Chapter 10. FAT Data Structures

In the previous chapter, we examined the basic concepts of a FAT file system and how to analyze it. Now we are going to get more detailed and examine the data structures that make up FAT. This chapter will ignore the five-category model and instead focus on each individual data structure. This makes it easier to understand FAT because many of the data structures are in more than one category. I assume that you have already read Chapter 9, "FAT Concepts and Analysis," or that you are reading it in parallel. All the hexdumps and data shown in this chapter correspond to the data that were analyzed in Chapter 9 using tools from The Sleuth Kit (TSK).

Boot Sector

The boot sector is located in the first sector of FAT file system and contains the bulk of the file system category of data. FAT12/16 and FAT32 have different versions of the boot sector, but they both have the same initial 36 bytes. The data structure for the first 36 bytes is given in Table 10.1, and the data structures for the remaining bytes are given in Tables 10.2 and 10.3.

Table 10.1. Data structure for the first 36 bytes of the FAT boot sector.

Byte Range	Description	Essential
0–2	Assembly instruction to jump to boot code.	No (unless it is a bootable file system)
3–10	OEM Name in ASCII.	No
11–12	Bytes per sector. Allowed values include 512, 1024, 2048, and 4096.	Yes
13–13	Sectors per cluster (data unit). Allowed values are powers of 2, but the cluster size must be 32KB or smaller.	Yes
14–15	Size in sectors of the reserved area.	Yes
16–16	Number of FATs. Typically two for redundancy, but according to Microsoft it can be one for some small storage devices.	Yes
17–18	Maximum number of files in the root directory for FAT12 and FAT16. This is 0 for FAT32 and typically 512 for FAT16.	Yes
19–20	16-bit value of number of sectors in file system. If the	Yes

Table 10.1. Data structure for the first 36 bytes of the FAT boot sector.

Byte Range	Description	Essential
	number of sectors is larger than can be represented in this 2-byte value, a 4-byte value exists later in the data structure and this should be 0.	
21–21	Media type. According to the Microsoft documentation, 0xf8 should be used for fixed disks and 0xf0 for removable.	No
22–23	16-bit size in sectors of each FAT for FAT12 and FAT16. For FAT32, this field is 0.	Yes
24–25	Sectors per track of storage device.	No
26–27	Number of heads in storage device.	No
28–31	Number of sectors before the start of partition.[1]	No
32–35	32-bit value of number of sectors in file system. Either this value or the 16-bit value above must be 0.	Yes

[1] My testing has shown that for file systems in an extended partition, Windows sets this value based on the beginning of the extended partition, not the beginning of the disk.

Table 10.2. Data structure for the remainder of the FAT12/16 boot sector.

Byte Range	Description	Essential
0–35	See Table 10.1.	Yes
36–36	BIOS INT13h drive number.	No
37–37	Not used.	No
38–38	Extended boot signature to identify if the next three values are valid. The signature is 0x29.	No
39–42	Volume serial number, which some versions of Windows will calculate based on the creation date and time.	No
43–53	Volume label in ASCII. The user chooses this value when creating the file system.	No
54–61	File system type label in ASCII. Standard values include "FAT," "FAT12," and "FAT16," but nothing is required.	No
62–509	Not used.	No

Table 10.2. Data structure for the remainder of the FAT12/16 boot sector.

Byte Range	Description	Essential
510–511	Signature value (0xAA55).	No

Table 10.3. Data structure for the remainder of the FAT32 boot sector.

Byte Range	Description	Essential
0–35	See Table 10.1.	Yes
36–39	32-bit size in sectors of one FAT.	Yes
40–41	Defines how multiple FAT structures are written to. If bit 7 is 1, only one of the FAT structures is active and its index is described in bits 0–3. Otherwise, all FAT structures are mirrors of each other.	Yes
42–43	The major and minor version number.	Yes
44–47	Cluster where root directory can be found.	Yes
48–49	Sector where FSINFO structure can be found.	No
50–51	Sector where backup copy of boot sector is located (default is 6).	No
52–63	Reserved.	No
64–64	BIOS INT13h drive number.	No
65–65	Not used.	No
66–66	Extended boot signature to identify if the next three values are valid. The signature is 0x29.	No
67–70	Volume serial number, which some versions of Windows will calculate based on the creation date and time.	No
71–81	Volume label in ASCII. The user chooses this value when creating the file system.	No
82–89	File system type label in ASCII. Standard values include "FAT32," but nothing is required.	No
90–509	Not used.	No
510–511	Signature value (0xAA55).	No

The first value in the boot sector, bytes 0 to 2, is a boot code instruction tells the computer where to find the code needed to boot the operating system. If the file system is

not used to boot the computer, the value is not needed. You could use this value to identify what boot code is used. Note that DOS and Windows require that the value be set on non-bootable file systems, but other OSes, such a Linux, do not.

The media type value is used to identify if the file system is on fixed or removable media, but Microsoft Windows does not use it. A second copy of the media type exists in the file allocation table, and it is the one that Windows uses [Microsoft 2001]. The concepts of the other fields were discussed in Chapter 9.

From bytes 36 onward, FAT12 and FAT16 have a different layout than FAT32. The one value that they both have in common is the signature 0x55 in byte 510 and 0xAA in byte 511. Note that this is the same signature at the same location that the DOS partition table uses in its first sector (you'll also see it again in the first NTFS sector). The data structure values for the rest of the FAT12 and FAT16 boot sector are given in Table 10.2.

The data structure for the rest of the FAT32 boot sector is given in Table 10.3.

The difference between the FAT12/16 and FAT32 boot sector is that the FAT32 sector includes data to make the file system more scalable and flexible. There can be different policies for how the FAT structures are written to and a backup copy of the boot sector exists. There is also a version field, but there seems to be only one version used by Microsoft at the time of this writing.

The data between bytes 62 to 509 in a FAT12/16 file system, and bytes 90 to 509 in a FAT32 file system do not have a specified purpose, but are typically used to store boot code and error messages. Here is a hex dump of the first sector of a FAT32 file system from a Windows XP system:

```
# dcat –f fat fat-4.dd 0 | xxd
0000000: eb58 904d 5344 4f53 352e 3000 0202 2600  .X.MSDOS5.0...&.
0000016: 0200 0000 00f8 0000 3f00 4000 c089 0100  ........?.@.....
0000032: 4023 0300 1d03 0000 0000 0000 0200 0000  @#..............
0000048: 0100 0600 0000 0000 0000 0000 0000 0000  ................
0000064: 8000 2903 4619 4c4e 4f20 4e41 4d45 2020  ..).F.LNO NAME
0000080: 2020 4641 5433 3220 2020 33c9 8ed1 bcf4    FAT32 3.....
0000096: 7b8e c18e d9bd 007c 884e 028a 5640 b408  {......|.N..V@..
0000112: cd13 7305 b9ff ff8a f166 0fb6 c640 660f  ..s......f...@f.
0000128: b6d1 80e2 3ff7 e286 cdc0 ed06 4166 0fb7  ....?.......Af..
[REMOVED]
0000416: 0000 0000 0000 0000 0000 0000 0d0a 5265  ..............Re
0000432: 6d6f 7665 2064 6973 6b73 206f 7220 6f74  move disks or ot
0000448: 6865 7220 6d65 6469 612e ff0d 0a44 6973  her media....Dis
0000464: 6b20 6572 726f 72ff 0d0a 5072 6573 7320  k error...Press
0000480: 616e 7920 6b65 7920 746f 2072 6573 7461  any key to resta
0000496: 7274 0d0a 0000 0000 00ac cbd8 0000 55aa  rt............U.
```

The first line shows us that the OEM name is "MSDOS5.0," which may have been generated by a Windows 2000 or XP system. The data is written in little endian ordering,

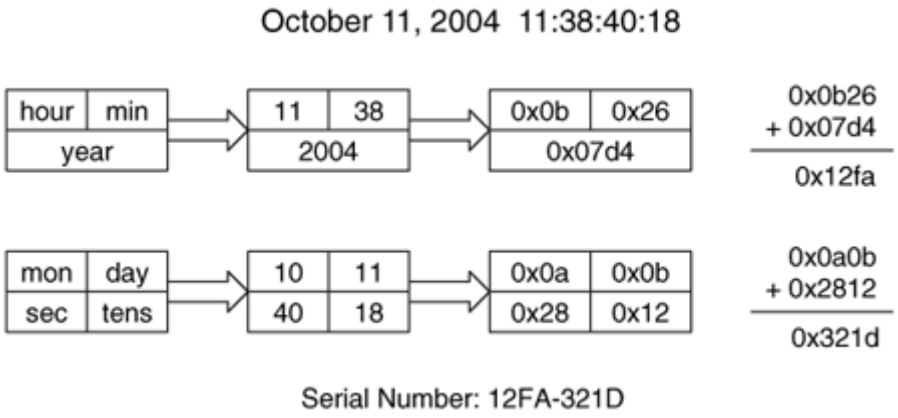
so the data structure fields that are numbers will appear in reverse order and strings will appear in the expected order. Bytes 11 to 12 show us that each sector is 512 bytes, (0x0200) and byte 13 shows us that the size of each cluster in the data area is 2 sectors, which is 1024 bytes. Bytes 14 to 15 show us that there are 38 (0x0026) sectors in the reserved area, so we know that the FAT area will start in sector 38, and byte 16 shows that there are two FAT structures. Bytes 19 to 20 contain the 16-bit file system size value and it is 0, which means that the 32-bit field in bytes 32 to 35 must be used. This field shows us that the size of the file system is 205,632 (0x00032340) sectors. Bytes 28 to 31 show that there are 100,800 (0x0001 89c0) sectors before the start of this file system, which may have been allocated to a small partition. For example, this could be a dual boot system, or there could be a hibernation partition for a laptop. The partition table should be analyzed for more information.

This image is FAT32, so we need to apply the appropriate data structure from now on. Bytes 36 to 39 show that the size of each FAT structure is 797 (0x0000 031d) sectors, and because we know there will be two FAT structures, the total size of the FAT area will be 1,594 sectors. Bytes 48 to 49 show that the FSINFO information is located in sector 1, and bytes 50 to 51 show that the backup copy of the boot sector is in sector 6.

The volume serial number is located in bytes 67 to 70, and its value is 0x4c194603. The volume label is in bytes 71 to 81 and has the value "NO NAME" (plus four spaces). We will see later that the real label is stored in another location in the file system. The type label is in bytes 82 to 89, and it is "FAT32" (plus three spaces) for this system. Bytes 90 to 509 are not used by the file system, but we can see data here that is used if the system tries to boot from this file system. Bytes 510 and 511 have the signature 0xAA55 value. The output from running the `fsstat` tool from TSK on this image was given in Chapter 9.

As mentioned in Chapter 9, some versions of Windows will assign the volume serial number using the file system creation date and time values. I found that Windows 98 does this, but that Windows XP does not. The calculation is broken up into the upper 16 bits and the lower 16 bits [Wilson 2003]. With the exception of the year, each field in the date is converted to a 1-byte hexadecimal value and placed in its location in the equation. The year gets a 2-byte hexadecimal value. Figure 10.1 shows the process. The upper 16 bits are the result of adding the hours, minutes, and year. The lower 16 bits are the result of adding the month, day, seconds, and tens of seconds. The sample file system image we looked at is from a Windows XP system and it does not use this calculation.

Figure 10.1. Process for calculating the volume serial number from the creation date and time.



FAT32 FSINFO

A FAT32 file system has a FSINFO data structure that includes hints about where the operating system can allocate new clusters. Its location is given in the boot sector, and its layout is given in Table 10.4.

Table 10.4. Data structure for the FAT32 FSINFO sector.

Byte Range	Description	Essential
0–3	Signature (0x41615252)	No
4–483	Not used	No
484–487	Signature (0x61417272)	No
488–491	Number of free clusters	No
492–495	Next free cluster	No
496–507	Not used	No
508–511	Signature (0xAA550000)	No

None of these values is required. They are there as a suggestion for the OS, but it is valid for them to not be updated. Here we see the contents of sector 1 from the file system we previously examined. The byte offsets are given relative to the start of this sector.

```
# dcat –f fat fat-4.dd 1 | xxd
0000000: 5252 6141 0000 0000 0000 0000 0000 0000 RRaA............
0000016: 0000 0000 0000 0000 0000 0000 0000 0000 ................
[REMOVED]
```

```
0000464: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000480: 0000 0000 7272 4161 1e8e 0100 4b00 0000 ....rrAa....K...
0000496: 0000 0000 0000 0000 0000 0000 0000 55aa ..............U.
```

We see the signatures at bytes 0 to 3, 484 to 487, and 508 to 511. The number of free clusters is in bytes 488 to 491, and this file system has 101,918 (0x0001 8e1e) free clusters. Note that this value is in clusters and not sectors. The next free cluster is located in bytes 492 to 495, and we see that it is cluster 75 (0x0000 004b).

FAT

The FAT is crucial to a FAT file system and has two purposes. It is used to determine the allocation status of a cluster and to find the next allocated cluster in a file or directory. This section covers how we find the FAT and what it looks like.

There are typically two FATs in a FAT file system, but the exact number is given in the boot sector. The first FAT starts after the reserved sectors, the size of which is given in the boot sector. The total size of each FAT is also given in the boot sector, and the second FAT, if it exists, starts in the sector following the end of the first.

The table consists of equal-sized entries and has no header or footer values. The size of each entry depends on the file system version. FAT12 has 12-bit entries, FAT16 has 16-bit entries, and FAT32 has 32-bit entries. The entries are addressed starting with 0, and each entry corresponds to the cluster with the same address.

If a cluster is not allocated, its entry will have a 0 in it. If a cluster is allocated, its entry will be non-zero and will contain the address of the next cluster in the file or directory. If it is the last cluster in a file or directory, its entry will have an end-of-file marker, which is any value greater than 0xff8 for FAT12, 0xfff8 for FAT16 and 0x0fff fff8 for FAT32. If an entry has a value of 0xff7 for FAT12, 0xfff7 for FAT16, or 0x0fff fff7 for FAT32, the cluster has been marked as damaged and should not be allocated.

Recall that the first addressable cluster in the file system is #2. Therefore, entries 0 and 1 in the FAT structure are not needed. Entry 0 typically stores a copy of the media type, and entry 1 typically stores the dirty status of the file system. There is also a storage value for the media type in the boot sector, but as previously noted, Windows might not use it and could use the value in FAT entry 0 instead. The dirty status can be used to identify a file system that was not unmounted properly (improper shutdown) or that hardware surface errors were encountered. Both of these values are non-essential and may not be accurate.

To examine FAT structure of our sample image we view sector 38, which is the first sector following the reserved area:

```
# dcat –f fat fat-4.dd 38 | xxd
[REMOVED]
0000288: 4900 0000 4a00 0000 4c00 0000 0000 0000 I...J...L.......
0000304: 4d00 0000 ffff ff0f 4f00 0000 ffff ff0f M.......O.......
0000320: 5100 0000 5200 0000 ffff ff0f ffff ff0f Q...R...........
0000336: ffff ff0f 0000 0000 0000 0000 0000 0000 ................
0000352: 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

This output is from a FAT32 file system, so each entry is 4 bytes (two columns) and the first entry shown has a value of 73 (0x0000 0049). Its actual location in the FAT structure is at byte offset 288, and we can divide it by four (the number of bytes per entry) to determine that it is entry 72. We know that this entry is allocated because it is non-zero and the next cluster in the file is cluster 73.

We can see that the entry at bytes 300 to 303 and the entries at bytes 340 onwards are all 0, which means that the clusters corresponding to those entries are not allocated. In this example, the bytes at 300 to 303 are for cluster 75, and the bytes at 340 are for cluster 85 and onwards. There is an example in the directory entry section that shows how to follow a cluster chain in the FAT.

We can see the allocation status of a cluster using tools from TSK, but we will have to translate sectors and clusters. In this example file system, the sector of cluster 2 is 1,632 and each cluster is 2 sectors. We can translate cluster 75 to a sector by using

```
(Cluster 75 – Cluster 2) * 2 (Sectors per Cluster) + Sector 1,632 =
Sector 1,778
```

The `dstat` tool in TSK will show us the allocation status and cluster address of any sector. When we run it on sector 1,778 we get

```
# dstat -f fat fat-4.dd 1778
Sector: 1778
Not Allocated
Cluster: 75
```

Directory Entries

The FAT directory entry contains the name and metadata for a file or directory. One of these entries is allocated for every file and directory, and they are located in the clusters allocated to the file's parent directory. This data structure supports a name that has only 8 characters in the name and 3 characters in the extension. If the file has a more complex name, there will be a long file name directory entry in addition to a directory entry. The long file name version is discussed in the next section of this chapter. The basic directory entry structure has the fields given in Table 10.5.

Table 10.5. Data structure for a basic FAT directory entry.

Byte Range	Description	Essential
0–0	First character of file name in ASCII and allocation status (0xe5 or 0x00 if unallocated)	Yes
1–10	Characters 2 to 11 of file name in ASCII	Yes
11–11	File Attributes (see Table 10.6)	Yes
12–12	Reserved	No
13–13	Created time (tenths of second)	No
14–15	Created time (hours, minutes, seconds)	No
16–17	Created day	No
18–19	Accessed day	No
20–21	High 2 bytes of first cluster address (0 for FAT12 and FAT16)	Yes
22–23	Written time (hours, minutes, seconds)	No
24–25	Written day	No
26–27	Low 2 bytes of first cluster address	Yes
28–31	Size of file (0 for directories)	Yes

The first byte of the data structure works as the allocation status, and if it is set to 0xe5 or 0x00, the directory entry is unallocated. Otherwise, the byte is used to store the first character of the file name. The name is typically in ASCII, but could also use one of the Microsoft code pages if the name uses non-ASCII symbols [Microsoft 2004]. If the file name has the value 0xe5 in that byte, 0x05 should be used instead. If the name does not have 8 characters in its name, unused bytes are typically filled in with the ASCII value for a space, which is 0x20.

The file size field is 4 bytes and, therefore, the maximum file size is 4GB. Directories will have a size of 0 and the FAT structure must be used to determine the number of clusters allocated to it. The attributes field can have one or more of the bits in Table 10.6 set.

Table 10.6. Flag values for the directory entry attributes field.

Flag Value (in bits)	Description	Essential
0000 0001 (0x01)	Read only	No
0000 0010 (0x02)	Hidden file	No

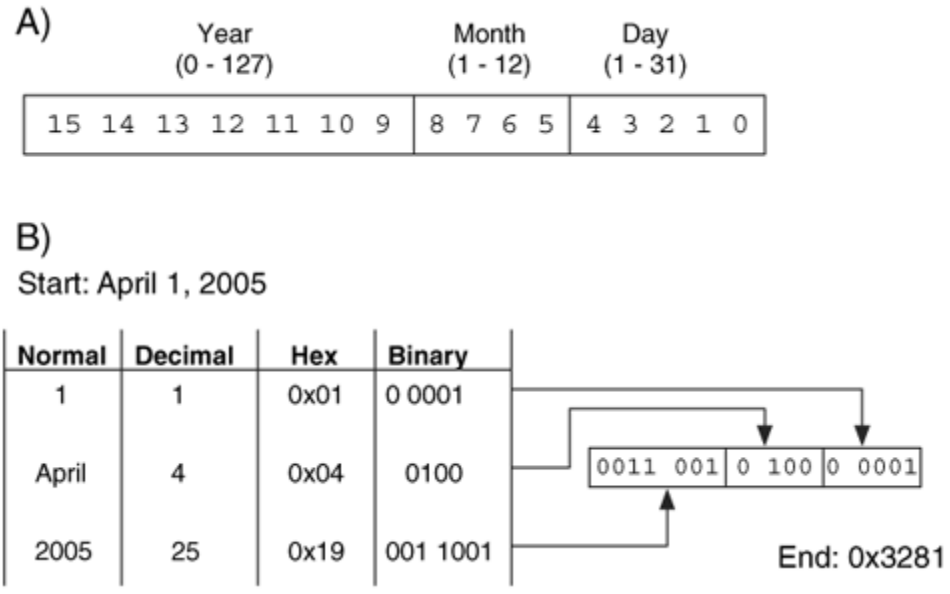
Table 10.6. Flag values for the directory entry attributes field.

Flag Value (in bits)	Description	Essential
0000 0100 (0x04)	System file	No
0000 1000 (0x08)	Volume label	Yes
0000 1111 (0x0f)	Long file name	Yes
0001 0000 (0x10)	Directory	Yes
0010 0000 (0x20)	Archive	No

The upper two bits of the attribute byte are reserved. Directory entries that have the long file name attribute set have a different structure because they are storing the long name for a file, and they will be described in the next section. Notice that the long file name attribute is a bit-wise combination of the first four attributes. Microsoft found that older OSes would ignore directory entries with all the bits set and would not complain about the different layout.

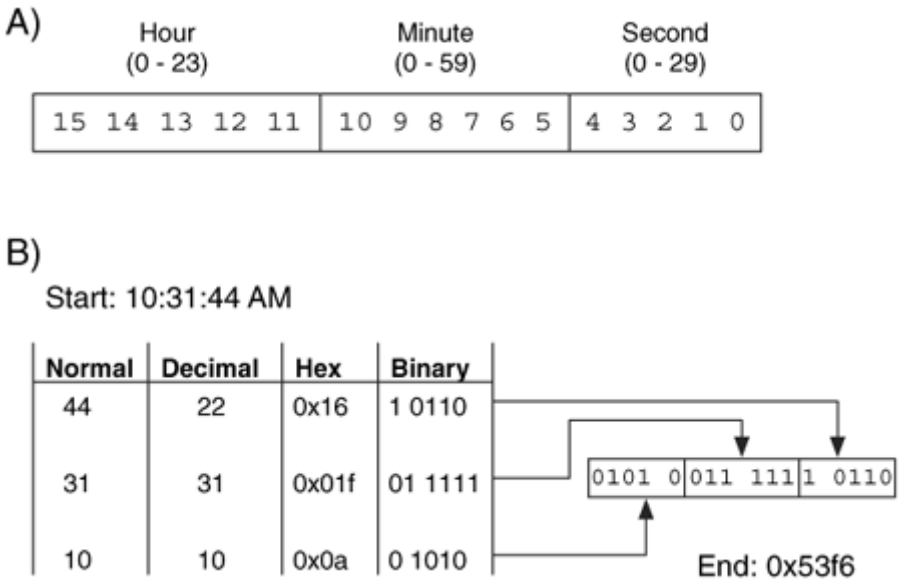
The date portion of each timestamp is a 16-bit value that has three parts, which are shown in Figure 10.2(A). The lower 5 bits are for the day of the month, and the valid values are 1 to 31. Bits 5 to 8 are for the month, and the valid values are 1 to 12. Bits 9 to 15 are for the year, and the value is added to 1980. The valid range is from 0 to 127, which gives a year range of 1980 to 2107. A conversion of the date April 1, 2005 to its hexadecimal format can be found in Figure 10.2(B).

Figure 10.2. Breakdown of the date value and the conversion of April 1, 2005 to its FAT date format.



The time value is also a 16-bit value and also has three parts. The lower 5 bits are for the second, and it uses two-second intervals. The valid range of this value is 0 to 29, which allows a second range of 0 to 58 in two-second intervals. The next 6 bits are for the minute and have a valid range of 0 to 59. The last 5 bits are for the hour and have a valid range of 0 to 23. This can be seen in Figure 10.3(A). An example of converting the time 10:31:44 a.m. to the FAT format can be found in Figure 10.3(B).

Figure 10.3. Breakdown of the time value and the conversion of 10:31:44 a.m. to its FAT time format.



Fortunately, there are many tools that will convert these values for you so that you do not have do always do it by hand.[1] Many hex editors will show the date if you highlight the value and have the correct options set. As we will see in the later chapters, this method of saving the time is much different from other file systems, which save the time as the number of seconds since a given time.

[1] An example is "Decode from Digital Detective" (https://www.digital-detective.co.uk).

Let's look at the raw contents of two directory entries from the root directory. The starting location of the root directory in a FAT32 file system is given in the boot sector.

```
# dcat –f fat fat-4.dd 1632 | xxd
0000000: 4641 5420 4449 534b 2020 2008 0000 0000 FAT DISK .....
0000016: 0000 0000 0000 874d 252b 0000 0000 0000 .......M%+......
0000032: 5245 5355 4d45 2d31 5254 4620 00a3 347e RESUME-1RTF ..4~
```

```
0000048: 4a30 8830 0000 4a33 7830 0900 f121 0000 .0.0.....0...!..
```

The first two lines show a directory entry with the attribute at byte 11 set to the binary value 0000 1000 (0x08), which is for a volume label. We can also see that the write time and date are set at bytes 22 to 25 on line 2. The write time on a volume label may contain the date when the file system was created. Note that the volume label in the boot sector was set to "NO NAME."

The third and fourth lines are for a second directory entry, and we see that the name of this file is "RESUME-1.RTF." The attribute value at byte 43 is 0000 0010 (0x20), which means that only the archive attribute bit is set. Byte 45 shows the tenths of a second for the create time, which is 163 (0xa3). Bytes 46 to 47 have the created time, 0x7e34, which is 15:49:40. The created day is in bytes 48 to 49 and has a value of 0x304a, which is February 10, 2004. The rest of the times are left as an exercise, if you are really bored.

We can see from bytes 52 to 53 and 58 to 59 that the starting cluster is 9 (0x0000 0009), and bytes 60 to 63 show that the file size is 8,689 (0x0000 21f1) bytes. To determine all the clusters in this file, we will need to refer to the FAT. Cluster 9 has a 36-byte offset into the FAT32 structure, and we previously calculated that the primary FAT structure starts in sector 38. Its contents are shown here:

```
# dcat –f fat fat-4.dd 38 | xxd
[REMOVED]
0000032: ffff ff0f 0a00 0000 0b00 0000 0c00 0000 ................
0000048: 0d00 0000 0e00 0000 0f00 0000 1000 0000 ................
0000064: 1100 0000 ffff ff0f 1300 0000 1400 0000 ................
```

The table entry for cluster 9 is located in bytes 36 to 39, and we see that the value is 10 (0x0000 000a), which means that cluster 10 is the next cluster in the chain. The table entry for cluster 10 is in bytes 40 to 43, and we see that the value is 11 (0x0000 000b). We can see that consecutive clusters were allocated to this file until we get to entry 17 at bytes 68 to 71, which has an end-of-file marker (0x0fff ffff). We can verify that we have the correct number of clusters by comparing the file size with the allocated space. The file has allocated 9 1,024-byte clusters, so there are 9,216 bytes of storage space for the 8,689-byte file.

We can now view some of this same data with TSK. Remember that TSK uses sector addresses instead of cluster addresses. To convert cluster 9 to its sector address, we need the sector address of cluster 2, which is 1,632:

```
(Cluster 9 – Cluster 2) * 2 (Sectors per Cluster) + Sector 1,632 =
Sector 1,646
```

The `fsstat` tool in TSK dumps the contents of the FAT structures. We previously saw part of the `fsstat` output when discussing the file system category of data, but the FAT contents were removed. Here is that output:

```
# fsstat –f fat fat-4.dd
[REMOVED]
1642-1645 (4) -> EOF
1646-1663 (18) -> EOF
1664-1681 (18) -> EOF
[REMOVED]
```

Here the output shows us the cluster chain for `RESUME-1.RTF` from sectors 1646 to 1663 and the End of File. Each cluster was 2 sectors in size, so we can see in the parentheses that there are 18 sectors in the cluster chain.

The `istat` tool in TSK shows the details of a directory entry and its output for this entry is given next. Using the metadata-addressing scheme of TSK, the `RESUME-1.RTF` file is the second entry in the root directory, which means that it has an address of 4.

```
# istat -f fat fat-4.dd 4
Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 8689
Name: RESUME-1.RTF

Directory Entry Times:
Written:        Wed Mar 24 06:26:20 2004
Accessed:       Thu Apr  8 00:00:00 2004
Created:        Tue Feb 10 15:49:40 2004

Sectors:
1646 1647 1648 1649 1650 1651 1652 1653
1654 1655 1656 1657 1658 1659 1660 1661
1662 1663
```

Long File Name Directory Entries

The standard directory entry can support names with only 8 characters in the name and 3 characters in the extension. Longer names or names that use special characters require long file name (LFN) directory entries. A file will have a normal entry in addition to any LFN entries, and the LFN entries will precede the normal entry. The LFN version of the directory entry has the fields shown in Table 10.7.

Table 10.7. Data structure for an LFN FAT directory entry.

Byte Range	Description	Essential
0–0	Sequence number (ORed with 0x40) and allocation status (0xe5 if unallocated)	Yes
1–10	File name characters 1–5 (Unicode)	Yes
11–11	File attributes (0x0f)	Yes
12–12	Reserved	No
13–13	Checksum	Yes
14–25	File name characters 6–11 (Unicode)	Yes
26–27	Reserved	No
28–31	File name characters 12–13 (Unicode)	Yes

The sequence number field is a counter for each entry needed to store the file name, and the first entry has a value of 1. The sequence number increases for each LFN entry until the final entry, which is a bitwise OR with the value 0x40. When two values are bitwise ORed together, the result has a 1 wherever any of the two inputs had a 1.

The LFN entries for a file are listed before the short name entry and are in reverse order. Therefore, the first entry that you will find in the directory will be the last LFN entry for the file and will have the largest sequence value. Unused characters are padded with 0xff, and the name should be NULL-terminated if there is room.

The file attributes of a LFN entry must be 0x0F. The checksum is calculated using the short name of the file, and it should be the same for each of the LFN entries for the file. If the checksum in a LFN entry does not match its corresponding short name, an OS that does not support long file names could have been used and made the directory corrupt. The checksum algorithm iterates over each letter in the name, and at each step it rotates the current checksum by one bit to the right and then adds the ASCII value of the next letter. The C code for it is

```
c = 0;
for (i = 0; i < 11; i++) {
    // Rotate c to the right
    c = ((c & 0x01) ? 0x80 : 0) + (c >> 1);
    // Add ASCII character from name
    c = c + shortname[i];
}
```

Let's look at the two LFN and one normal entry for a file in the root directory of our test image:

```
# dcat –f fat fat-4.dd 1632 | xxd
[REMOVED]
0000064: 424e 0061 006d 0065 002e 000f 00df 7200  BN.a.m.e......r.
0000080: 7400 6600 0000 ffff ffff 0000 ffff ffff  t.f.............
0000096: 014d 0079 0020 004c 006f 000f 00df 6e00  .M.y. .L.o....n.
0000112: 6700 2000 4600 6900 6c00 0000 6500 2000  g. .F.i.l...e. .
0000128: 4d59 4c4f 4e47 7e31 5254 4620 00a3 347e  MYLONG~1RTF ..4~
0000144: 4a30 8830 0000 4a33 7830 1a00 8f13 0000  J0.0..J3x0......
```

The first entry is in the first two lines, and we see that byte 75 is set to 0x0f, so we know it is a LFN entry. The sequence number is in byte 64, and it is 0x42. If we remove the OR of 0x40 for the end marker, we get 0x02 and see that there will be two LFN entries. The checksum in byte 77 gives a value of 0xdf, which we can later verify when we find the short name. The first five characters in this entry (although not the first characters in the name) are in bytes 65 to 74 and give 'Name.' The second section of characters in bytes 78 to 89 gives 'rtf.' The rest of the values are set to 0xffff. The final section of characters in bytes 92 to 95 is also set to 0xffff. Therefore, we know that the end of the long name is 'Name.rtf.'

The second entry, starting at byte 96, also has its LFN attribute set, and its sequence number is 1. It has the same checksum as the first entry, 0xdf. The characters in this entry give us 'My Lo,' 'ng Fil,' and 'e.' The sequence number of this entry is 1, so we know it is the last one we will find. We can append the characters of the two entries to get 'My Long File Name.rtf.' The short version of this name can be found in the third entry, which is a normal directory entry and has the name 'MYLONG~1.RTF.'

We can now verify the checksum, but first we need to know the ASCII values for the characters in binary. These are given in Table 10.8.

Table 10.8. ASCII values for the characters in our example LFN.

Character	Hex	Binary
M	0x4d	0100 1101
Y	0x59	0101 1001
L	0x4c	0100 1100
O	0x4f	0100 1111
N	0x4e	0100 1110
G	0x47	0100 0111
~	0x7e	0111 1110

Table 10.8. ASCII values for the characters in our example LFN.

Character	Hex	Binary
1	0x31	0011 0001
R	0x52	0101 0010
T	0x54	0101 0100
F	0x46	0100 0110

For clarity, we will do this whole thing in binary instead of constantly translating. The first step is to assign our variable 'check' to the value of the first letter of the name, 'M.'

```
check = 0100 1101
```

For the remaining 10 rounds, we rotate the current checksum to the right by one bit and then add the next letter. The next two steps will shift our current value and add 'Y.'

```
check = 1010 0110
check = 1010 0110 + 0101 1001 = 1111 1111
```

We rotate (with no effect because it is all 1s) and add 'L.'

```
check = 1111 1111
check = 1111 1111 + 0100 1100 = 0100 1011
```

We rotate and add 'O.'

```
check = 1010 0101
check = 1010 0101 + 0100 1111 = 1111 0100
```

From now on, I'll leave out the rotate line and show only the addition. The next step is to rotate and add 'N.'

```
check = 0111 1010 + 0100 1110 = 1100 1000
```

We rotate and add 'G.'

```
check = 0110 0100 + 0100 0111 = 1010 1011
```

We rotate and add '~.'

```
check = 1101 0101 + 0111 1110 = 0101 0011
```

We rotate and add '1.'

```
check = 1010 1001 + 0011 0001 = 1101 1010
```

We rotate and add 'R.'

```
check = 0110 1101 + 0101 0010 = 1011 1111
```

We rotate and add 'T.'

```
check = 1101 1111 + 0101 0100 = 0011 0011
```

Finally, we rotate and add 'F.'

```
check = 1001 1001 + 0100 0110 = 1101 1111 = 0xdf
```

Hopefully, you will never have to do this by hand, but now you can at least say that you have seen it before. The final value of 0xdf is the same that we saw in each of the LFN entries.

As an example output of processing this directory entry, we can look at the `fls` tool from TSK. `fls` prints the LFN and puts the short name in parentheses, as shown here:

```
# fls -f fat fat-2.dd
r/r   3: FAT DISK     (Volume Label Entry)
r/r   4: RESUME-1.RTF
r/r   7: My Long File Name.rtf (MYLONG~1.RTF)
r/r * 8: _ile6.txt
```

The first two lines of the output show the volume label and short file name directory entries that we saw in the "Directory Entries" section. The third line shows the long name that we recently dissected and shows the name of a deleted file, `_ile6.txt`. The star in front of the name shows that it is deleted and the first letter is missing because the first letter of the name is used to set the unallocated status. The number before the name shows the address of the directory entry where the details can be found.

Summary

FAT has simplicity because of its small number of data structures. The boot sector and FAT are crucial to analyzing the file system in an automated way, and the directory entries are crucial to recovering deleted files without application-level techniques.

Bibliography

Microsoft. "ScanDisk May Not Fix the Media Descriptor Byte." Knowledge Base Article—158869, July 28, 2001. http://support.microsoft.com/default.aspx?scid=kb;en-us;158869.

Microsoft. "Encodings and Code Pages." Global Development and Computing Portal, 2004. http://www.microsoft.com/globaldev/getWR/steps/wrg_codepage.mspx.

Wilson, Craig . "Volume Serial Numbers & Format Verification Date/Time." Digital Detective White Paper, October 2003. http://www.digital-detective.co.uk/documents/Volume%20Serial%20Numbers.pdf.

See also the Bibliography section of Chapter 9.