Directory Structure

Introduction

Welcome to hell. This document is devoted to the structure of the special type of DOS files called directories.

Each directory consists of a number of fixed-size entries. Each entry is 32 bytes long. The number of sectors in the directory is fixed for the root directory on FAT12 and FAT16 partitions, and sectors are consecutive on disk. For non-root directories, as well as for the root directory on the FAT32 partitions, the number of sectors is not fixed, and the directory is stored according to the normal cluster chain.

There are two different types of the entries for long filenames and for aliases.

Aliases

The entry for an alias is:

Offset in the Entry	*Length in Bytes*	*Description*
00	8	Filename
08	3	Extension
0B	1	Attribute
0C	1	Case
0D	1	Creation time in ms
0E	2	Creation time
10	2	Creation date
12	2	Last access date
14	2	High word of starting cluster for FAT32
16	2	Time stamp
18	2	Date stamp
1A	2	Starting cluster
1C	4	Size of the file

Filename and extension are left-justified and blank-padded. Note that filename cannot consist solely of spaces, but extension can. Watch for illegal characters in filenames. My humble suggestion is replacing any illegal characters with underscores.

Some characters in the filename have special meaning. If the first character has the code `05`, then actually the first character has the code `E5` and it is not a special character. If the first character has the code `E5`, then the file was deleted. You may save some time when going through the directory structure by checking the first character in the filename. If it is zero, there are no more entries in current directory.

Two entries have a special meaning. They are present only in subdirectories, but not in the root directory. The entry with the name consisting of exactly one dot is the pointer to the root directory. Its starting cluster is the first cluster of the root directory, which is usually two. You are best advised to ignore this value because the location of the root directory can be easily calculated otherwise. The entry with the name made up of exactly two dots points to the next higher-level directory in the hierarchy. Its starting cluster is the first cluster in that directory. These entries should be the first and the second one in the directory, correspondingly. Their attributes should be `10h` (Directory). They are created at the time the subdirectory is created. There are no corresponding long names for them.

Attribute is a collection of bit flags:

Value	*Meaning*
01	Read Only
02	Hidden
04	System
08	Volume Label
10	Directory
20	Archive
40	Unused
80	Unused

Read Only, Hidden, and System are pretty self-explanatory. I will only note that neither Hidden nor System files should be moved during defragmentation or any other disk service. If you remember, I recommended certain actions when file corruption is detected. You are best advised not to try any corrections on Hidden or System files. Also, Hidden files should not be returned by search commands unless they were explicitly asked to do so.

Volume label attribute means that this entry contains the disk label in the filename and extension fields. Volume label is valid only in the root directory. Common sense says, there should be only one volume label per disk. For the entry to really contain the volume label, the attribute should be exactly `08`. *If Attribute is equal to* `0Fh` *(Read Only, Hidden, System, Volume Label) then this entry does not contain the alias, but it is used as part of the long filename or long directory name.*

Directory bit is set if the entry is a subdirectory. In this case the starting cluster contains the beginning cluster for the subdirectory, and the file size field is ignored (set to zero). Directories can also be Read Only, Hidden, System, or Archive. Directory bit is not set for the long directory name entries.

Archive bit is somewhat symbolic. It should be set if the file was not archived by the backup utility. Never in my life I have seen the use of this bit.

Two values are unused, which means that the entries with either of these bits set should be considered invalid. Another invalid combination is when both, Directory and Volume Label bits are set. Unless you are a disk analyzing tool, the best technique is to ignore the entries with the invalid attribute.

Case is zero if the filename and extension need to be converted to upper case. This field is used only by Windows NT.

Time stamp and creation time have the following format:

Bits	*Range*	*Translated Range*	*Valid Range*	*Description*
`0..4`	`0..31`	`0..62`	`0..59`	`Seconds/2`
`5..10`	`0..63`	`0..63`	`0..59`	`Minutes`
`11..15`	`0..31`	`0..31`	`0..23`	`Hours`

Date stamp, last accessed date, and creation date have the following format:

Bits	*Range*	*Translated Range*	*Valid Range*	*Description*
`0..4`	`0..31`	`0..31`	`1..28 up to 1..31`	`Day, blame Julian for complexity`
`5..8`	`0..15`	`0..15`	`1..12`	`Month`
`9..15`	`0..127`	`1980..2107`	`1980..2107`	`Year, add 1980 to convert`

Generally, creation time and date say when the file was created. Accessed time and date say when the file was last modified. Time and date stamps are set to the time that applications want you to think is the time of the last modification.

Starting cluster is the beginning cluster for the file or directory cluster chain. For FAT32, this value consists of the two 16-bit words, and the high four bits of the high word should be masked out. I have never seen any documentation regarding this, but a couple of hours of playing with FAT32 convinced me that this is the case.

Size of the file specifies the real file size in bytes. This value might be in conflict with the file size calculated by going through the cluster chain. Whenever they are in conflict, the smaller value takes over.

Long Filenames

The entries for long filenames look pretty odd because Microsoft tried to maintain compatibility with the older software. However, their format has not changed in FAT32, which sounds somewhat ironic because FAT32 is in no way compatible with the older software. You may notice that these entries look much similar to those of aliases. The difference is, they use the strange combination of attributes, so they are likely to be skipped by the older software. The word "slot" was used for these entries by Galen C. Hunt, and I will stay with his terminology.

Offset in Entry	*Length in Bytes*	*Description*
00	1	Sequence number for the slot
01	10d	First five characters in filename
0B	1	Attribute
0C	1	Reserved, always zero
0D	1	Alias checksum
0E	12d	Next six characters in filename
1A	2	Starting cluster
1C	4	Last two characters in filename

The starting cluster number is always zero, and the attribute is always 0Fh.

Slots are always positioned right before the alias in the directory. The closest to the alias slot contains the first thirteen characters of the long filename. The slot above it contains the next thirteen characters, and so on, up to 256 characters. Additionally, the sequence number of the slot contains its number in the slot chain, starting from one. The sequence number for the last slot in the chain is or'ed with 40h to indicate end of chain.

Slot Number	*Sequence Number*	*Characters*
3	43h	me.text
2	02	y long filena
1	01	This is a ver
Alias	Alias	THISIS~1.TEX

If the length of the filename is not the multiple of thirteen, the name in null-terminated. Otherwise, it is not null-terminated. If after null termination there are any characters left in the slot, they are filled with `FFFF`.

Checksum contains the checksum for the corresponding alias. It is calculated in the following way:

```
unsigned char sum, i;
for(sum=i=0; i<11; i++)
    sum=(((sum&1)<<7)|((sum&0xFE)>>1))+name[i];
```

In a more common language, they rotate the sum right with cycling and add the next character at each iteration. Note that the checksum is case-sensitive.

When the file is deleted, all entries for the long filename start with `E5`.

What can go wrong with long filenames? Give some space to your imagination...

Operations

Let us perform some operations with long filenames and aliases. I do not describe the steps that are related to cluster chain management because they were already described. As for search operations, they should really be performed in a different way to be any quick. However, the directory structure itself does not suggest anything other than a sequential search.

Find File by Alias

1. Start from the top of the directory.
2. Check the first byte of the entry. If it is zero, finish.
3. Then check the attribute. Skip the entry if the attribute is invalid, `0F`, or does not correspond to the specific search demands.
4. Finally, compare the filename and extension with the search pattern. Don't forget about wildcards.
5. If you have not found the needed entry yet, go to the next entry. Watch for the end of the directory. If not end of the directory, go to step two.

Find File by Long Name

1. Start from the top of the directory.
2. Check the first byte of the entry. Exit the loop if it is zero. Skip to the next entry if it is not one and not `41h`.
3. Retrieve the long name. Skip to the next entry in case of an error at this stage. Compare the long name against your search string. If you search operation has any restrictions for attributes, date, etc. use the fields of the corresponding alias for comparison. Skip to the next entry if the current one doesn't match.
4. If you have not found the needed entry, go to the next one. Watch for end of directory.

Retrieve the Long Name

1. Start from the entry with the sequence number one or `41h`. Start with an empty string. The string must be at least 512 bytes long. Any error in the first entry should be interpreted as an invalid long name. Any error in the next entries should end up in the valid but truncated long name. Invalid long name is signaled by the empty string on return.
2. Check the attribute. It must be `0Fh`.
3. If sequence number is one or `41h`, check the next entry in the directory. It must be the alias for the long name. Check if its attribute is valid. Calculate the checksum.
4. Compare the checksum in the slot with the checksum calculated for the alias. They must be equal.
5. Concatenate the current string and the strings in the slot. Remember that strings may be null-terminated. Watch for the invalid characters and replace them with underscores.
6. If sequence number and `40h`, or the string in the slot is null-terminated, or out of buffer space, return your current string.
7. Else go to the previous entry in the directory. Watch for the beginning of the directory.

Allocate New Entry

When you need to allocate a new entry in the directory, look through the entries from the beginning. Use the first one that is marked as deleted or the first one that has zero as the first character of the filename. Expand the directory if no entries are free. Also follow this simple algorithm to allocate the block of consecutive entries. You will need it for aliasing. You can make it a bit smarter by overwriting the deleted entries only after all free entries are used up, but because the probability that deleted files will be recovered is low and memory for the directory cache is expensive, I do not recommend it.

Delete and Undelete File or Directory

To delete the file:

1. Set the first character of the alias to `E5`.
2. Set the first byte for each of the slots for the long name to `E5`.
3. Fill the cluster chain with zeroes.

To delete the directory, follow these very steps, but first check if the directory to be deleted is empty. Empty directory consists of no entries but `"."`, `".."`, and, possibly, deleted entries. Do not delete the directory unless it is empty.

Because the directory entry is otherwise untouched, recovery is possible. However, recovery is not guaranteed. Furthermore, it is not guaranteed that the recovered file will have the same contents as the original file. To recover the deleted file:

1. Find the directory entry marked as deleted. Begin from the starting cluster and check the number of the consecutive FAT entries corresponding to the file size. If they all are zero, build a file chain through them and conclude that undeleting is completed. If any is marked as physically bad, just skip it. Otherwise, conclude that undeleting is impossible.
2. Ask the user for the first character in the filename, or get the first character from the long filename. Write the first character to the directory entry (for the alias).
3. Read the previous directory entry, if any. If this entry is also marked as deleted and its attribute is `0F`, this is probably the first slot for the long filename. Continue going up the directory while these conditions are true. Build the chain for the long filename.

Create New Directory

1. Allocate the necessary number of entries. You should have enough space to store the alias and, possibly, the long filename. The case when the long filename is not created was mentioned earlier.
2. Allocate at least one cluster. The recommended number of clusters is such a number that can contain 512 directory entries.
3. Write the first two required entries "." and "..".
4. Write the alias and the long filename. Set the beginning cluster field of the alias to the just allocated cluster chain.