Introduction to PDF

From GNUpdf

PDF stands for Portable Format Document. This is a format mean to display content identically in all platforms and media (screen, printer...).

Contents

A first example

Let's begin with a very simple "Hello, world!" PDF. You can copy/paste it in a text editor (or download it from File:Hello.pdf), save it as `hello.pdf`, and open it with a PDF reader (https://web.archive.org/web/20140704124557/http://www.pdfreaders.org/) such as xpdf or Evince:


```
%PDF-1.7

1 0 obj  % entry point
<<
  /Type /Catalog
  /Pages 2 0 R
>>
endobj

2 0 obj
<<
  /Type /Pages
  /MediaBox [ 0 0 200 200 ]
  /Count 1
  /Kids [ 3 0 R ]
>>
endobj

3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 4 0 R
    >>
  >>
  /Contents 5 0 R
>>
endobj

4 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Times-Roman
>>
endobj

5 0 obj  % page content
<<
  /Length 44
>>
stream
```

```
BT
70 50 TD
/F1 12 Tf
(Hello, world!) Tj
ET
endstream
endobj

xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
492
%%EOF
```

This PDF was written manually and is voluntarily simplied for the purpose of this introduction. Production PDF are usually more complex.

Exercise 1: produce a "Hello, world!" document with OpenOffice.org and export it as PDF. Compare.

Note: to inspect a PDF with the `less` command-line tool, you may need to add the `-L` option (which disable input preprocessors that run `pdftotext` transparently and hence mask the raw file content).

General structure

As we can see the document has a general text-based structure, which will be useful to play with it. However, PDF files usually contain non-ASCII ("binary") data and should always be considered binary files.

A simple PDF contains 4 parts:

- the header, with the PDF version (and an option line to specify if the PDF contains binary data)

```
%PDF-1.7
```

- the body, containing a series of objects that are used in the document

```
1 0 obj
...
endobj
2 0 obj
...
```

```
endobj
...
```

- a cross-reference table, that specifies the position of the objects

```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
```

- a trailer, with information about where the document starts

```
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
492
%%EOF
```

In the complete document structure, one can append additional body+cross-reference+trailer elements to complete an existing document, but we won't see this case in this first example.

Object syntax

In the body (the object list), we see different kind of definitions:

- Indirect Object (`1 0 obj ... endojb`): define a numbered top-level object. The first number (`1`) is the object number, the second number (`0`) is the revision number, which we don't use in this example.

There are 9 types of objects:

- Number: e.g. `3`
- Indirect reference (`n r R`): references an object, e.g. `5 0 R`. If the objects doesn't exist this is equivalent to the Null object (see below).
- Name (`/Name`): names are identifiers. If you know Lisp or Scheme, this is similar to the `quote` special form (e.g. `'ok`). The initial `/` introduces the name but isn't part of the name; this is similar to `$` in Bash, Perl or PHP.
- Dictionary (`<< ... >>`): this is a unordered list of (Name,Object) pairs. They are essentially hash tables. The Object part can be another Name (e.g. `/Type /Font`).
- Array (`[ x y z ... ]`): an ordered list of objects, e.g. `[ 0 0 200 200 ]`.

- String Object (`(text)`): text. The complete syntax is complex, but for now suffice to say it's text between parenthesis, e.g. `(Hello, world!)`.
- Stream (<< /Length ... >> stream ... endstream): embedded data, can be compressed. It starts with a dictionary that describes the stream such as its length or the encoding (`/Filter`) is uses.

And not used in this example:

- Boolean: `true` or `false`.
- Null Object: `null`.

Representing and manipulating these objects forms the Object layer of the GNUpdf library.

The cross-reference table

```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
```

This is a sequential list of objects (#2, #3, #4...), more exactly the object offsets (the position in bytes from the beginning of the file). The cross-reference table allow to access any given object by its number easily, and fast. For example, this contrasts with HTML, which is purely sequential and doesn't handle large documents so well.

The first 2 numbers mean "I'll introduce 6 objects offsets, counting from 0".

Each line contains the offset of the object definition, a revision number (not used here), and an on/off marker `f` (free) or `n` (in use).

We can ignore the first offset for now.

For example, here:

- Object #1 is at offset 10
- Object #2 is at offset 79
- ...
- Object #5 is at offset 380

If you modify this test document, remember to update all these offsets, as well as the `startxref` line, which describes the offset of the `xref` section.

The cross-reference table can also include more complex declarations, which

we will cover later.

Comments

```
% page content
```

Comments starts with the percent (`%`) character and end at the next newline. Technically, comments are equivalent to a whitespace character, and do not include the ending newline.

How to read the file

A PDF reader won't analyse the document sequentially (from top to bottom), but will access the file in a more complex manner:

- First it reads the first line to get the PDF **version**
- It will then go to the *end* of the document, check the `%%EOF` marker, then get one line above and read `492`, which is the **offset** of the cross-reference table (the previous part). You now understand why you usually can't read a big PDF that you didn't finish downloading yet. When the trailer is lacking or corrupt, some readers attempt to rebuild the index by scanning the whole file, but this is much slower.
- It jumps to the cross-reference table and builds a list of **object offsets**, as described above.
- After the cross-reference table, it can read the trailer dictionary, which namely contains the **Catalog**, which is the start of the document. It's specified by an *indirect reference* to object 1: `1 0 R`.

So far, the reader made random (non-sequential) access to the PDF files, and read the global structure of the document.

- Now the reader checks the **Catalog** object. In this case it only contains a refence to a **Pages** object, number 2.
- The Pages object is a **tree**-like data structure. It's a node that can reference either leaves (pages) or other nodes (which themselves can references leaves and nodes). In this case the Pages object references only 1 page (`/Count 1`), specified in the `Kids` list that contains an indirect reference to object 3: `3 0 R`. The Pages object also defines the size of the medium, which is inherited by the Page (leaf) objects and can be redefined. Here we're defining a small 200x200 box.
- The **Page** object references its parent (`/Parent 2 0 R`), a set of resources necessary to render the page (here, a font, object 4), and its actual content (object 5).
- Object 4 is a **font definition**. Here's we're using one of the 14 base fonts

(`Times-Roman`) for simplicity.

- Object 5 is a **stream** object that contains instructions to render the page. These instructions are very different from the rest of the document and can be considered a different language altogether. The instructions describe 3 directives, between `BT` and `ET`. They use postfixed operators, which is reminiscant of PostScript, PDF's predecessor.
  - position - `70 50 TD`: apply the `TD` operator, which place the text cursor on the page, with argument `70` and `50`, which means the (x,y) coordinates. By default the cursor is at (0,0) which is the *bottom-left* corner of the page. So (70,50) means "up by 70 units" and "right by 50 units". This is different that 2D graphics programming, where the y axis is usually reversed. Here's we have a classic maths/geometry coordinate system.
  - font - `/F1 12 Tf`: apply the `Tf` operator, which sets the font name and font size. The font is `F1` as defined in the resources, with size `12`. This is mandatory: there's no default font.
  - text display - `(Hello, world!) Tj`: apply the `Tj` operator, which shows text, on the text string `Hello, world!`.

We're done!

Filters

The stream was here represented directly, as clear text. In fact, this is uncommon: most streams are compressed. The content of the stream (between `stream` and `endstream`) is then binary data.

The goal of this section is to replace the clear-text stream from our PDF to a compressed stream (you can check the result at File:Hello-stream.pdf).

To encode or decode a stream, one can use the `pdf-filter` utility from GNUpdf.

Let's experiment with the FlateDecode filter:

```
# Encode
$ ./pdf-filter --flateenc <<EOF > filtered.bin
BT
70 50 TD
/F1 12 Tf
(Hello, world!) Tj
ET
EOF

# Check size:
$ ls -l filtered.bin
-rw-r--r-- 1 me me 52 Jan 26 23:59 filtered.bin
# It's binary:
$ file filtered.bin
filtered.bin: data

# Decode it back:
./pdf-filter --flatedec < filtered.bin
```

```
BT
70 50 TD
/F1 12 Tf
(Hello, world!) Tj
ET
```

Note: if you copy/paste a stream from an existing PDF for decoding, make sure that there's no encoding issues. In particular, verify that your extracted stream file (`filtered.bin` above) has the same size as specified in `/Length`. If you use Emacs, try using the `raw-text` encoding (with `M-x set-buffer-file-coding-system` or `C-x RET f`) before copy/pasting between buffers.

Here's a new version of object #5 (the "Hello, world!" text display):

```
5 0 obj  % page content
<<
  /Length 52
  /Filter /FlateDecode
>>
stream
xs
á27P05P€qáÒw3T04R€IãÒðHÍÉÉ×Q(Ï/ÊIQÔT€Éâr
á€á·
Á
endstream
endobj
```

- Note: since the stream is very short, the compressed version is actually longer than the uncompressed version!
- Note: Don't attempt to copy/paste it from this page, the binary data won't support this

Since we're introducing non-ASCII content in the file, the PDF format mandates that we add at least 4 commented binary characters (ASCII value >= 128), so that generic tools have a chance to detect it's a binary file:

```
%PDF-1.7
%éééé
```

The best would be to use characters in `[128,159]`, which aren't part of ASCII nor Latin-1. This won't necessarily work though, since programs such as diff(1) and mercurial (https://web.archive.org/web/20140704124557/http://www.selenic.com/mercurial/wiki/index.cgi/BinaryFiles) consider a file as binary only if it contains a NUL (0) character.

We also need to update the offsets in the reference table and `startxref`:

```
xref
0 6
0000000000 65535 f
0000000016 00000 n
0000000085 00000 n
```

```
0000000179 00000 n
0000000307 00000 n
0000000386 00000 n
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
530
%%EOF
```

Note: to quickly determine the offset of a position in the file under Emacs, you can make a selection from the beginning of the document to that position, use `M-=` (`count-lines-region`) and note the "characters" stat:

```
Region has 52 lines, 530 characters
```

More filters are available:

- Category:PDF_Filters: list of filters
- Lib:Architecture/Base_Layer/Stream_Module: stream handling is provided by the Base layer of libgnupdf

If you want to study an existing PDF, the pdftk (https://web.archive.org/web/20140704124557/http://www.accesspdf.com/pdftk/) tool can come in handy: the `uncompress` command will convert all compressed streams to clear text:

```
pdftk hello-stream.pdf output hello-clear.pdf uncompress
```

References

PDF was initially a proprietary format from Adobe, but since 2008 it's the ISO-32000 standard (more precisely ISO 32000-1:2008).

You can get an electronic copy from ISO for 380 swiss francs (around 250 euros). However, Adobe publishes a copy, which is not official, but has the same technical content and page numbering.

- Adobe - PDF Developer Center: PDF Reference (https://web.archive.org/web/20140704124557/http://www.adobe.com/devnet/pdf/pdf_reference.html)

You'll want to download "Document management – Portable document format – Part 1: PDF 1.7, First Edition (July, 2008)".

Please note that ISO-32000 is non-free documentation and is not to be

reproduced on this wiki. The PDF Knowledge section at gnupdf.org, which this page is part of, aims at offering free documentation on the PDF format that you can read, modify, share and redistribute.

You'll also see documentation about extensions: those are PDF features that are not part of ISO-32000, but that may be proposed in the next revision of the format, but not necessarily. Keeping PDF an open standard is an ongoing battle.

TODO

- Make the "How to read the file" section clearer. Currently it's a bit hard to follow.
- A picture with a basic tree structure (nodes & leaves) would well illustrate the difference between `/Pages` and `/Page`.
- Clarify where PDF/A and PDF/X fit (cf. Goals_and_Motivations)