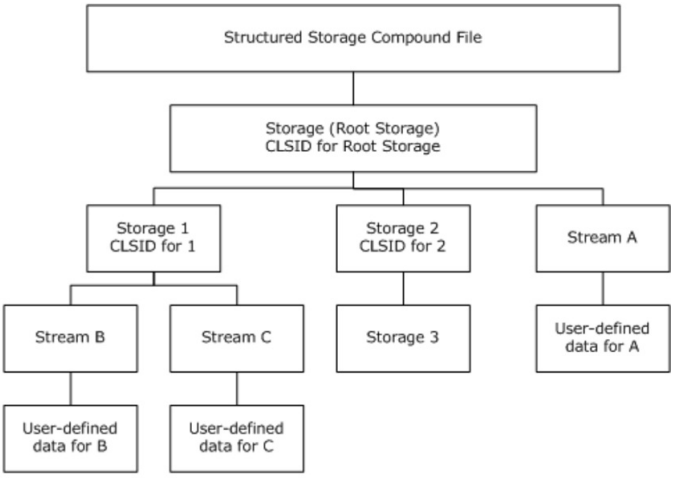
Report for
85700ce85d1e0206a9d4f7558b1e1bc54014df934d15bfbe185f83d1f10452b2

OLE Format

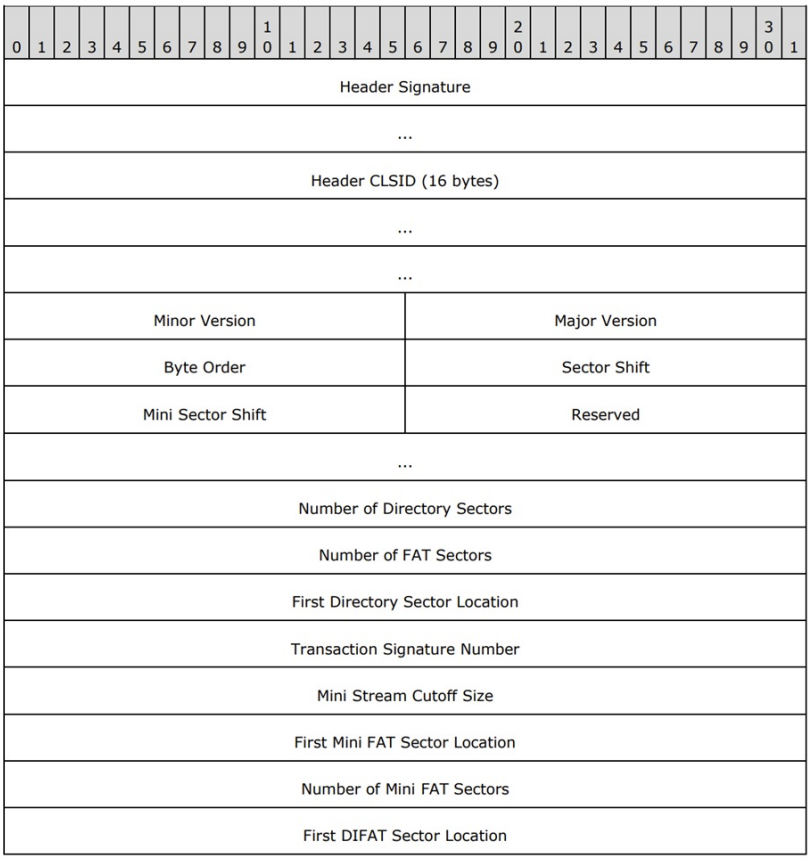
We will not be discussing the whole OLE format itself but rather we will be discussing some parts of it to give insight as to how OLE file can be parsed. Microsoft has improved their documentation for OLE format which can be found here: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cfb/53989ce4-7b05-4f8d-829b-d08d6148375b

OLE format, initially called compound binary format, is based on file system called FAT (File Allocation Table). Where FAT has its main header that is located on the first sector of the hard drive and contains information and/or pointers to other files and directories within the hard drive. Similarly, OLE is a structure that stores storage object and stream objects. Storage objects are just like directories that may store another directories and files and on this case, it stores storage objects and stream objects. And stream objects are just like the traditional files.



Compound binary format (Reference: Microsoft Compound Binary Documentation)

An OLE file contains Compound File Header which has the header signature as well as other information such as byte order, number of FAT sectors, sector shift/size, mini sector shift/size, location of first directory sector, mini stream cutoff size, location of first mini FAT sector, number of mini-FAT sectors and DIFAT.

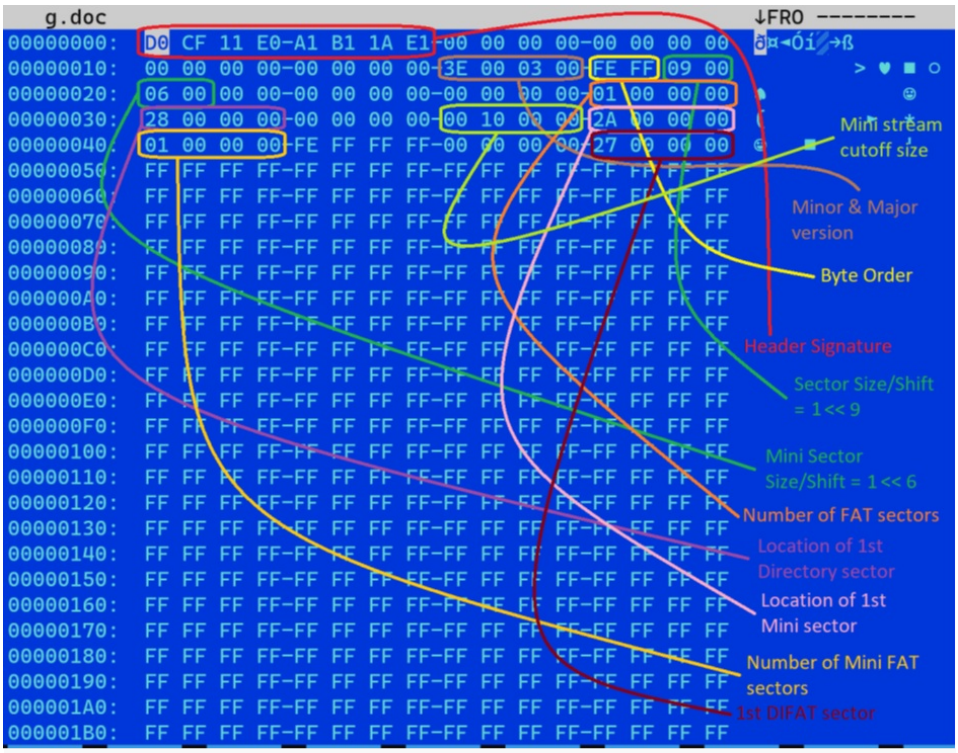


Specifications of OLE Header (Reference: Microsoft Compound Binary Documentation)

In a file system, a file can be sub-divided into several sectors which are located in different parts of the hard-drive. Same goes with OLE files, it is sub-divided into equal length of sectors. The size of Compound File Header is usually 512 bytes depending on version. Objects that are stored in sectors are linked list called sector chains where each sector is located in different parts of the file. The idea of this sector chain is to let you know which sector can be found the other parts of the object. And to let you know that it is the last sector of the chain, it has a value of 0xFFFFFFFE.

Now, mini sector is introduced to maximize the storage as sector is always allocated based on sector size e.g. 512 bytes. And if there are objects smaller than 512 bytes, that can be a potential waste.

The file has the following Compound Binary File Header:



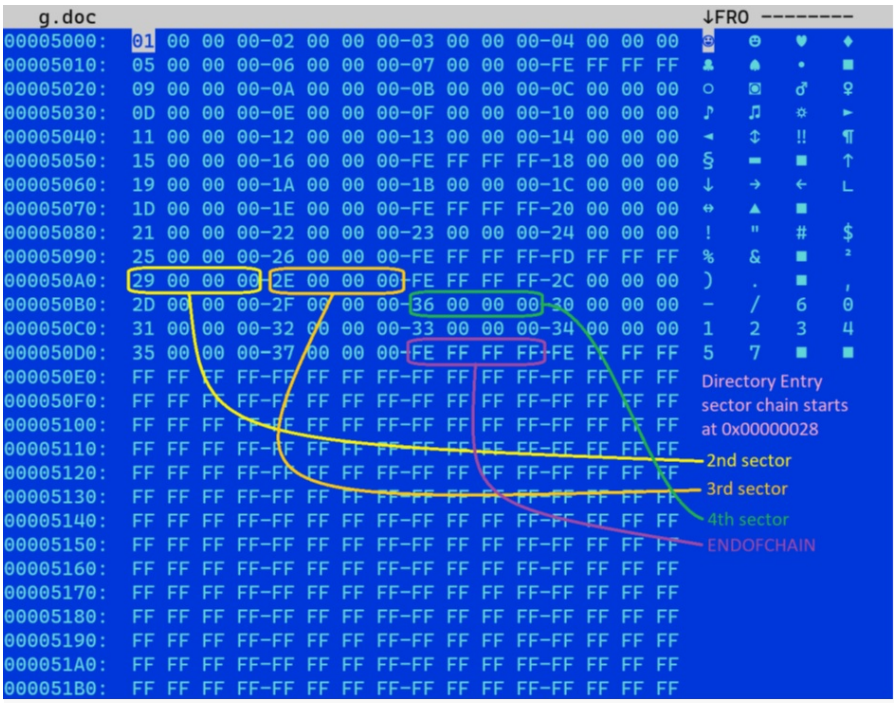
Hex view of Compound File Header with its corresponding data

Keep in mind that the â€œlocationâ€ of sectors is basically index to the FAT sector array or Mini FAT sector array. And you can also calculate its corresponding fileoffset using the following calculation:

```
fileOffset = sizeOfHeader + sector (1 << sectorShift)
```

Now, to build the FAT sector array, we need the first DIFAT index and succeeding DIFAT indices if thereâ€™s any, On this example, thereâ€™s only one DIFAT sector. To calculate the fileoffset of DIFAT sector location is shown below:

```
fileOffsetDIFAT = sizeOfHeader + firstDIFATSector (1 << 9)
fileOffsetDIFAT = 0x200 + 0x00000027 (0x200)
fileOffsetDIFAT = 0x5000
```



DIFAT sector that contains FAT sector array where one could build FAT sector chain

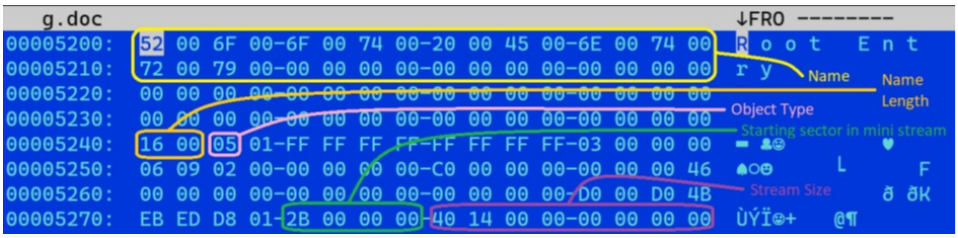
Letâ€™s check the first directory entries of the said file. Looking at the Compound File Header, we know that the sector of the first directory entry is 0x00000028. And to get the sector chain for the directory entry, we will look at the FAT sector array which we will have the following:

```
startOffset of FAT sector array: [0x5000]
next sector for Directory Entry: [0x5000 + 0x00000028 (4)] = 0x00000029
next sector for Directory Entry: [0x5000 + 0x00000029 (4)] = 0x0000002E
next sector for Directory Entry: [0x5000 + 0x0000002E (4)] = 0x00000036
next sector for Directory Entry: [0x5000 + 0x00000036 (4)] = 0xFFFFFFFE (ENDOFCHAIN)
sector chain of Directory Entry: (0x00000028, 0x00000029, 0x0000002E, 0x00000036, 0xFFFFFFFE)
```

Now we know that there 4 sectors in the sector chain for Directory Entry. To get the locations of Directory Entries, we do the following calculation:

```
DirectoryEntry[0] = sizeOfHeader + sector (sectorSize)
DirectoryEntry[0] = 0x200 + 0x00000028 (1 << 9)
DirectoryEntry[0] = 0x5200 (fileoffset)
```

Same calculation goes to the next sectors in the chain.



Location of first Directory Entry in an OLE file with its corresponding data

The first directory entry is root storage which has a name of â€˜Root Entryâ€™. It follows a data structure and below are some of the data within the directory entry structure (you may find other data in Microsoft documentation):

`Name Length â€“ 0x0016 bytes`
`Object Type â€“ 0x05 which means root storage. Other types could be 0x01 (storage object), 0x02 (stream object) and 0x00 (Unknown or unallocated)`
`Stream Size â€“ 0x0000000000001440 bytes`
`Starting Sector of Mini Stream â€“ 0x0000002B (this is needed when building the mini stream)`

Looking at each sector of Directory Entries, we will find the following entry names together with its object types and sizes:

`'Root Entry' (root) 5184 bytes`
`'\x01CompObj' (stream) 114 bytes`
`'\x05DocumentSummaryInformation' (stream) 4096 bytes`
`'\x05SummaryInformation' (stream) 4096 bytes`
`'1Table' (stream) 7179 bytes`
`'Macros' (storage)`
`'PROJECT' (stream) 374 bytes`
`'PROJECTwm' (stream) 41 bytes`
`'VBA' (storage)`
`'ThisDocument' (stream) 1502 bytes`
`'_VBA_PROJECT' (stream) 2460 bytes`
`'dir' (stream) 521 bytes`
`'WordDocument' (stream) 4096 bytes`

To get the data or contents of the streams, we will refer to the mini-FAT sector array if its size is below the mini stream cutoff size (4096 bytes). And if the stream size is greater than or equal to the mini stream cutoff size (4096 bytes), we will refer to the standard FAT sector array. Letâ€™s take for example the â€˜WordDocumentâ€™ stream with a size of 4096 bytes. Therefore, to get the sector chain â€˜WordDocumentâ€™ stream, we will refer to FAT sector array located as shown below:

`startOffset of FAT sector array: 0x5000`
`next sector for WordDocument stream: [0x5000 + 0x00000000 (4)] = 0x00000001`
`next sector for WordDocument stream: [0x5000 + 0x00000001 (4)] = 0x00000002`
`next sector for WordDocument stream: [0x5000 + 0x00000002 (4)] = 0x00000003`
`...`
`next sector for WordDocument stream: [0x5000 + 0x00000006 (4)] = 0x00000007`
`next sector for WordDocument stream: [0x5000 + 0x00000007 (4)] = 0xFFFFFFFE (ENDOFCHAIN)`

`sector chain for WordDocument stream: (0x00000000, 0x00000001, 0x00000002, 0x00000003, 0x00000004, 0x00000005, 0x00000006, 0x00000007, 0xFFFFFFFE)`

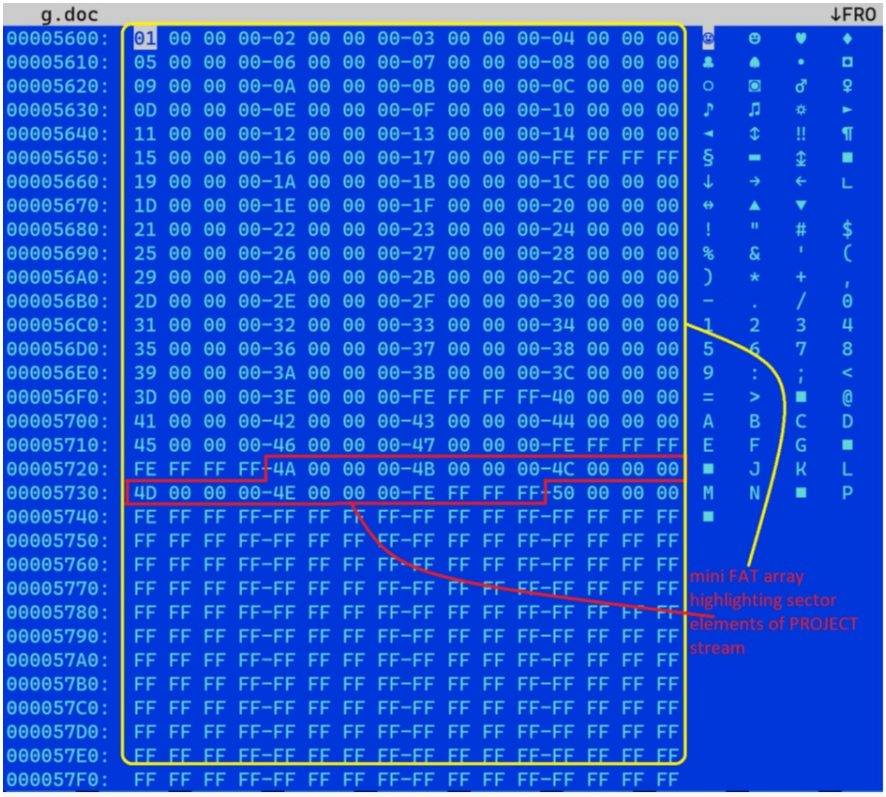
And to build the stream for WordDocument byte-per-byte, hereâ€™s a pseudo-code:

```
sectorChainWordDocument = [0, 1, 2, 3, 4, 5, 6, 7]
WordDocumentStream = [ ]
sizeOfHeader = 0x200
sectorSize = 1 << 9
```

```
for n in sectorChainWordDocument:
   foffset = sizeOfHeader + (n * sectorSize)
   for i in range(sectorSize):
       ole.seek(foffset)
       WordDocument.append(ole.read(1))
       foffset+=1
```

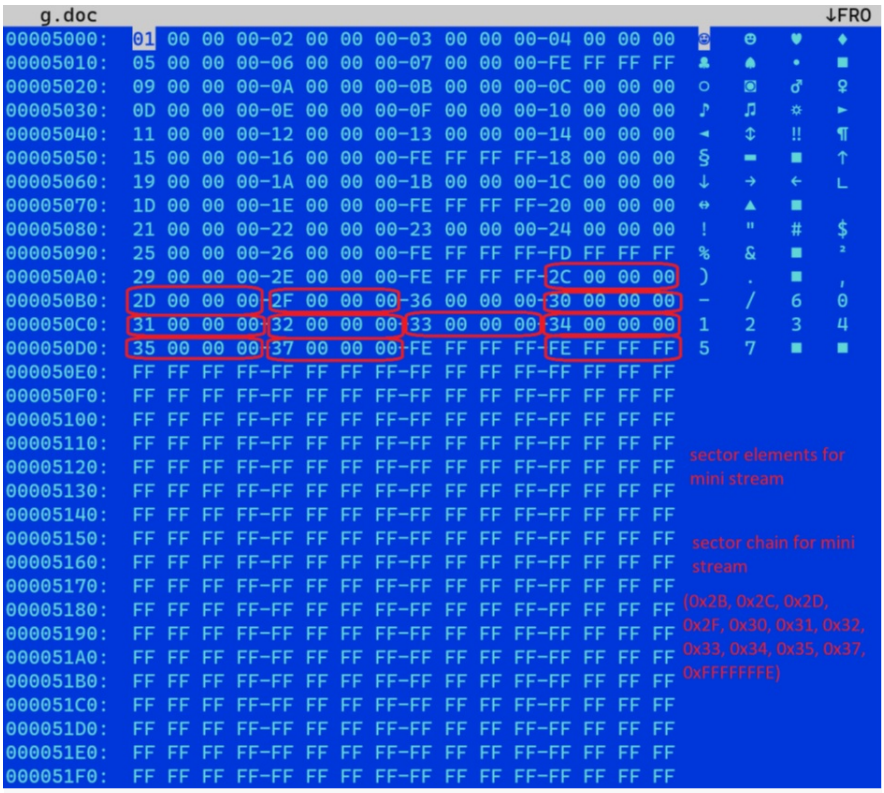
In addition, from the Directory Entries, we can see that there are streams that are less than the mini stream cutoff size (4096). This means that the data/contents of those streams are referenced in mini FAT sector array, and on this sample, thereâ€™s only one Mini FAT sector.
And from the Compound File Header, we can see the starting sector of the mini FAT sector. Hereâ€™s how to calculate its location:

```
fileOffsetMiniFAT = sizeOfHeader + firstMiniFATSector (1 << sectorShift)
fileOffsetMiniFAT = 0x200 + 0x0000002A (1 << 9)
fileOffsetMiniFAT = 0x5600
```



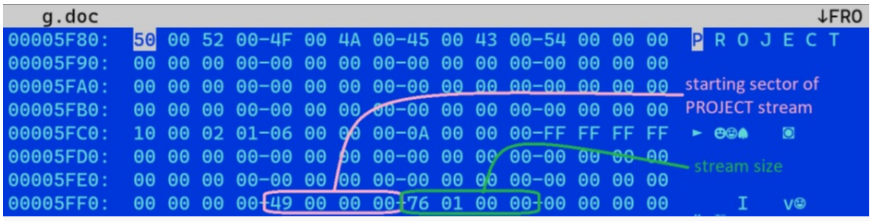
Mini FAT array that shows mini sectors for PROJECT

Now, the first sector of the mini stream is specified in Root Entry as mentioned previously, which is 0x0000002B. And to build the mini stream sector chain, we reference it from FAT sector array which is shown below:



sector chain of Mini Stream

Letâ€™s take for example the stream PROJECT below. Its size is 174 bytes, this means that its contents are referenced in mini FAT array since its size is less than the mini stream cutoff size (4096).



PROJECT stream showing its first sector location and its size

To get the contents of PROJECT stream, first we need to build the Mini Stream. And the starting sector of mini stream is specified in Root Entry which is 0x0000002B. With starting sector, we can build the sector chain for mini stream which is: [0x2B, 0x2C, 0x2D, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x37]. And to get the contents of Mini Stream byte-per-byte, hereâ€™s a pseudo-code:

```
mini_stream = [0x2B, 0x2C, 0x2D, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x37]
mini_stream_data = []
sectorSize = 1 << 9
for n in mini_stream:
   foffset = 0x200 + n * sectorSize
   for i in range(sectorSize):
       ole.fp.raw.seek(foffset)
       mini_stream_data.append(ole.fp.raw.read(1))
       foffset+=1
```

The pseudo-code above tells us that the Mini Stream contents are now in mini_stream_data array.

Now, PROJECT stream has a starting sector of 0x00000049 and size of 0x0176, then this stream is referenced within mini FAT, again its size is lower than the mini stream cut-off size. Its sector chain is [0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E]. And to get all of PROJECT stream contents, hereâ€™s a pseudo-code:

```
proj_sect_chain = [0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E]
proj_data = []
proj_size = 0x0176
mini_sect_size = 1 << 6
total_read = 0
for n in proj_sect_chain:
   idx = n * mini_sect_size
   if (total_read+mini_sect_size) >= proj_size:
       proj_data.append(b''.join(mini_stream_data[idx:(idx+(proj_size-total_read))]))
       total_read+=proj_size-total_read
   else:
       proj_data.append(b''.join(mini_stream_data[idx:(idx+mini_sect_size)]))
       total_read+=mini_sect_size
print(b''.join(proj_data))
print(total_read)
```

And its contents shows below:

```
b'ID="{31B7E753-2B2D-4C35-9D66-
A1276E9FFC7D}"\r\nDocument=ThisDocument/&H00000000\r\nName="Project"\r\nHelpContextID="0"\r\nVersionC
ompatible32="393222000"\r\nCMG="95976DB893EB97EB97EB97EB97"\r\nDPB="CCCE34F15C295D295D29"\r\nGC="0301
FBCE33CF33CFCC"\r\n\r\n[Host Extender Info]\r\n&H00000001={3832D640-CF90-11CF-8E43-
00A0C911005A};VBE;&H00000000\r\n\r\n[Workspace]\r\nThisDocument=49, 49, 1901, 489, Z\r\n'
```