POI-HSLF - A Guide to the PowerPoint File Format

Overview

by Nick Burch

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

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.

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.

```
/-------------------------------\
| CurrentUserAtom (own stream)  |
|   OffsetToCurrentEdit = 10562 |==\
\-------------------------------/  |
                                   |
/==================================/
|                                         /-----------------------------------\
|                                         | PersistPtrIncrementalBlock @ 6144 |
|                                         \-----------------------------------/
|  /---------------------------------\                  |
|  | UserEditAtom @ 6176             |                  |
|  |   LastUserEditAtomOffset = 0    |                  |
```

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.

```
|  |   PersistPointersOffset =  6144 |==================/
|  \---------------------------------/
|                 |                       /-----------------------------------\
|                 \====================\  | PersistPtrIncrementalBlock @ 8646 |
|                                      |  \-----------------------------------/
|  /---------------------------------\ |                |
|  | UserEditAtom @ 8674             | |                |
|  |   LastUserEditAtomOffset = 6176 |=/                |
|  |   PersistPointersOffset =  8646 |==================/
|  \---------------------------------/
|                 |                       /------------------------------------\
|                 \====================\  | PersistPtrIncrementalBlock @ 10538 |
|                                      |  \------------------------------------/
|  /---------------------------------\ |                |
\==| UserEditAtom @ 10562            | |                |
   |   LastUserEditAtomOffset = 8674 |=/                |
   |   PersistPointersOffset = 10538 |==================/
   \---------------------------------/
```

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
```

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.