POI-HPBF - A Guide to the Publisher File Format

Overview

by Nick Burch

1. Document Streams

The file is made up of a number of POIFS streams. A typical file will be made up as follows:

```
Root Entry -
  Objects -
    (no children)
  SummaryInformation <(0x05)SummaryInformation>
  DocumentSummaryInformation <(0x05)DocumentSummaryInformation>
  Escher -
    EscherStm
    EscherDelayStm
  Quill -
    QuillSub -
      CONTENTS
      CompObj <(0x01)CompObj>
  Envelope
  Contents
  Internal <(0x03)Internal>
  CompObj <(0x01)CompObj>
  VBA -
    (no children)
```

2. Changing Text

If you make a change to the text of a file, but not change how much text there is, then the *CONTENTS* stream will undergo a small change, and the *Contents* stream will undergo a large change.

If you make a change to the text of a file, and change the amount of text there is, then both the *Contents* and the *CONTENTS* streams change.

3. Changing Shapes

If you alter the size of a textbox, but make no text changes, then both *Contents* and *CONTENTS* streams change. There are no changes to the Escher streams.

If you set the background colour of a textbox, but make no changes to the text, (to finish off)

4. Structure of CONTENTS

First we have "CHNKINK ", followed by 24 bytes.

Next we have 20 sequences of 24 bytes each. If the first two bytes at 0x1800, then that sequence entry exists, but if it's 0x0000 then the entry doesn't exist. If it does exist, we then have 4 bytes of upper case ASCII text, followed by three little endian shorts. The first of these seems to be the count of that type, the second is usually 1, the third is usually zero. The we have another 4 bytes of upper case ASCII text, normally but not always the same as the first text. Finally, we have an unsigned little endian 32 bit offset to the start of the data for this, then an unsigned little endian 32 bit offset of the length of this section.

Normally, the first sequence entry is for TEXT, and the text data will start at 0x200. After that is normally two or three STSH entries (so the first short has values 0, then 1, then 2). After that it seems to vary.

At 0x200 we have the text, stored as little endian 16 bit unicode.

After the text comes all sorts of other stuff, presumably as described by the sequences.

For a contents stream of length 7168 / 0x1c00 bytes, the start looks something like:

```
CHNKINK       // "CHNKINK "
04 00 07 00   // Normally 04 00 07 00
13 00 00 03   // Normally ## 00 00 03
00 02 00 00   // Normally 00 ## 00 00
00 1c 00 00   // Normally length of the stream
f8 01 13 00   // Normally f8 01 11/13 00
ff ff ff ff   // Normally seems to be ffffffff

18 00
TEXT 00 00 01 00 00 00       // TEXT 0 1 0
TEXT 00 02 00 00 d0 03 00 00 // TEXT from: 200 (512), len: 3d0 (976)
18 00
STSH 00 00 01 00 00 00       // STSH 0 1 0
STSH d0 05 00 00 1e 00 00 00 // STSH from: 5d0 (1488), len: 1e (30)
18 00
STSH 01 00 01 00 00 00       // STSH 1 1 0
STSH ee 05 00 00 b8 01 00 00 // STSH from: 5ee (1518), len: 1b8 (440)
18 00
STSH 02 00 01 00 00 00       // STSH 2 1 0
STSH a6 07 00 00 3c 00 00 00 // STSH from: 7a6 (1958), len: 3c (60)
18 00
```

```
FDPP 00 00 01 00 00 00       // FDPP 0 1 0
FDPP 00 08 00 00 00 02 00 00 // FDPP from: 800 (2048), len: 200 (512)
18 00
FDPC 00 00 01 00 00 00       // FDPC 0 1 0
FDPC 00 0a 00 00 00 02 00 00 // FDPC from: a00 (2560), len: 200 (512)
18 00
FDPC 01 00 01 00 00 00       // FDPC 1 1 0
FDPC 00 0c 00 00 00 02 00 00 // FDPC from: c00 (3072), len: 200 (512)
18 00
SYID 00 00 01 00 00 00       // SYID 0 1 0
SYID 00 0e 00 00 20 00 00 00 // SYID from: e00 (3584), len: 20 (32)
18 00
SGP  00 00 01 00 00 00       // SGP  0 1 0
SGP  20 0e 00 00 0a 00 00 00 // SGP  from: e20 (3616), len: a (10)
18 00
INK  00 00 01 00 00 00       // INK  0 1 0
INK  2a 0e 00 00 04 00 00 00 // INK  from: e2a (3626), len: 4 (4)
18 00
BTEP 00 00 01 00 00 00       // BTEP 0 1 0
PLC  2e 0e 00 00 18 00 00 00 // PLC  from: e2e (3630), len: 18 (24)
18 00
BTEC 00 00 01 00 00 00       // BTEC 0 1 0
PLC  46 0e 00 00 20 00 00 00 // PLC  from: e46 (3654), len: 20 (32)
18 00
FONT 00 00 01 00 00 00       // FONT 0 1 0
FONT 66 0e 00 00 48 03 00 00 // FONT from: e66 (3686), len: 348 (840)
18 00
TCD  03 00 01 00 00 00       // TCD  3 1 0
PLC  ae 11 00 00 24 00 00 00 // PLC  from: 11ae (4526), len: 24 (36)
18 00
TOKN 04 00 01 00 00 00       // TOKN 4 1 0
PLC  d2 11 00 00 0a 01 00 00 // PLC  from: 11d2 (4562), len: 10a (266)
18 00
TOKN 05 00 01 00 00 00       // TOKN 5 1 0
PLC  dc 12 00 00 2a 01 00 00 // PLC  from: 12dc (4828), len: 12a (298)
18 00
STRS 00 00 01 00 00 00       // STRS 0 1 0
PLC  06 14 00 00 46 00 00 00 // PLC  from: 1406 (5126), len: 46 (70)
18 00
MCLD 00 00 01 00 00 00       // MCLD 0 1 0
MCLD 4c 14 00 00 16 06 00 00 // MCLD from: 144c (5196), len: 616 (1558)
18 00
PL   00 00 01 00 00 00       // PL   0 1 0
PL   62 1a 00 00 48 00 00 00 // PL   from: 1a62 (6754), len: 48 (72)
00 00                        // Blank entry follows
00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

(the text will then start)
```

We think that the first 4 bytes of text describes the the function of the data at the offset. The first short is then the count of that type, eg the 2nd will have 1. We think that the second 4 bytes of text describes the format of data block at the offset. The format of the text block is

easy, but we're still trying to figure out the others.

4.1. Structure of TEXT bit

This is very simple. All the text for the document is stored in a single bit of the Quill CONTENTS. The text is stored as little endian 16 bit unicode strings.

4.2. Structure of PLC bit

The first four bytes seem to hold the count of the entries in the bit, and the second four bytes seem to hold the type. There is then some pre-data, and then data for each of the entries, the exact format dependant on the type.

Type 0 has 4 2 byte unsigned ints, then a pair of 2 byte unsigned ints for each entry.

Type 4 has 4 2 byte unsigned ints, then a pair of 4 byte unsigned ints for each entry.

Type 8 has 7 2 byte unsigned ints, then a pair of 4 byte unsigned ints for each entry.

Type 12 holds hyperlinks, and is very much more complex. See `org.apache.poi.hpbf.model.qcbits.QCPLCBit` for our best guess as to how the contents match up.