POI-HSLF - A Guide to the PowerPoint File Format

Overview

by Nick Burch, Yegor Kozlov

1. Records, Containers and Atoms

PowerPoint documents are made up of a tree of records. A record may contain either other records (in which case it is a Container), or data (in which case it's an Atom). A record can't hold both.

PowerPoint documents don't have one overall container record. Instead, there are a number of different container records to be found at the top level.

Any numbers or strings stored in the records are always stored in Little Endian format (least important bytes first). This is the case no matter what platform the file was written on - be that a Little Endian or a Big Endian system.

PowerPoint may have Escher (DDF) records embeded in it. These are always held as the children of a PPDrawing record (record type 1036). Escher records have the same format as PowerPoint records.

2. Record Headers

All records, be they containers or atoms, have the same standard 8 byte header. It is:

- 1/2 byte container flag
- 1.5 byte option field
- 2 byte record type
- 4 byte record length

If the first byte of the header, BINARY_AND with 0x0f, is 0x0f, then the record is a container. Otherwise, it's an atom. The rest of the first two bytes are used to store the "options" for the record. Most commonly, this is used to indicate the version of the record, but the exact useage is record specific.

The record type is a little endian number, which tells you what kind of record you're dealing

with. Each different kind of record has it's own value that gets stored here. PowerPoint records have a type that's normally less than 6000 (decimal). Escher records normally have a type between 0xF000 and 0xF1FF.

The record length is another little endian number. For an atom, it's the size of the data part of the record, i.e. the length of the record *less* its 8 byte record header. For a container, it's the size of all the records that are children of this record. That means that the size of a container record is the length, plus 8 bytes for its record header.

3. CurrentUserAtom, UserEditAtom and PersistPtrIncrementalBlock

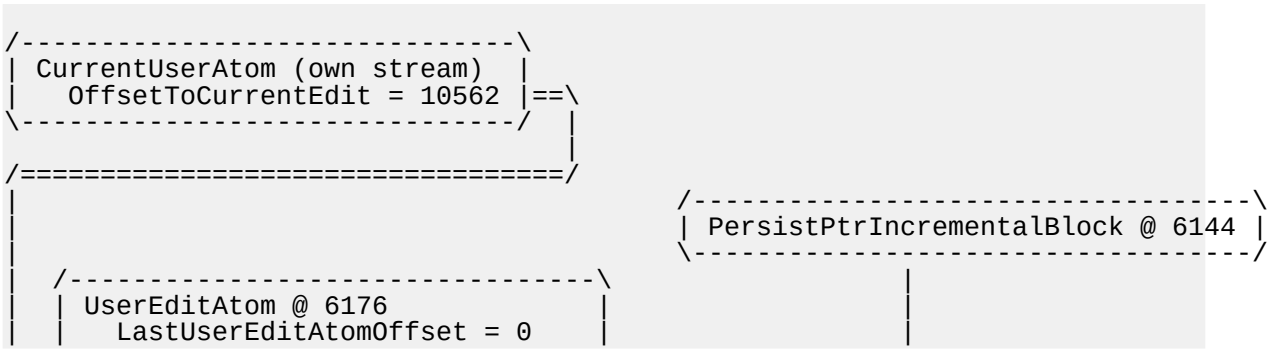
aka Records that care about the byte level position of other records

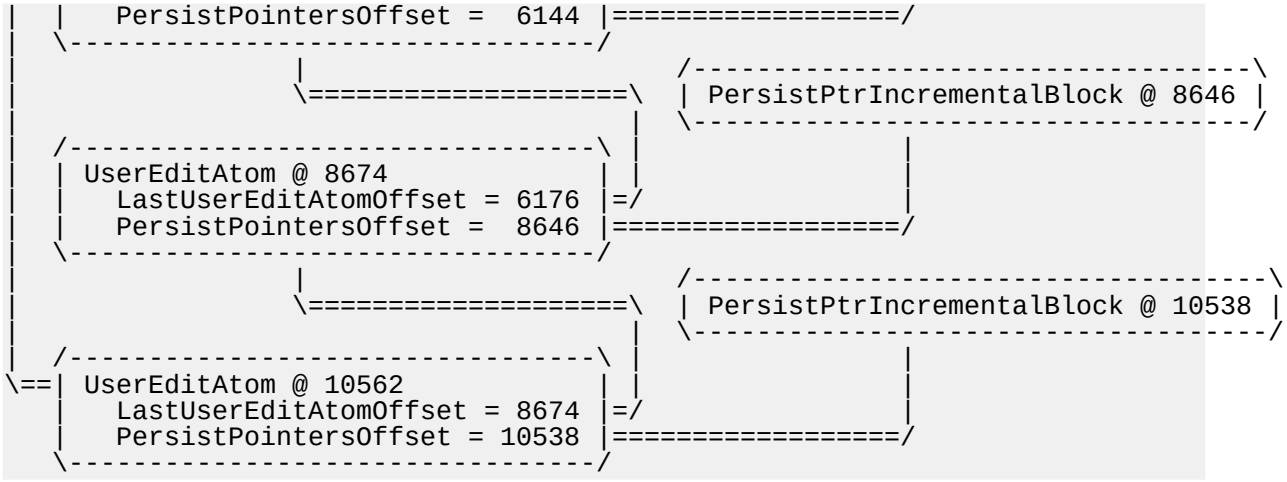
A small number of records contain byte level position offsets to other records. If you change the position of any records in the file, then there's a good chance that you will need to update some of these special records.

First up, CurrentUserAtom. This is actually stored in a different OLE2 (POIFS) stream to the main PowerPoint document. It contains a few bits of information on who lasted edited the file. Most importantly, at byte 8 of its contents, it stores (as a 32 bit little endian number) the offset in the main stream to the most recent UserEditAtom.

The UserEditAtom contains two byte level offsets (again as 32 bit little endian numbers). At byte 12 is the offset to the PersistPtrIncrementalBlock associated with this UserEditAtom (each UserEditAtom has one and only one PersistPtrIncrementalBlock). At byte 8, there's the offset to the previous UserEditAtom. If this is 0, then you're at the first one.

Every time you do a non full save in PowerPoint, it tacks on another UserEditAtom and another PersistPtrIncrementalBlock. The CurrentUserAtom is updated to point to this new UserEditAtom, and the new UserEditAtom points back to the previous UserEditAtom. You then end up with a chain, starting from the CurrentUserAtom, linking back through all the UserEditAtoms, until you reach the first one from a full save.





The PersistPtrIncrementalBlock contains byte offsets to all the Slides, Notes, Documents and MasterSlides in the file. The first PersistPtrIncrementalBlock will point to all the ones that were present the first time the file was saved. Subsequent PersistPtrIncrementalBlocks will contain pointers to all the ones that were changed in that edit. To find the offset to a given sheet in the latest version, then start with the most recent PersistPtrIncrementalBlock. If this knows about the sheet, use the offset it has. If it doesn't, then work back through older PersistPtrIncrementalBlocks until you find one which does, and use that.

Each PersistPtrIncrementalBlock can contain a number of entries blocks. Each block holds information on a sequence of sheets. Each block starts with a 32 bit little endian integer. Once read into memory, the lower 20 bits contain the starting number for the sequence of sheets to be described. The higher 12 bits contain the count of the number of sheets described. Following that is one 32 bit little endian integer for each sheet in the sequence, the value being the offset to that sheet. If there is any data left after parsing a block, then it corresponds to the next block.

hex on disk	decimal	description
0000	0	No options
7217	6002	Record type is 6002
2000 0000	32	Length of data is 32 bytes
0100 5000	5242881	Count is 5 (12 highest bits) Starting number is 1 (20 lowest bits)
0000 0000	0	Sheet (1+0)=1 starts at offset 0
900D 0000	3472	Sheet (1+1)=2 starts at offset 3472
E403 0000	996	Sheet (1+2)=3 starts at offset 996
9213 0000	5010	Sheet (1+3)=4 starts at offset 5010
BE15 0000	5566	Sheet (1+4)=5 starts at offset 5566
0900 1000	1048585	Count is 1 (12 highest bits) Starting number is 9 (20 lowest bits)
4418 0000	6212	Sheet (9+0)=9 starts at offset 9212

4. Paragraph and Text Styling

There are quite a number of records that affect the styling of text, and a smaller number that are responsible for the styling of paragraphs.

By default, a given set of text will inherit paragraph and text stylings from the appropriate master sheet. If anything differs from the master sheet, then appropriate styling records will follow the text record.

(We don't currently know enough about master sheet styling to write about it)

Normally, powerpoint will have one text record (TextBytesAtom or TextCharsAtom) for every paragraph, with a preceeding TextHeaderAtom to describe what sort of paragraph it is. If any of the stylings differ from the master's, then a StyleTextPropAtom will follow the text record. This contains the paragraph style information, and the styling information for each section of the text which has a different style. (More on StyleTextPropAtom later)

For every font used, a FontEntityAtom must exist for that font. The FontEntityAtoms live inside a FontCollection record, and there's one of those inside Environment record inside the Document record. *(More on Fonts to be discovered)*

5. StyleTextPropAtom

If the text or paragraph stylings for a given text record differ from those of the appropriate master, then there will be one of these records.

This record is made up of two lists of lists. Firstly, there's a list of paragraph stylings - each made up of the number of characters it applies two, followed by the matching styling elements. Following that is the equivalent for character stylings.

Each styling list (in either list) starts with the number of characters it applies to, stored in a 2 byte little endian number. If it is a paragraph styling, it will be followed by a 2 byte number (of unknown use). After this is a four byte number, which is a mask indicating which stylings will follow. You then have an entry for each of the stylings indicated in the mask. Finally, you move onto the next set of stylings.

Each styling has a specific mask flag to indicate its presence. (The list may be found towards the top of org.apache.poi.hslf.record.StyleTextPropAtom.java, and is too long to sensibly include here). For each styling entry will occur in the order of its mask value (so one with mask 1 will come first, followed by the next higest mask value). Depending on the styling, it is either made up of a 2 byte or 4 byte numeric value. The meaning of the value will depend on the styling (eg for font.size, it is the font size in points).

Some stylings are actually mask stylings. For these, the value will be a 4 byte number. This is then processed as mask, to indicate a number of different sub-stylings. The styling for bold/italic/underline is one such example.

```
hex on disk      decimal        description
-----------      -------        -----------

0000             0              No options
A10F             4001           Record type is 4001
8000 0000        128            Length of data is 128 bytes
1E00 0000        30             The paragraph styling applies to 30 characters
0000             0              Paragraph options are 0
0018 0000        6144           0x0800=Text Alignment, 0x1000=Line Spacing
0000             0              Text Alignment = Left
5000             80             Line Spacing = 80

1C00 0000        28             The paragraph styling applies to 28 characters
0000             0              Paragraph options are 0
0010 0000        4096           0x1000=Line Spacing
5000             80             Line Spacing = 80

1900 0000        25             The paragraph styling applies to 25 characters
0000             0              Paragraph options are 0
0018 0000        6144           0x0800=Text Alignment, 0x1000=Line Spacing
0200             0              Text Alignment = Right
5000             80             Line Spacing = 80

6100 0000        61             The paragraph styling applies to 61 characters
                                (includes final CR)
0000             0              Paragraph options are 0
0018 0000        6144           0x0800=Text Alignment, 0x1000=Line Spacing
0000             0              Text Alignment = Left
5000             80             Line Spacing = 80

1E00 0000        30             The character styling applies to 30 characters
0100 0200        131073         0x0001=Char Props Mask, 0x20000=Font Size
0100             1              Char Props 0x0001=Bold
1400             20             Font Size = 20

1C00 0000        28             The character styling applies to 28 characters
0200 0600        393218         0x0002=Char Props Mask, 0x20000=Font Size,  0x40000=Fon
0200             2              Char Props 0x0002=Italic
1400             20             Font Size = 20
0000 0005        83886080       Blue

1900 0000        25             The character styling applies to 25 characters
0000 0600        393216         0x20000=Font Size,  0x40000=Font Color
1400             20             Font Size = 20
FF33 00FE        4261426175     Red

6000 0000        96             The character styling applies to 96 characters
0400 0300        196612         0x0004=Char Props Mask, 0x10000=Font Index, 0x20000=Fon
```

```
0400            4               Char Props 0x0004=Underlined
0100            1               Font Index = 1 (2nd Font in table)
1800            24              Font Size = 24
```

6. Fonts in PowerPoint

PowerPoint stores information about the fonts used in FontEntityAtoms, which live inside Document.Environment.FontCollection. For every different font used, a FontEntityAtom must exist for that font. There is always at least one FontEntityAtom in Document.Environment.FontCollection, which describes the default font.

7. FontEntityAtom

The instance field of the record header contains the zero based index of the font. Font index entries in StyleTextPropAtoms will refer to their required font via this index.

The length of FontEntityAtoms is always 68 bytes. The first 64 bytes of it hold the typeface name of the font to be used. This is stored as a null-terminated string, and encoded as little endian unicode. (The length of the string must not exceed 32 characters including the null termination, so the typeface name cannot exceed 31 characters).

After the typeface name there are 4 bytes of bitmask flags. The details of these can be found in the Windows API, under the LOGFONT structure. The 65th byte is the output precision, which defines how closely the system chosen font must match the requested font, in terms of heigh, width, pitch etc. The 66th byte is the clipping precision, which defines how to clip characters that occur partly outside the clipping region. The 67th byte is the output quality, which defines how closely the system must match the logical font's attributes to those of the physical font used. The 68th (and final) byte is the pitch and family, which is used by the system when matching fonts.