FSF ASSIGNMENT 2

Ethan Roche Woodward (20094256)

Partition Table:

To get my laptop's partition table I did the command "dd --localwrt if=//./physicaldrive1 of=partition.dd bs=512 skip=0 count=1" which took the first sector of my physical drive with Windows. This is the partition table. The hex for the partition table starts at byte 446 and my MBR partition was follows:

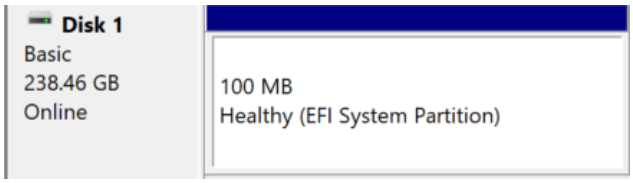
```
00 00 02 00 EE FE 7F 99 01 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 55 AA
```

I used this template to decode my MBR:

Bootable Flag	Starting CHS	Partition Type	Ending CHS	Starting LBA	Length of LBA
0	1,2,3	4	5,6,7	8,9,10,11	12,13,14,15

Byte 4 here shows that the disk is a GPT partitioned disk. This means the disk has a protective GPT MBR with a GPT or EFI header after. This matches with what using Windows' disk management utility shows me along with mmls from the sleuth kit.

```
      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000000000   0000000001   Unallocated
002:  000:000   0000000001   4294967295   4294967295   GPT Safety Partition (0xee)
```



The first byte(Byte 0) indicates that the drive isn't bootable but this is only so older/legacy BIOSes cannot boot from it as they will consider the drive as damaged.

The next three bytes point to where the GPT header is located. They point to CHS(0,0,2) . This is decoded by converting the hex values into binary then moving the first two bits from the Sector value to the start of the Cylinder Value and then simply converting the resulting binary into decimal.

```
Starting CHS
~~~~~~~~~~~~
00       02       00
00000000 00000010 00000000

Head     Sector Cylinder
00000000 000010 0000000000
0        2      0
(0,0,2)
```

Bytes 5,6 and 7 point to the ending CHS(Cylinder Head Sector). These are decoded in the same way as the Starting CHS. The decoded values are as follows:

```
Ending CHS
~~~~~~~~~~
FE       7F       99
11111110 01111111 10011001

Head     Sector Cylinder
11111110 111111 0110011001
254      63     409
(409,254,63)
```

The following four bytes are the Starting LBA, or the Logical Block Address. They are in little endian so had to be reordered, leaving me with the Starting Logical Block Address being 1. Therefore CHS(0,0,2) = LBA 1. The next four bytes were the length of the LBA which was 4,294,967,295.

```
Starting LBA      Length of LBA
~~~~~~~~~~~~      ~~~~~~~~~~~~~
01 00 00 00       FF FF FF FF
Little Endian     4,294,967,295
00 00 00 01
1
```

As my MBR had a GPT header, I needed to decode entries from the next 10 sectors using the following templates:

GPT Header:

Byte Range	Description
0-7	Signature value EFIPART
8-11	Version
12-15	Size of GPT header in bytes
16-19	CRC32 of Partition Table
20-23	Reserved
24-31	LBA of the current GPT partition structure
32-39	LBA of the other GPT partition structure
40-47	LBA of start of partition area
48-55	LBA of end of partition area
56-71	Disk GUID
72-79	LBA of the start of the partition table
80-83	Number of entires in partition table
84-87	Size of each entry in partition table
88-91	CRC32 of Partition Table
92-End	Reserved

Series of entries describing each partition:

Byte Range	Description
0-15	Partition Type
16-31	Unique Partition ID
16-31	Starting LBA
32-39	Ending LBA
48-55	Partition Attributes
56 - 127	Partition Name Unicode

GPT Header:

Bytes 0 to 7 are the Signature value. Decoding the hex values gives “EFI PART” which identifies the GPT as an EFI compatible header.

```
Signature Value EFIPART (0-7)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
45 46 49 20 50 41 52 54
EFI PART
```

The next four bytes are referring to the Version/Revision of the GPT Header. This decodes to be version 1 which is the current one.

```
Version (8-11)
~~~~~~~~~~~~~~~
00 00 01 00
Version 1.0
Current Version
```

Following this is four bytes corresponding to the size of the GPT header.

```
Size of GPT header in bytes (12-15)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
5C 00 00 00
Little Endian
00 00 00 5C
92 Bytes
```

Bytes 16-19 are the CRC 32 checksum in little endian. It can also be confirmed by zeroing the bytes and then running a checksum analysis on a hex editor.

```
CRC32 (16-19)
~~~~~~~~~~~~~
F7 9F 40 CF
Little Endian
CF 40 9F F7
```

Algorithm	Checksum
CRC-32	CF409FF7

Bytes 20-23 are reserved and will always be 00 00 00 00.

24-31 are the LBA (Logical Block Address) of the current header. This is in little endian and is normally 01, as mine is.

```
LBA of current header(24-31)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
01 00 00 00 00 00 00 00
Little Endian
01
```

32-39 are the LBA of the other header structure which is a backup of the main header. These are stored in little endian and when converted to decimal gives Sector 500,118,191.

```
LBA of other header(32-39)
~~~~~~~~~~~~~~~~~~~~~~~~~~
AF 32 CF 1D 00 00 00 00
Little Endian
00 00 00 00 1D CF 32 AF
Converted to Decimal: 500,118,191
```

The next 8 bytes are for the LBA of the start of the partition area. For my GPT I got Sector 34.

```
LBA of start of partition area(40-47)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
22 00 00 00 00 00 00 00
Little Endian
22
Conv to decimal: 34
```

Following this are the 8 bytes for the LBA of end of the partition area. For me this was Sector 500,118,158.

```
LBA of end of partition area(48-55)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
8E 32 CF 1D 00 00 00 00

00 00 00 00 1D CF 32 8E
Converted to Decimal: 500,118,158
```

52-71 is the Disk GUID (Globally Unique Identifier) of this disk drive.

```
Disk GUID(56-71)
~~~~~~~~~~~~~~~~
E1 75 24 EA 55 04 CB 46
AF 61 C5 33 0F 99 9F 54
Or
0x549F990F33C561AF46CB0455EA2475E1
```

Then there’s 72-79 which have the LBA of the start of the partition table. This ended up as 2.

```
LBA of start of the partition table(72-79)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
02 00 00 00 00 00 00 00
Little endian
02
```

80-83 gives the number of areas in the partition table, this being 128 for me.

```
Number of areas in the partition table (80-83)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
80 00 00 00

80
Conv to decimal: 128
```

The next four bytes give the size of each entry in the partition table. This is usually 128 and that is what I also have.

```
Size of each entry in the partition table(84-87)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
80 00 00 00

00 00 00 80
128 in decimal
```

Then 88-91 is the CRC 32 checksum of the Partition table.

```
CRC32 Partition table checksum(88-91)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1F A3 B6 1D
or
0x1DB6A31F
```

From here 92 until the end of the header is all reserved and padded with all zero bytes.

Partitions:

Slot 000:

```
28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B
2E FB 5A B1 96 21 DC 41 A9 E0 2B AC 29 D9 56 48
00 08 00 00 00 00 00 00 FF 27 03 00 00 00 00 00
00 00 00 00 00 00 00 80 45 00 46 00 49 00 20 00
73 00 79 00 73 00 74 00 65 00 6D 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Partition Type (0-15)
~~~~~~~~~~~~~~~~~~~~~~
28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B

Unique Partition ID(16-31)
~~~~~~~~~~~~~~~~~~~~~~~~~~
2E FB 5A B1 96 21 DC 41 A9 E0 2B AC 29 D9 56 48

Starting LBA(32-39)
~~~~~~~~~~~~~~~~~~~
00 08 00 00 00 00 00 00
Little Endian
08 00
2048

Ending LBA(40-47)
~~~~~~~~~~~~~~~~~
FF 27 03 00 00 00 00 00
le
00 00 00 00 00 03 27 FF
206,847

Partition Attributes(48-55)
~~~~~~~~~~~~~~~~~~~~~~~~~~
00 00 00 00 00 00 00 80
0x8000000000000000
Won't recieve a drive letter by default when seen by a computer for the first time or moved
to another computer

Partition Name Unicode(56-127)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
45 00 46 00 49 00 20 00
73 00 79 00 73 00 74 00 65 00 6D 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00

EFI system partition
```

Slot 001:

```
16 E3 C9 E3 5C 0B B8 4D 81 7D F9 2D F0 02 15 AE
50 89 13 0E 61 70 10 47 9E 3D FD F0 D2 88 13 EB
00 28 03 00 00 00 00 00 FF A7 03 00 00 00 00 00
00 00 00 00 00 00 00 80 4D 00 69 00 63 00 72 00
6F 00 73 00 6F 00 66 00 74 00 20 00 72 00 65 00
73 00 65 00 72 00 76 00 65 00 64 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Partition Type (0-15)
~~~~~~~~~~~~~~~~~~~~~~
16 E3 C9 E3 5C 0B B8 4D 81 7D F9 2D F0 02 15 AE

Unique Partition ID(16-31)
~~~~~~~~~~~~~~~~~~~~~~~~~~
50 89 13 0E 61 70 10 47 9E 3D FD F0 D2 88 13 EB

Starting LBA(32-39)
~~~~~~~~~~~~~~~~~~~
00 28 03 00 00 00 00 00
le
03 28 00
Dec: 206,848

Ending LBA(40-47)
~~~~~~~~~~~~~~~~~
FF A7 03 00 00 00 00 00
03 A7 FF
Dec: 239,615

Partition Attributes(48-55)
~~~~~~~~~~~~~~~~~~~~~~~~~~
00 00 00 00 00 00 00 80
0x8000000000000000
Won't recieve a drive letter by default when seen by a computer for the first time or moved
to another computer


Partition Name Unicode(56-127)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
4D 00 69 00 63 00 72 00
6F 00 73 00 6F 00 66 00 74 00 20 00 72 00 65 00
73 00 65 00 72 00 76 00 65 00 64 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Microsoft reserved partition
```

Slot 002:

```
A2 A0 D0 EB E5 B9 33 44 87 C0 68 B6 B7 26 99 C7
78 A5 94 8B 32 14 27 42 98 C5 D2 A5 2B 0B 51 9C
00 A8 03 00 00 00 00 00 FF EF AE 1D 00 00 00 00
00 00 00 00 00 00 00 00 42 00 61 00 73 00 69 00
63 00 20 00 64 00 61 00 74 00 61 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Partition Type (0-15)
~~~~~~~~~~~~~~~~~~~~~~
A2 A0 D0 EB E5 B9 33 44 87 C0 68 B6 B7 26 99 C7

Unique Partition ID(16-31)
~~~~~~~~~~~~~~~~~~~~~~~~~~
78 A5 94 8B 32 14 27 42 98 C5 D2 A5 2B 0B 51 9C

Starting LBA(32-39)
~~~~~~~~~~~~~~~~~~~
00 A8 03 00 00 00 00 00
03 A8 00
Dec: 239,616

Ending LBA(40-47)
~~~~~~~~~~~~~~~~~
FF EF AE 1D 00 00 00 00
1D AE EF FF
Dec: 498,003,967

Partition Attributes(48-55)
~~~~~~~~~~~~~~~~~~~~~~~~~~
00 00 00 00 00 00 00 00
Reserved

Partition Name Unicode(56-127)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
42 00 61 00 73 00 69 00
63 00 20 00 64 00 61 00 74 00 61 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Basic data partition
```

Slot 003:

```
A4 BB 94 DE D1 06 40 4D A1 6A BF D5 01 79 D6 AC
A5 75 2E 3C B8 A4 B3 4B 8A 77 73 9E C6 88 50 25
00 F0 AE 1D 00 00 00 00 FF EF CE 1D 00 00 00 00
01 00 00 00 00 00 00 80 42 00 61 00 73 00 69 00
63 00 20 00 64 00 61 00 74 00 61 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00
00 00 08 F8 FB 7F 00 00 50 FD ED 5D 9B 02 00 00
70 45 42 5E 9B 02 00 00 5D A8 60 03 2F E3 00 00

Partition Type (0-15)
~~~~~~~~~~~~~~~~~~~~~~
A4 BB 94 DE D1 06 40 4D A1 6A BF D5 01 79 D6 AC

Unique Partition ID(16-31)
~~~~~~~~~~~~~~~~~~~~~~~~~~
A5 75 2E 3C B8 A4 B3 4B 8A 77 73 9E C6 88 50 25


Starting LBA(32-39)
~~~~~~~~~~~~~~~~~~~
00 F0 AE 1D 00 00 00 00
le
1D AE F0 00
Dec:498,003,968

Ending LBA(40-47)
~~~~~~~~~~~~~~~~~
FF EF CE 1D 00 00 00 00
le
1D CE EF FF
Dec:500,101,119

Partition Attributes(48-55)
~~~~~~~~~~~~~~~~~~~~~~~~~~
01 00 00 00 00 00 00 80
0x8000000000000001
This is a combination of 0x8000000000000000 (prevents assignment of a drive letter
automatically from new machines) and 0x0000000000000001 (required partition by system)


Partition Name Unicode(56-127)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
42 00 61 00 73 00 69 00
63 00 20 00 64 00 61 00 74 00 61 00 20 00 70 00
61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00

Basic data partition
```

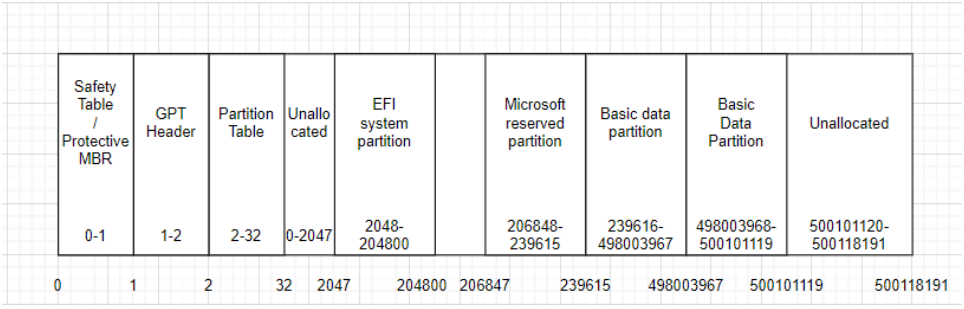
For all of these, the start LBA and ending LBA along with the names are the exact same as what doing “mmls \\.\physicaldrive1” gives:

```
GUID Partition Table (EFI)
Offset Sector: 0
Units are in 512-byte sectors

     Slot      Start        End          Length       Description
000:  Meta     0000000000   0000000000   0000000001   Safety Table
001:  -------  0000000000   0000002047   0000002048   Unallocated
002:  Meta     0000000001   0000000001   0000000001   GPT Header
003:  Meta     0000000002   0000000033   0000000032   Partition Table
004:  000      0000002048   0000206847   0000204800   EFI system partition
005:  001      0000206848   0000239615   0000032768   Microsoft reserved partition
006:  002      0000239616   0498803967   0497764352   Basic data partition
007:  003      0498803968   0500101119   0002097152   Basic data partition
008:  -------  0500101120   0500118191   0000017072   Unallocated
```

Using Microsoft’s “PARTITION_INFORMATION_GPT structure” page (PARTITION_INFORMATION_GPT - Win32 apps, 2021) and the Partition type of each partition, I was able to figure out the partition types. Slot 000 is an EFI system partition, Slot 1 is a Microsoft Reserved Partition, Slot 2 is a partition type made by windows and can be assigned a drive letter by windows and Slot 3 is a Microsoft recovery partition.

Using that same page, I was also able to know what each attribute meant which is shown in the image of each decoded partition.



Mapping of MBR and GPT.

MFT:

This is the Master File Table and has information about all the files and directories, including itself. Before I began decoding the MFT, I had to make an image of the supplied file by using the dd command. From here I used fsstat on the image file to find the MFT. I then used "blkcat -h image.dd 4" to look at the hex of the MFT. To take out and decode the MFT entry itself, I took the first 42 bytes and followed this template along with using information from "BASIC CONCEPTS | NTFS DATA STRUCTURES" (Basic Concepts | NTFS Data Structures, 2022):

Table 13.1. Data structure for a basic MFT entry.

Byte Range	Description	Essential
0–3	Signature ("FILE")	No
4–5	Offset to fixup array	Yes
6–7	Number of entries in fixup array	Yes
8–15	$LogFile Sequence Number (LSN)	No
16–17	Sequence value	No
18–19	Link count	No
20–21	Offset to first attribute	Yes
22–23	Flags (in-use and directory)	Yes
24–27	Used size of MFT entry	Yes
28–31	Allocated size of MFT entry	Yes
32–39	File reference to base record	No
40–41	Next attribute id	No
42–1023	Attributes and fixup values	Yes

Looking at the first four bytes, the values are the values for the word "FILE" which is the signature of an MFT entry.

Bytes 4-5 contain the offset to the fixup array in little endian. This means that the fixup array is 48 bytes into the MFT entry.

```
Offset to Fixup Array 4-5: 3 0 0 0: (Little endian) == 00 30 == Decimal 48
```

6-7 show the number of entries in the fixup array which in this case is 3.

```
Entries in Fixup Array 6-7: 0 3 0 0: (Little endian) == 00 03: 3
```

After this, 8-15 has the $Logfile Sequence Number which records when updates are made to the file system to more quickly fix a corrupt system if it happens. For the supplied file this was 0.

The Sequence Value in 16-17 is raised by one when an entry is allocated. For this file, the value was 1 meaning this entry has only been allocated once.

The Link Count in 18-19 goes up by one for each directory with an entry for this MFT entry. This value is 1 for the MFT entry.

Then Bytes 20-21 show the offset to the first attribute which in this case is byte offset 56. (Note, "==" used when changing from little endian and ":" after used when converting to decimal).

```
Offset to first attribute 20-21: 3 8 0 0 == 00 38 : 56
```

22-23 is the flags and for this entry the flags are 1 which reveals that the entry is in use by the system.

From this is the Used size of the MFT entry, bytes 24-27, which is 408 bytes and then the allocated size of the MFT entry in 28-31 which is 1024 bytes.

Bytes 32-39 have the file reference to base record as 0 which indicates that this is a base entry.

Bytes 40 and 41 show that the next attribute ID that will be assigned is 4, telling me that 1-3 will already be assigned to attributes.

After this I went to byte 48 as it is the start of the fixup array. The first two bytes (04 00/0x0004) are the signature value. The two following bytes need to be used to replace the signature value at the last two bytes of each sector. Both of them have 0x0004 which gets replaced with 0x0000. The first attribute is in byte 56, the value of which is 10(hex) in the case for this file and the attributes end at 504 marked by 0xFFFF FFFF.

```
FF FF FF FF
```

MFT Entries & Associated Attributes:

To decode these entries, I used fls and istat from TSK along with NTFS FILE TYPES - NTFS.COM (NTFS File Types - NTFS.com, n.d.) to understand the attributes what they mean.

Entry 7: $Boot

This file is 8192 bytes large and it's parent MFT entry is entry 5. It has one link. The values of it's $STANDARD_INFORMATION Attribute show it's flags, owner ID, Security ID, and created, modified, MFT modified and Accessed dates and time.

This entry is flagged as a hidden entry and a system entry. Both the owner and security IDs are 0.

The times are as follows:

```
Created:        2012-01-24 21:49:52.000000000 (GMT Standard Time)
File Modified:  2012-01-24 21:49:52.000000000 (GMT Standard Time)
MFT Modified:   2012-01-24 21:49:52.000000000 (GMT Standard Time)
Accessed:       2012-01-24 21:49:52.000000000 (GMT Standard Time)
```

The $FILE_NAME attribute simply gives the name of the entry as $BOOT, it's parent entry, sequence and both allocated and actual sizes. For this entry the Parent MFT entry is 5 and so is it's sequence number. The entry is allocated and uses 8192 bytes.

The entry also has a $SECURITY_DESCRIPTOR attribute and a $DATA attribute. The Security Descriptor attribute is 100 bytes, and the Data attribute is 8192 bytes with a starting address of 0 and length of 2 clusters (0 and 1). $STANDARD_INFORMATION, $FILE_NAME and $SECURITY_DESCRIPTOR are all Resident while $DATA is Non-Resident. This means that the Data is too large to fit into the MFT record, so it is allocated clusters of disk space somewhere else on the volume.

Entry 3: $LogFile

This file is 2097152 bytes with 1 link. It is entry 2 in the MFT and it's sequence number is 2. The values of the $STANDARD_INFORMATION Attribute Values are as follows: The flags of the entry are Hidden and System. The Owner ID is 0 and Security ID is 256. The times of creation, modification, MFT modification and access are as follows:

```
Created:        2012-01-24 21:49:52.000000000 (GMT Standard Time)
File Modified:  2012-01-24 21:49:52.000000000 (GMT Standard Time)
MFT Modified:   2012-01-24 21:49:52.000000000 (GMT Standard Time)
Accessed:       2012-01-24 21:49:52.000000000 (GMT Standard Time)
```

The $FILE_NAME Attribute Values were as follows. The name of the file is $LogFile and it's Parent MFT entry is 5, along with it's Sequence number. The size the entry is allocated and takes up is here too(2097152 bytes).

The Entry also has a $DATA attribute of size 2097152 which starts at Cluster 977 and is 512 clusters long. This attribute is also Non-Resident meaning the Data is stored elsewhere on the disk.

Data Categories:

Information here is mainly from "File System Forensic Analysis" (Carrier, 2011)

File System:

This has general file system information. While all file systems may have a typical form, every instance is unique due to having a unique size. Data here can say where some data structures can be found or the size of a data unit.

Content:

The data that makes up the actual content of a file is here. Most data belongs here and is usually organised into standard-sized containers. These have different names depending on the file system.

Metadata:

Metadata stores the information that describes files and data. This information contains the location of file content, the size of the file and creation, modified and accessed times of the file. An example of this is the NTFS MFT entries.

File Name:

This has the data of the name of each file. Usually these are in the contents of a directory with their corresponding metadata address.

Application:

This category has data that provides special features that aren't needed in reading or writing a file. This data is in the file system specification as it may improve efficiency when using it in the file system.

File Categories Comparison:

To compare the file categories of NTFS and FAT I used my lecture notes along with information from A Forensic Comparison of NTFS and FAT32 File Systems (Laine Rusbarsky, 2012).

In NTFS the file system is a Journaling File System. This tracks changes in the system via a log. This allows for quicker recovery if there is an error or crash. Any changes made in the volume via metadata is stored in the $LogFile. The main part of this file system is the Master File Table(MFT), This has records of itself and all other files. The first 16 entries are usually reserved for MFT data and metadata files and the rest is used for file records. There is also a clone of the first four MFT records called the $MFTMirr. On the supplied image, $MFT is in Cluster 4 and $MFTMirr is in cluster 976. Entries data is stored in the first 42 bytes of an entry and the remainder is for attributes. Resident attributes are stored in the MFT and Non-resident are stored elsewhere on the partition. NTFS attributes are sorted into a B-Tree structure, allowing grouping into large folders. FAT on the other hand needs to scan all the files in a folder to create a file listing. FAT's two main data structures are the File Allocation Table and it's directory entries. Here each file is stored and given a directory entry and if multiple clusters are used then you can use the FAT (file allocation table) to get the clusters of the files.

Both NTFS and FAT store content via clusters. NTFS can make use of smaller cluster size to increase disk efficiency. NTFS makes use of $Bitmap and starts at sector 0 to find the first available cluster to assign a file to. On the image provided $Bitmap is in Cluster 252 and takes up 248 bytes. There is also an MFT entry that contains information on bad clusters to be avoided when searching for clusters to assign a file to. The main data of this on the supplied file is Non-resident and takes up 7999488 bytes. Meanwhile FAT has a number of reserved sectors and FAT areas which includes the Boot Sector, FAT and the copy of the FAT. The data area of FAT starts after the FAT structures. The File Allocation Table has entries for every cluster on the system.

The metadata of the NTFS system is stored in the MFT via three attributes, $STANDARD_INFORMATION, $FILE_NAME and $DATA. $STANDARD_INFORMATION has the primary metadata for a file. $FILE_NAME has the reference for the parent directory and $DATA can store any data. FAT's metadata is stored in the root directory and will store data with both a regular file name (short file name) and a long file name. A folder entry has 13 characters of the long file name.

References:

Carrier, B., 2011. File system forensic analysis. Upper Saddle River, NJ: Addison-Wesley.

Docs.microsoft.com. 2021. PARTITION_INFORMATION_GPT - Win32 apps. [online] Available at: <https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-partition_information_gpt?redirectedfrom=MSDN> [Accessed 21 December 2021].

Flylib.com. 2022. Basic Concepts | NTFS Data Structures. [online] Available at: <https://flylib.com/books/en/2.48.1/basic_concepts.html> [Accessed 22 December 2021].

Laine Rusbarsky, K., 2012. *A Forensic Comparison of NTFS and FAT32 File Systems*. [ebook] Kansas City: MARSHALL UNIVERSITY FORENSIC SCIENCE CENTER & FBI, HEART OF AMERICA REGIONAL COMPUTER FORENSICS LABORATORY. Available at: <https://www.marshall.edu/forensics/files/RusbarskyKelsey_Research-Paper-Summer-2012.pdf> [Accessed 30 December 2021].

Ntfs.com. n.d. NTFS File Types - NTFS.com. [online] Available at: <http://ntfs.com/ntfs-files-types.htm> [Accessed 28 December 2021].