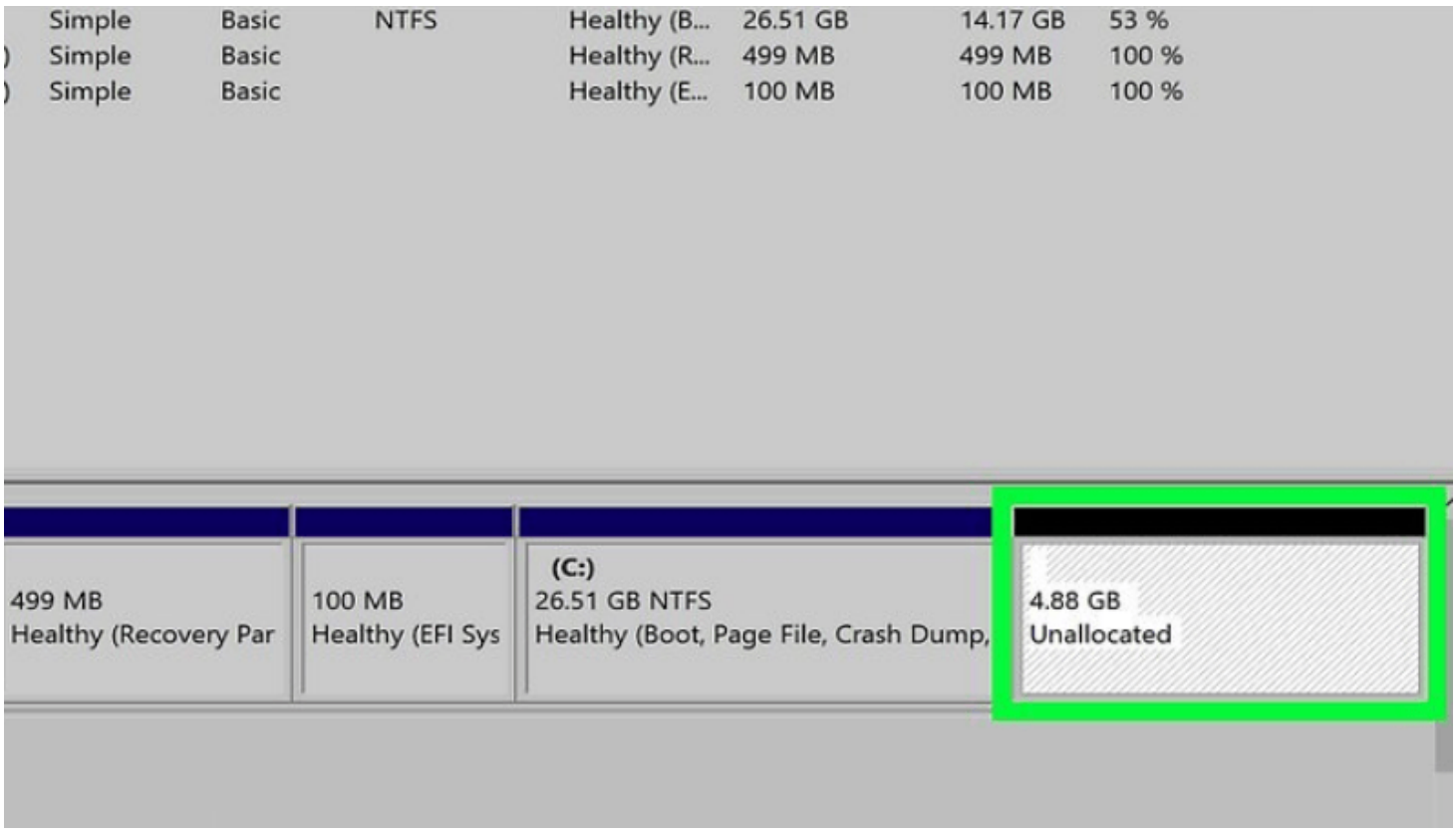
Post 1

Friday, January 25, 2019 12:11 AM



BOOT PROCESSES/PARTITIONS/VOLUMES/HW

BOOT PROCESSES/PARTITIONS/VOLUMES/HW

September 24, 2018 Moo Comments 0 Comment

ROM	**Read-Only Memory**: Permanently holds data: Nonvolatile: Data remains when system off • **Ideal for:** Files containing start-up config settings/code needed to boot machine
RAM	**Random Access Memory:** Temporary data storage for code, settings, etc. • Volatile: On power loss, data lost | **NVRAM**: Nonvolatile form of RAM
Power Supply	Transforms supply voltage [120/240 VAC] to current flows req by components: 3.3/5/12: PS ATX MB
Motherboard	CPU socket, BIOS, CMOS, RTC, RAM, IDE/SATA/Floppy/SCSI/AGP/PCI/PCIE, NIC, video, sound, FireWire
CPU	Massive array of transistors in microscopic layers: Data processing, interprets/exe instructions **Heat sink:** Draws heat away from CPU: Thermal compound/high conductance material
HDD	**Storage:** Series of thin platters revolving at 4,800-15K RPM: Boot files, OS, programs, data • Platters (magnetized) accessed by heads moving across surfaces as they spin • Heads read/write/detect/create microscopic changes in polarity

	• Positive changes: 1 | Negative changes: 0 **Addressing scheme:** Various loc where data stored can be located for reads/writes • Originally CHS system: Cylinder, Head, Sector **Sector:** Smallest amount of space a drive can be written to at a time: 512 bytes that can be used by OS **Tracks:** Each side of platter is fmt w/series of concentric circles • Sectors contained in tracks: Originally each track contains the same # of sectors **Cylinder**: Logical construct: Point on all platters where heads align along a vertical axis • Passes through the same sector number on all platters 2 heads for each platter (one for each side: 0/1): • Depending on number of platters present: Heads will be numbered **Determine number of bytes present on HDD: C x H x S x 512 = Total storage bytes** • Cylinders x Heads x Sectors per track x 512 [constant that represents number of bytes in a sector] • **Holds true if**: Number of sectors per track remains same for all tracks: Lower-capacity HDD's **Outer tracks:** Can always hold more data than inner tracks/contain wasted storage space **ZBR: Zoned-Bit Recording:** Number of sectors per track varies in zones • Outer zones containing more sectors per track than inner ones • To address larger-capacity HDD's: New addressing scheme developed **LBA: Logical Block Addressing: Sectors addressed by sector number** • Starts w/sector 0 • HDD's electronics translate sector number to CHS value understood by drive **Determine storage capacity: Total LBA sectors x 512 = Total storage capacity in bytes** HDD's can be ATA/PATA/SATA/SCSI
SSD	**Solid State Drive:** Does away w/moving parts: Data stored on NAND mem chips: USB: Persistent/nonvolatile **Hybrid drive:** Traditional HDD w/SSD

SCSI/IDE/SATA/SAS

SCSI	**Small Computer Systems Interface controller:** High-speed/performance: Devices w/high I/O: Scanners/HDD's • **SCSI BIOS:** Intelligent: Queues read/write requests in manner that improves perf • Doesn't use master/slave pin configs like IDE: Assigned by numbers often set by pinning jumpers
IDE	**Integrated Drive Electronics controller:** Generic term: Any drive w/own integrated drive controller • **Originally 3 types:** 1 survived: **ATA: Advanced Technology Attachment** **2 connectors on MB: Primary/2ndary IDE:** Each capable of 2 IDE devices: HDD/CD/DVD • Of 2 devices on same ribbon: 1 master/1 slave w/jumpers on pins to designate which **CS: Cable Select** method of pinning: Assignment done auto: Cable that supports CSEL signaling
SATA	**Serial Advanced Technology Attachment controller:** Serial circuitry: Allows data to be sent at 150MBps • SATA II: 2002: Buffer-to-host transfer rates of 300MBps: No pinning: Found on most modern MB's
SAS	**Serial Attached SCSI:** Backwards compatibility w/gen II SATA drives: Supports of 6-12GBps

RAID

RAID	**Redundant Array of Inexpensive Disks:** Array of 2/more disks combined to increase performance/fault tolerance
RAID 0	**Data striped over 2/more disks:** Increases performance by reducing read/write times: If any disk fails: All data lost
RAID 1	**Data mirrored over drives in array:** Doesn't increase performance: Creates redundant data: Increase in fault tolerance
RAID 5	**Data stored on 3 drives:** Other configs can be made: Data striped over 2 drives | Parity

	stripe created on 3rd • If any drive fails: Can be rebuilt from data of other 2: Fault tolerance/increased performance
RAID 0 + 1	**Data stored on 4 drives:** 1 pair for striping data | Other a mirror of striped pair: High performance/fault tolerance • **0+1:** Stripe built before the mirror | **1+0:** Mirror built before the stripe

Drives/Ports

Floppy	1.44MB data: Forensic examiners use them as boot drives to boot systems for DOS acquisitions
CD	Laser beams to read indentations/flat areas as 1/0's: Data fmt into continuous spiral from center
DVD	Laser beams w/DVD's shorter wavelength, creating smaller pits/lands: Depressions/elevations on surface
USB Port	Connected to USB controller w/pins for 4 conductors surrounded by shielding: • **Cable power** • **Data negative** • **Data positive** • **Ground**
IEEE 1394	**FireWire:** High-speed serial I/O standard: Plug & Play on parallel w/USB: 2 speeds • **1394a:** Original version: 400Mbps: Similar to USB ports: 1 end slightly rounded/pointed • 6 wires/pins: ○ 2 pairs of clock/data lines ○ 2 for power (positive/negative) • **1394b:** 800Mbps: Rectangle shaped w/dimpled inset ○ 9 conductors: 2 of 3 additional conductors used for shielding (A Shield/B Shield) ○ Shielding for improved signal/higher xfer rate: 786.432Mbps • Allows daisy chaining devices w/63 node max
Thunderbolt	Serial connection int for peripherals being connected to computer via expansion bus • Up to 7 devices can be daisy-chained: 2 can be high-resolution monitors using DisplayPort • 10Gbps bidirectionally

The Boot Process

POST	**Power On Self-Test:** Power supply checks voltage/current levels to see if acceptable • Current from PS follows predetermined path to CPU • Residual data in CPU erased via the signal resetting the **program counter** register ○ AT/later computers: Signal **F000:** Value address of next piece of code to be processed **F000:** Corresponds to beginning of a boot program in ROM BIOS
Bootstrap	**Boot program in ROM BIOS initiates a series of system checks** • Runs set of instructions/code intended to check CPU/POST • Matches against set of values stored in BIOS chipset **If good:** Signals are then sent from the CPU to the system bus to ensure proper functioning
RTC	CPU tests RTC/system clock: RTC keeps all electrical component signals in sync
Video	POST tests video: Video mem/signals sent by the device are tested • Video's BIOS added to overall BIOS stored in RAM: User starts to see things on screen
RAM	Main mem tested: Data written to RAM: Read/compared to original data sent: If match, passes

	• User may see countdown as volume of RAM tested
Keyboard	CPU checks if keyboard is properly attached/any keys are pressed: If something on KB during post: Beep/screen msg
Drives	CPU sends signals to specific buses to determine which drives (Floppy/CD/HDD/etc.) available to system
Comparison	Results of POST compared to expected system config stored in CMOS in RTC/NVRAM • If doesn't match: User given chance to update through setup utility
Components	Other system components that contain their own BIOS are loaded into overall BIOS RAM: Ex: SCSI BIOS, Plug & Play • System ready to load OS **Bootstrap code finishes 1 of 2 missions during POST:** Searches for avail drives for OS according to boot sequence **ROM BIOS boot code looks to 1st sector of default boot HDD for MBR** • Reads it into memory/tests for valid signature ○ **Signature: 55AA** or **0x55AA** ○ Located last 2 bytes of this sector ○ If doesn't match, error returned: Otherwise continues
MBR	**Master Boot Record:** Contains 64-byte partition table located at **byte offsets 446-509** • Each of up to 4 partitions described by 16 bytes in table • MBR reads its own partition table for boot indicator byte **Boot indicator byte:** Marks 1 of the partitions as active: 1 must be active to boot: Can't have more than 1 active • MBR reads **VBR: Volume Boot Record:** of active partition, loads it into mem, carries signature test out • **Looks for the last 2 bytes** of VBR to read as **55AA** • If signature test fails: Error msg | If passes: VBR code executes/runs • VBR code/program: Searches for and runs OS on that volume After: Depends on OS loaded

DOS boot:

IO.SYS	Code in VBR locates/executes initial/primary sys file: **IO.SYS**[**IBMBIO.COM** for IBM] • **SYSINIT**: Subroutine of **IO.SYS:** Also runs as part of execution • Code copies itself into highest region of contiguous DOS memory • Next it locates/reads MSDOS.SYS: Copies it into low memory • Overwrites portion of IO.SYS in low memory ○ This contains the initialization code: SYSINIT is no longer needed there
SYSINIT	**SYSINIT** runs **MSDOS.SYS**: Initializes basic device drivers/checks status of system equipment • Resets the system/initializes various devices attached to it • Sets default parameters • Works w/BIOS to manage files/execute code/respond to HW signals
DOS	**DOS FS running/active: SYSINIT** resumes control of boot process • **SYSINIT** reads **CONFIG.SYS** file as many times as statements w/in it needed to process ○ **DEVICE** statements processed 1st in order appeared ○ **INSTALL** statements processed in order of appearance **Once done:** • **SHELL statement present?** Runs • **SHELL statement not present?** Default shell w/default params run **COMMAND.COM** • **SYSINIT** now complete • **COMMAND.COM** written into section of mem previously occupied by **SYSINIT**
AUTOEXEC.BAT	**If present: COMMAND.COM** runs it • Each cmd in batch file executed

	• If 1 of batch cmds calls for launching app/shell: User presented w/prompt ○ Otherwise: When a bunch of batch cmds executed: User sees blinking cursor at DOS prompt **If no AUTOEXEC.BAT file present:** • **COMMAND.COM** runs **DATE** and **TIME** commands • Displays copyright message • User shown a blinking cursor at DOS prompt

Windows NT/2000/XP Boot:

VBR	**Code in VBR locates/runs primary sys file** **NT: NTLDR (NT Loader):** Places processor in *protected* mode: Starts FS: Reads **BOOT.INI** file **Dual booting w/non-NT type (Linux)? BOOTSEC.DOS** runs • If SCSI drives attached to system: **NTBOOTDD.SYS** containing SCSI drivers executes
NTDETECT.COM	Executes/searches sys for installed HW: Passes config data to **NTLDR** • If more than 1 HW profile exists: **NTDETECT** determines correct profile for HW
NTOSKRNEL.EXE	Config data obtained by **NTDETECT** Passed by **NTLDR** to **NTOSKRNL.EXE** • Loads kernel: **HAL: HW Abstraction Layer**: System registry info
Drivers	**Loads drivers/code for networking (TCP/IP):** Services config to run at start-up load/run • Logon services: Provides user w/prompt • When user logs on: Current config status considered good/updated into system registry • **Last Known Good Config**
Logon	**Device detection takes place simultaneously:** If new devices detected: • Plug & Play assigns sys resources, extracts drivers from **DRIVER.CAB**: Completes config/mnt of devices **If drivers can't be found:** User has GUI that allows them to interact w/HW/SW

Vista/Server 08/7

Boot	Boot code: **VBR loads BOOTMGR**: **Windows Boot Manager** instead of **NTLDR** • NTLDR reads BOOT.INI file: **BOOTMGR** reads **BCD** file located in **Boot folder:** Root of sys volume ○ BCD: DB of boot-time config data ○ File fmt same as registry hive file **EnCase can mount:** Right-click/View File Structure • **BOOT.INI**: Menu entries presented by NTLDR • **BCD**: Menu entries presented by Win Boot Manager Boot options include: • Vista: Booting prior ver of NT: Resuming from hibernation: Loading/executing volume boot record • Previous Win vers: NTLDR: loaded kernel: NTOSKRNL.EXE passing boot config info in process **Windows Vista+/Win Boot Manager:** • Invokes **WINLOAD.EXE** > In turn loads **NTOSKRNL.EXE** > Boot-class device drivers

Partitions/Volumes

Partition	**Consecutive sectors w/in volume:** Sectors addressable by single FS specific to/contained w/in that partition
Volume	**Addressable sectors used by OS/app to store data:** Addressable sectors in volume don't have to be consecutive: • When volume consists of single partition: Two are functionally same • When volume spans more than 1 partition/drive: Difference noted

	Logical storage units: Assigned drive letters by OS: • Most can support up to 24 volumes using letters C-Z reserving A/B for floppy • If single physical HDD were installed: Drive could be partitioned into 24 volumes • Partitioning table in MBR only permits 16-byte entries for 4 partitions **How could a system support 24 logical volumes?** Extended partition system • 1 of 4 defined partitions in MBR partition table can be an extended • Disk space assigned to extended partition is subdivided into logical volumes by OS • Each sub-partition of extended volume contains partition ○ Optionally points to another partition table in another sub-partition **Nesting of sub-partitions w/in extended partition:** Can extend as far as letter assignments permit **Each nested sub-partition:** Partition table describing itself/pointing to next level down until done • **In theory:** Could encounter upper limit of 24

Partition types encountered usually specific to OS
5th byte w/in each 16-byte partition entry: Offset 446-598 MBR:

- Determines partition type/FS for each defined: Also true for partition tables w/in extended partition/sub-partitions

1st byte of each of 4 partition table entries: Determines which partition active/boot partition

Only 1 partition table can be active

- **80**: Active partition
- **Other 3 partition entries:** If defined: **00** for 1st byte in respective entries

Partition Table Fields Defined

Offset (Dec)	Name	Length	Description
446	Boot Byte	1 byte	**Boot status: 80**: Active/bootable|Otherwise **00**
447	Starting Head	1 byte	**CHS:** Start head/side of partition
448	Starting Cylinder/Sector	2 bytes (16 bits)	**CHS: Total:** 16 bits **Starting cylinder:** 10 bits **Starting sector:** Next 6 bits
450	Partition Type	1 byte	Partition type/FS
451	Ending Head	1 byte	**CHS:** Ending head/side of partition
452	Ending Cylinder/Sector	2 bytes (16 bits)	**CHS: Total:** 16 bits **Ending cylinder:** 10 bits **Ending sector:** Next 6 bits
454	Relative Sector	4 bytes (32 bits/Dword)	**LBA:** Number of sectors before partition • Starting sector of partition
458	Total Sectors	4 bytes (32 bits/Dword)	**LBA:** Total number of sectors in partition

FAT12/16/32/NTFS/exFAT partitions/FS used when running various flavors of Win:

- Can be created by utilities that ship w/Win: **FDISK, DISKPART, Disk Manager**

Other partition types:

- Linux Native (EXT2/3/4/Reiser)
- Swap partitions: Solaris (UFS)
- Mac OS X (HFS+)

FDSIK: format cmd for a FAT12/16/32 partition:

1. Disk scans for errors: Bad sectors marked
2. Drive heads placed at 1st cylinder of partition: DOS VBR written
3. FAT1 written to Head 1 Sector 2: FAT1/2 Is written: Entries in FAT (File Allocation Table) mostly null: Bad clusters marked
4. Blank root dir written
5. **/s param selected?** Sys files transferred
6. **/v param selected?** User prompted for a volume label

Following written during FDISK/Disk partitioning process:

- MBR: Contains MBR booting code
- Partition table entries
- MBR signature
- During high-lvl fmt process: VBR is typically written along w/other FS features

File Systems

FS	**Method of storing/retrieving data on sys that allows for a hierarchy of dirs/subdirs/files** • Must be consistent bet systems using same FS: Structural/org files/data/user data • Contained w/in partition: Data/files that describe layout/size of FS • How large data storage units (clusters, blocks, etc.) will be **Allocation units/clusters/blocks: Data storage units:** Groups of sectors that hold content data • Filenames have to be linked to actual data OS can locate **Metadata:** Data w/in data describing data: Must be attribute to point where data starts • Done via a dir entry (FAT) or entry in file table such as **MFT: Master File Table** in NTFS systems • **Data may be larger than 1 allocation unit can hold:** Must track containing data storage units ○ **FAT:** Clusters linked together in file allocation table ○ **NTFS:** Clusters containing data described by data runs in MFT **Any FS must have a system that tracks allocation unit usage/availability** • W/out this: Data could be overwritten ○ **FAT:** File allocation table ○ **NTFS/Other:** Single-purpose **VBM: Volume Bit Map:** Array of bits w/each representing allocation unit ▪ **0:** Available for use ▪ **1:** Allocated

From <https://www.piratemoo.net/moosings/winfor/boot-processes-partitions-volumes-hw/>

Post 2

Friday, January 25, 2019 12:11 AM

FAT BASICS: ADVANCED WIN FORENSICS: CH 2 NOTES

September 30, 2018 Moo Comments 0 Comment

FAT Basics: 1st major component:

Directory entry	**All vers FAT:** Every file/dir referenced/described in separate dir entry • 32 bytes • File/dirs name | size in bytes | starting extent (beginning cluster) • **Metadata:** Created | last accessed | last-written timestamps | etc. **No data content exists in dir entry** • Data content stored in **clusters:** data allocation units **Tracks only starting cluster:** Doesn't track other clusters used by file
Clusters	**1/more sectors: Smallest unit in which a file/dir can be stored** • If file size exceeds amt contained in 1 cluster: Assigns as many clusters as needed
FAT	**File Allocation Table** • Tracks allocation status of clusters (when more than 1 cluster used as well) • Ensures OS stores data in clusters avail/files not overwritten • Tracks bad clusters: Marks them so won't be used

Comes in 3 ver: FAT 12/16/32: MBR imposes limit of 67,092,481 clusters

FAT12	**12-bit entries:** Each 12-bit sequence represent cluster: Starts at 0/Ends: Last cluster in volume • Theoretical max: 4096: Certain values reserved: 4084 largest # of clusters supported
FAT16	**16-bit entries**: Supports up to 65,524 clusters
FAT32	**32-bit entries**: Only 28 used: Supports 268,435,445 clusters

Max # of sectors support by FAT types

FAT	Max # of clusters supported
FAT12	4,084
FAT16	65,524
FAT32	67,092,481

Physical layout of FAT: 3 major components

1. Reserved area: Volume boot sector
2. FAT area
3. Data storage area: Dir entries/data content

Reserved Area: Consists of volume boot sector/boot sector

- Size of reserved area defined w/in boot-sector data: FAT12/16: 1 sector for usually

FAT32: Way more data: 7 sectors length (0,1,2: Backup: 6,7,8): Reserved: Usually 32 sectors length

- Backup copy of boot sector unique to FAT 32

- Runs if volume sector 0/boot corrupted: Offsets 50-51
- MS: Standards call for being loc at volume sector 6 at backup loc
  - If loc relied on data in sector 0: Corrupted sector could destroy ability to find backup boot sector

FSINFO	**File System Information: Data structure** • Sector immediately following backup boot: 7 • Provides info to OS about # of free clusters avail to sys/loc of next free • Backup copy doesn't match primary
VBR	**Volume Boot Record: AKA Volume Boot Sector:** 1st sector of logical volume: • Reserved FAT12/16: Often only sector in reserved **4 distinct segments:** 1. **JMP instruction to boot code:** First 3 bytes • **BPB: BIOS Parameter Block:** ▪ FAT12/16: Bytes 3-61 ▪ FAT32: Bytes 3-89 • **Boot code/error msgs** ○ FAT12/16: Bytes 32-509 ○ FAT32: Bytes 90-509: Continues at sector 2 bytes 0-509 1. **Signature:** Bytes 510-511: 0x55AA

Jump instructions: Tells machine where to find beginning OS bootstrap code

Bytes	**0-2 | 3-35:** Same structure/purpose for all 3 ver of FAT **36-61:** FAT 12/16: Unique purpose **36-89:** FAT32: Unique purpose **510-511:** Boot sector signature

BPB: DB w/defined fields: Params of partition/FS w/in
Volume Boot Sector Fmt: FAT 12/16 FS:

Byte Offset (dec)	Name/Description
0-2	**ASM JMP** instruction to bootstrap code found in sector: **0xEB3C90**
3-10	**OEM ID in ASCII:** OS that fmt vol: Win95: MSWIN 4.0 Win2K/XP/Vista: MSDOS 5.0 Linux: mkdosfs (mkfs cmd used to create)
11-12	**Bytes per sector:** 512: 1024/2048/4096 can happen
13	**Sectors per cluster:** Value power of 2 greater than 0: 1/2/4/8/12/16/32/64
14-15	**Number of sectors in reserved area**
16	**Number of FATs**: Typically FAT1/FAT2: • FAT2: Duplicate of FAT1 redundancy
17-18	**Max number of 32-byte dir entries in root:** FAT12/16: 512 | FAT32: 0
19-20	**Number of sectors in partition:** 16-bit int: 0? Number exceeds 65,536 • Offset 32-35: 32-bit int describing
21	**Media descriptor:** Duplicated 1st entry of FAT: **Cluster 0:** Would be used if addressable cluster at 0/1: Value used instead

	• **0xF8:** Non-removable media (HDDs) • **0xF0:** Removable media
22-23	**Number of sectors used by each FAT** (12/16): 16-bit int: FAT32: 0
24-25	**Sectors per track for interrupt 13h:** 63 for HDD
26-27	**Number of heads for interrupt 13h:** 255 for HDD
28-31	**Number of hidden sectors before start of partition:** 63 for 1st volume on disk
32-35	**Number of sectors in partition:** 32-bit int: • 0? Number doesn't exceed 65,536: 16-bit int: Bytes 19-20 • Only 1 of 2 (19-20/32-35) NOT both must be set to 0
36	**Interrupt 13h drive number** • **0x00:** Floppy • **0x80:** HDD
37	Not used: Exception Win NT: Usually 0
38	**Extended boot signature:** Determine validity of 3 fields that follow • **0x29:** Next 3 fields present/valid • **0x00**: Otherwise
39-42	**Volume serial number:** w/field that follows to track volumes on removeable media • Some OS: Value generated using date/time seed value at time of creation
43-53	**Volume label ASCII:** Given by usr at time of creation • Limit: 11 bytes: Should match value in root dir of FAT 12/16 • If none given: **NO NAME** should appear
54-61	**FS type at time of formatting**: ASCII as FAT/FAT12/FAT16: Not used after fmting: Could be altered
62-509	Bootstrap program code/error msgs
510-511	**Signature value**: 2 bytes: **0x55AA**

Volume Boot Sector Fmt: FAT32 FS (Differences only)

Byte Offset (dec)	Name/Description
36-39	**Number of sectors used by one FAT** on FAT32: 32-bit int: Bytes 22-23: Must be set to 0 for FAT32
40-41	**Series bit-field values:** Describe how multiple FATs written to • **Bit 7 off (value 0)?** FAT duplicated • **Bit 7 on (value 1)?** Duplication disabled/only FAT referenced in 0-3 bits active • **Bits 0-3:** Valid only if bit 7 on/duplication not happening • **Bits 4-6/8-16:** Reserved • **Default: 0x0000** (FAT 1/2 replicated)
42-43	**Major/minor ver numbers of FAT32 volume:** High byte: Major | Low byte: Minor |Expect 0x00/0x00
67-70	**Volume serial number:** Used w/field: Follows to track volumes on removable media
71-81	**Volume label ASCII:** Given by user at creation: 11 bytes: Should match value in root • None given? NO NAME
82-89	**FS type at time of formatting:** ASCII as FAT32: Not used after fmt/could be altered
90-509	Bootstrap program code/error msgs: FAT32: Continues at sector 2: Bytes 0-509

FAT Area: File Allocation Table: Begins: Sector that follows last sector of

reserved area: Default: 2 FATs (1 & 2) in FAT FS

- Exact # of FATs/size of/total size for all FATs specified in boot sector

FAT: 2 purposes:

1. Account for allocation status of cluster
2. To find clusters that follow starting cluster for any given file/dir

FAT1: Starts immediately following reserved sectors

FAT2: Duplicate of FAT1: Default config/immediately follows FAT1: Not required: Could be config to only have 1 FAT

- FAT1/2: Equal in size/duplicate img of each other

Each cluster on FS: Represented sequentially starting w/0 in table

- Each entry 12 bits (16 for FAT16/32 for FAT32 w/28 used and 4 reserved)
- Cluster 2: Starts immediately following the EOF allocation tables

FAT entry for nonaddressable cluster 1: Stores value for "dirty status" of FS:

- If improperly dismounted (improper shutdown) Value used to track: Causes OS to prompt usr to check FS on reboot

3rd 16-bit entry in FAT16	**1st addressable cluster:** Certain values present • **Unallocated:** Value 0: Cluster avail for use by OS to store file/dir • **Allocated:** Value: Represented by next cluster used by file (unless last) • **Populated by EOF marker:** Value greater than ○ FAT12: **0xFF8** ○ FAT16: **0xFFF8** ○ FAT32: **0xFFFF FFF8** • **Bad cluster:** Not avail for use by OS ○ FAT12: **0xFF7** ○ FAT16: **0xFFF7** ○ FAT32: **0xFFFF FFF7**

Data Storage Area: Contains clusters used to store dir entries/data

Location of Root dir:

FAT12/16	**Root: 1st part data area:** Follows end of FAT area • Fixed-length: Max 32 sectors: Each dir entry 32 bytes • Number of bytes avail to root: 16,384: **32 sectors x 512 bytes per sector** • Each dir in 12/16 is 512 entries: **16,384 bytes / 32 bytes per entry** • Cluster 2: Immediately follows root dir/all clusters contained w/in defined partition
FAT32	**Overcame limitations of 512 entries in root dir:** Dynamic • NOT fixed: Can be anywhere in data area: Almost always begins at cluster 2 • Cluster 2: Immediately following FAT area

Dir Entries: Critical component of FAT FS: Every file/dir w/in partition: Dir entry exists

- Each entry: 32 bytes: W/in bytes: Name of file/dir/starting cluster/length described
- Parent keeps track of children: If parent del: Children orphaned: Lost files/folders

8 dot 3: DOS naming convention: Max of 8 chars for filename: 3 chars for

extension

- **Long filename?** Attribute set in metadata: Series of 32-byte entries: Precede main entry

Data Structure for FAT dir Entry

Byte offset	Description
0	1st char of filename/status byte
1-7	Chars 2-8 of filename
8-10	3 chars of file ext
11	Attributes
12-13	Reserved
14-17	Created time/date of file: Stored as MS-DOS 32 bit date/time stamp
18-19	Last accessed date: No time stored in FAT!
20-21	2 high bytes of FAT32 starting cluster: FAT 12/16: 0's
22-25	Last written time/date of file: Stored as MS-DOS 32 bit date/time stamp
26-27	Starting cluster for FAT12/16: 2 low bytes of starting cluster for FAT32
28-31	Size in bytes of file (32-bit int): 0 for dirs

Bit flag values for attribute Field at offset 11

Bit flag values (bin)	Description
0000 0001	Read only
0000 0010	Hidden
0000 0100	Sys file
0000 1000	Volume label
0000 1111	Long file name
0001 0000	Dir
0010 0000	Archive

Attribute bit flag values: Can be combined: Long filenames: When exceed length of DOS 8 dot 3 limit/illegal chars:

- Special entries created in dir to accommodate up to 255 chars (including length of path)

When long filename create: 8 dot 3 alias created as filename in 32-byte entry

Done through following scheme (Win9x/ME)

1. 1st 3 chars: Ext used as ext of 8 dot 3 alias
2. 1st 6 chars of filename: Rendered to uppercase: Any illegal DOS 8 dot 3: Converted to underscores
3. 2 chars added to 6: 7 is ~ | 8 is numeral 1 |
   - If another file exists that has same alias: 1 sequences to 2 etc..

Long file name storage scheme

Offset	Description
0	Sequence number used to link together multiple LFN entries • 0xE5: Deleted/unallocated

	• 1st LNF entry above alias 8 dot 3: Contains beginning of LFN: Built on each other until end reached
1-10	LFN chars 1-5 in Unicode
11	Attributes
12	Reserved
13	Checksum
14-25	LFN chars 5-11 in Unicode
26-27	Reserved
28-31	LFN chars 12-13 in Unicode

Viewing Dir Entries w/EnCase: Dir entry raw data can be viewed/analyzed/bookmarked:

- Determine parent folder: Parent folder table pane: Highlight each dir on line
- Text Style tab > Text Styles > New > Name to FAT Dir > Max Size > Set Wrap Length to 32

How File Stored: OS calls for file to be read by filename/path: Path leads to filename stored in dir entry

- Info stored in dir entry allows OS to begin to loc data that constitutes file

Find how many clusters file will occupy: Divide file size by number of bytes in a cluster

- Partial clusters not allowed: Check math: EnCase > File Extents
- Select file of interest table pane > Details viewed in bottom/view pane

Status byte: 1st char of file/dir name in a dir entry

- In use: Unclear: Seen as part of file/dir name
- File/dir deleted: Purpose clear: 1st char of file changed to **0xE5**: Signifies to OS entry del: OS ignores file/doesn't display to usr

Effects of Del/Undeleted Files: Can change 1st char of filename in dir to 0xE5: Undelete reverses process

- EnCase does this auto: If cluster in use by another file: Will report overwritten

Slack Space	**Data bet end of logical file – end of last cluster allocated** • Usually contains data from files that used space before Composed of data from 2 diff sources • When data written to media: Written in blocks of 512 bytes/1 sector • If 1 byte of file data written: Sys must write other 511 bytes to make 512 Before Win95B: Extra data/filler randomly taken from slack: Caused sec concerns • Win95B: Filler became 0's **RAM slack: AKA Sector Slack:** Portion of slack space: End of logical file TO end of sector **File slack: Remainder of slack:** From end of last sector containing logical file TO end of cluster **Entire slack space:** Both RAM/Sector/File slack: Default red color in

	EnCase

Directory Entry Status Byte: 1st byte of the 32 byte dir entry: Contains 1st char of valid file/dir name: **0xE5**

- **0x00**: Entry not used/entries beyond not searched

Dot double dot: 0x2E/0x2E2E: Signature for a dir

- When value of 1st dir entry line begins w/**0x2E (dot)**: Denotes dir entry: Points to self
- 2nd dir line begins w/**0x2E2E (dot dot)**: Points to parent dir
- **cd ..** Change parent: Value in starting sector (offsets 26/27) will be parent dir of where you are

Importance: Can search for folder in unallocated clusters of partition looking for this signature

NTFS Basics:

- Compare to FAT: More robust: Stronger sec/greater recoverability: Better perf w/read/writes
- Support for long filenames: Highly granular of file perms/access control
- Compression of individual files/dirs

WinNT/2000: Provided NTFS as fmt option: Defaulted to FAT unless usr chose otherwise

WinXP onward: Default fmt NTFS

Regardless of name: Must perform basic functions for system: Tracks:

- Name of file/dir
- Point where file starts
- Length of file along w/metadata (time/date stamps)
- Clusters used by file
- Which allocation units (clusters) are allocated/which aren't

When NTFS is formatted: VBR is created and 16 sectors reserved for use: Typically only 8 used for data

- **Bytes 3-6:** 1st sector of VBR will be NTFS
- **FAT32:** Stored backup of VBR in reserved area at beginning of volume
- **NTFS:** Stores backup copy in last sector of partition

Important FS data: Contain in actual files: Uses many sys files:

1. $MFT
2. $Bitmap

$MFT	**Master File Table**: Similar to entry dir in FAT • DB w/entry for every file/dir in partition: Including entry for itself • Entries fixed in length: Almost always 1024 bytes • Each entry has a header: FILE0: Followed by series of attributes • Everything about a file is an attribute including the file itself **Resident data**: If a file is small: Sometimes stored w/in $MFT • Example: Cookies | Avg max length: 480 bytes ○ Varies w/type/length attributes of $MFT **If a file can't be stored in $MFT:** Stored in cluster • In place of resident data: Cluster runs stored
$Bitmap	**1 bit for each cluster in partition:** Similar to FAT 1&2 • Tracks allocation only: Doesn't track cluster runs

	• **Cluster has 0:** Avail for use by system • **Cluster has 1:** Cluster allocated to a file

NTFS System Files

MFT Record #	Filename	Description
0	**$MFT**	Master File Table: Each record is 1024 bytes length
1	**$MFTMirr**	Backup copy of 1st 4 entries of MFT
2	**$LogFile**	Journal file: Contains file metadata transactions used for sys recovery/file integrity
3	**$Volume**	NTFS ver/volume label/identifier
4	**$AttrDef**	Attribute info
5	**$.**	Root dir of FS
6	**$Bitmap**	Tracks allocation status of all clusters in partition
7	**$Boot**	Contains partition boot sector/boot code
8	**$BadClus**	Bad clusters on partition tracked w/this
9	**$Secure**	File perms/access control settings for file sec
10	**$UpCase**	Converts lowercase chars in Unicode by storing uppercase ver of all Unicode chars in file
11	**$Extend**	Dir reserved for options ext

CD FS: 1986: High Sierra: Revised: ISO 9660: Cross-platform capability for CD's bet PC/Unix/Mac:

Limitations at Level 1 implementation:

- Uppercase chars (A-Z/0-9) and underscore permitted in filename
- Names used 8 dot 3 naming convention
- Dir names couldn't exceed 8 chars/couldn't have ext
- Nesting was limited: No deeper than 8 lvls
- Files had to be contiguous

Level 3 interchange rules were allowed: Improvements: 30 chars in filename: Non-contiguous files

Joliet	MS dev Win95: Files/dirs can be 64 chars long (Unicode support)/dirs can have ext/8-lvls subdir barrier gone • Multiple session recording supported • Examining CD created w/Joliet: ISO9660/Joliet dir
UDF	**Universal Disk Format:** Uses packet writing to write data in increments to CD-R/RW • 255 char per filename
Mac	Native HFS fmt on media: Can use on CDs/can't be read by PC's: Hybrid disc often used to overcome issue • Creates both an HFS/Joliet dir: Both point to same set of data
Unix	**Rock Ridge** ext: FS features: Can't be read by PC: Basic ISO 9660 standard

exFAT: 2006: Introduced w/Win Embedded CE 7.0 : Proprietary: Designed for flash media: Often dubbed FAT64

- **File size limit:** 16 EiB (1 exbibyte = 2^60, bytes = 1,152,921,504,606,846,976 bytes)

- FAT32 for years imposed 4GB size limit

Following OS support exFAT: WinXP/Server 03 w/SP2 +patch/Vista SP1/Server 08/7/OS X Snow Leopard/Lion

4 major areas of exFAT

1. Main boot region
2. Backup boot region
3. FAT region (normally only FAT1 unless TFAT config then FAT2)
4. Data region

First logical sector: 0: Partition known as VBR: FS name EXFAT highlighted in view pane of EnCase: Offset 3: Runs 8 bytes

- Main boot region: Logical sector 0: Runs for 12 sectors: Ends at logical sector 11
- Backup boot region: Logical sector 12: Runs for 12 sectors
- FAT 1 after: Offsets 80/84 (4 bytes each)
- Data area follows FAT region: Also defined in VBR

Volume Boot Record: exFAT FS

Offset (dec)	Length	Description
0	3	JMP code
3	8	EXFAT: OEM FS ID
11	53	Must be 0x00
64	8	Partition sector offset (0 for removeable media)
72	8	Total sectors in volume
80	4	Offset to beginning of FAT
84	4	Physical size of FAT in sectors
88	4	Offset to bitmap
92	4	Allocation units in volume (bit count)
96	4	1st cluster of root dir
100	4	Volume serial #
104	2	FS revision # V.M
106	1	Volume flags
107	1	Active FAT
108	1	Bytes per sector (power of 2 – 2^9 = -512)
109	1	Sectors per cluster (power of 2)
110	1	Number of FATs (1/2: 2 only if TexFAT config/in use)
111	1	Used by INT 13
112	1	Percentage in use
113	7	Reserved
120	390	Boot program
510	2	VBR Signature **0xAA55**

Important concepts w/exFAT:

- FS uses 32-bit arrays w/in FAT to describe cluster numbers: Limit of 2^32-11 cluster addresses [4,294,967,285]

        - Uses free space bitmaps to reduce fragmentation/allocation/detection issues
        - Each cluster tracked in bitmap: Single bit used to denote status of each cluster (1: allocated 0:unallocated)
        - **When file created:** May differ from FAT sys:
            - **If file fragmented:** exFAT & FAT function in same manner
            - **If not fragmented:** FAT not updated
        - **Uses FAT to doc data file fragmentation:** Flag in 1 of dir entry records to indicate whether FAT used to doc
        - Will contain data indicating 1 of 3 possible conditions
            - Pointer to next cluster/fragment
            - EOF: **FF FF FF FF**
            - No fragmentation being tracked: **00 00 00 00**
        - **W/in dir entries:** Multiple 32-byte records w/at least 3 for each dir entry: Each will have identifier byte

- **Directory Entry Record: Record ID: 0x85: Attributes/timestamp/last-accessed/written timestamp**

- **Stream Ext Record: Record ID: 0xC0: Logical size/starting extent/size of filename/CRC of filename**

        - Flag that determine whether FAT being used to track clusters allocated to file

- **Filename Ext Record: Record ID: 0xC1: Filename in Unicode/Addl records added to accommodate longer filenames**

    - **When file del:** FAT isn't zeroed out: Higher likelihood of successful recovery as long as clusters not reused
        - 1st bit of record identifier changed from 1 to 0: Similar to marking 1st byte w/0xE5 on FAT

From <https://www.piratemoo.net/moosings/winfor/fat-basics-advanced-win-forensics-ch-2-notes/>