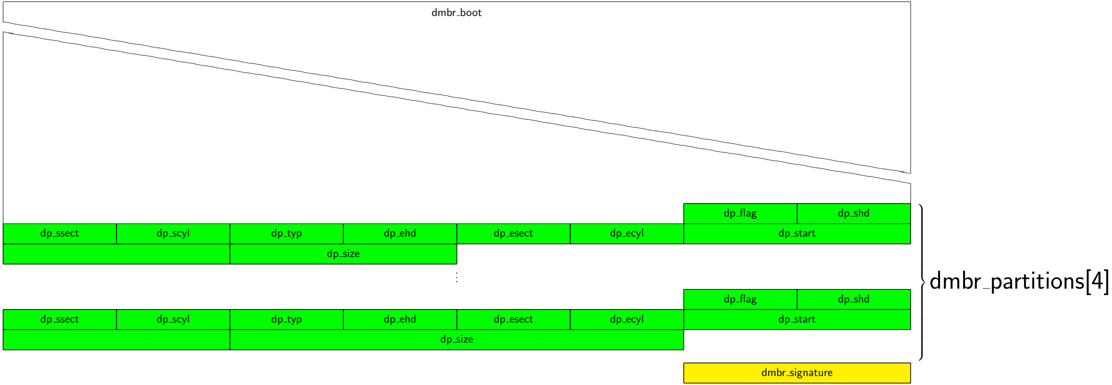

Figure 14: MBR Format

<table>
<tr><td>dp_flag</td><td>dp_shd</td><td>dp_ssect</td><td>dp_scyl</td><td></td></tr>
<tr><td>dp_typ</td><td>dp_ehd</td><td>dp_esect</td><td>dp_ecyl</td><td></td></tr>
<tr><td colspan="4">dp_start</td><td rowspan="2">LE uint32_t</td></tr>
<tr><td colspan="4">dp_size</td></tr>
</table>

Figure 15: MBR Partition

PBR

Jump Instruction e9:XX:XX or eb:XX:90
8-byte OEM Name
13 byte DOS 2.0 BIOS Parameter Block
0 to 66 bytes various other BIOS Parameter Block versions
Boot code
1-byte Physical Drive # (DOS 3.2 to 3.31)
2 byte signature

Figure 16: PBR Contents

BPB

Bytes per Sector ($(1 \ldots 8) \cdot 512$)	
Logical sectors per cluster	
Reserved sectors	
FATs (1 or 2)	
Root Directory Entries	
Total Logical Sectors	
Media	
Sectors Per FAT	

Figure 17: BPB Format