Hand-coded PDF tutorial

Helpful sections from the specification

The PDF specification ("ISO approved copy of the ISO 32000-1 Standards document") is authoritative and quite readable, so don't be intimidated by it. Here are a few particularly helpful sections to check when trying to write a PDF file:

The "Syntax" section, in particular the "Objects" and "Document Structure" subsections

The Objects subsection explains the available objects in PDF files (e.g., `dictionaries`, `arrays`, `strings`). The Document Structure subsection explains the dictionary entries in `Catalog`, `Pages`, and `Page` dictionaries.

The "Type 1 fonts" sub-subsection of the "Simple Fonts" subsection within the "Text" section

Explains the dictionary entries in Type 1 `Font` dictionaries. TrueType font dictionaries have mostly the same entries.

The two annexes containing the PDF operators

At least in version 1.7 of the specification, the first two annexes contain all the operators in the PDF language. These are the stack-language commands that position text, draw graphics, perform arithmetic, and so on.

The "Example PDF Files" annex

Includes a relatively simple textual PDF file (here's an even simpler one), as well as a simple graphical PDF file.

The specification also discusses PDF's more advanced features, like color profiles, embedded 3D drawings, and active content. ~~On the topic of color profiles, **avoid viewing Annex L "Colour Palettes"; it takes forever to render**, at least using libpoppler in Ubuntu.~~ (Edit: recent versions of libpoppler are better, even if still a little slow compared to Adobe Reader.)

Decompressing PDF files with ps2pdf

The specification suggests using Acrobat to create decompressed versions of existing PDF files as a way to learn about PDF file structure. Conveniently, a free software alternative to the Distiller component of Acrobat is available: ps2pdf (part of Ghostscript). To decompress a PDF file with ps2pdf, run a command like…

```
ps2pdf -dCompressPages=false in.pdf out.pdf
```

Note that bitmap graphics in the output file will still be compressed, but all the text and vector graphics commands will be decompressed.

PDF uses a stack-based command language

PDF's ancestor PostScript is a full stack-based programming language. Commands are run in PostScript by pushing arguments on the stack (for example `1 2`) and then invoking operators (for example `add`). Although PDF does not have all of the capabilities of PostScript, it too works by pushing arguments on the stack and invoking operators.

So in the lines…

```
0 0 Td
(Hello hello.) Tj
```

the PDF interpreter first pushes 0 and 0 on the stack. It then calls `Td`. `Td` pops the top two items from the stack and uses them to set the text display position. In the second line, the interpreter pushes the string “Hello hello.” and calls `Tj` to pop the string and write it on the page.

PDF files contain 4 parts

1. the header
2. the document body, containing at least 3 indirect objects
3. the `xref` cross-reference table
4. the `trailer` (including `startxref` and `%%EOF`).

The header

```
%PDF-1.1
%¥±ë
```

Take for example a minimal PDF file. It is specified as a version 1.1 PDF file because it is limited to features available in 1.1. The second line comment contains 6 high bit characters (displayed as 3 characters in UTF-8 encoding), as required by the “File Header” subsection of the specification (Section 7.5.2):

> If a PDF file contains binary data, as most do (see 7.2, "Lexical Conventions"), the header line shall be immediately followed by a comment line containing at least four binary characters—that is, characters whose codes are 128 or greater. This ensures proper behaviour of file transfer applications that inspect data near the beginning of a file to determine whether to treat the file’s contents as text or as binary.

The body

As seen in the minimal PDF file, only 4 “indirect” objects are required to display a single page of text: an indirect `Catalog` dictionary, an indirect `Pages` dictionary, an indirect `Page` dictionary, and an indirect `stream`. Indirect objects are accessed by reference. A reference looks like `2 0 R`. 2 is the object number, 0 is the “generation” number, and `R` is the operator. In contrast, direct objects appear directly inline. Any datum in the body that is not a reference to an indirect object is a direct object.

- << >> double angle-bracketed expressions are direct `dictionary` objects
- [ ] bracketed expressions are direct `array` objects
- / forward slash-preceded words are direct `name` objects
- ( ) parenthesized expressions are direct `string` objects
- numbers are direct objects too

For the most part, direct objects and indirect objects are interchangeable, but not always. None of the 4 indirect objects in the minimal file, nor the direct `Length` dictionary may be replaced. Converting a direct object to an indirect object allows easy reuse of that object anywhere in the PDF file.

Why can't the 4 indirect objects in the minimal PDF file be direct objects? For the `stream` object, the reason is that the specification states that “all streams shall be indirect objects.” For the others, the reason is that they are used as values in special dictionaries that contain a `Type` key. Dictionaries with a defined `Type` have rules about what other key–value pairs they may contain. One possible rule is that some of the values in the dictionary must be indirect objects. Each of the 3 indirect dictionary objects in the minimal file is at some point used as a value subject to this rule. For example the specification says that the value paired with the `Pages` key in the `Catalog` dictionary “shall be an indirect reference” :

Key	Type	Value
Pages	dictionary	*(Required; shall be an indirect reference)* The page tree *node* that shall be the root of the documents *page tree* (see 7.7.3, "Page Tree").

Note that PDF “objects” are just data; they are not OOP “objects.” For more details on objects and special dictionaries, see the specification subsections “Objects” and “Document Structure” within the “Syntax” chapter.

The cross-reference table

Line

```
xref
0 5
0000000000 65535 f
0000000018 00000 n
0000000077 00000 n
0000000178 00000 n
0000000457 00000 n
```

The cross-reference table tells the PDF viewer where to find the indirect objects in the document. The first line indicates the number of the first indirect object in the table, which in this case is the special object numbered 0 (the second example xref table below might help clarify this). It also gives the total number of objects in the table (five). The next line is for object 0, the head of the linked list of free objects (see the specification for more information). The third line describes object 1: the byte offset of the object relative to the beginning of the file, the generation number, the letter “n,” and a two-byte end-of-line (for example `space + linefeed`). Each successive line is implicitly counted as the next object, so the fourth line in this example is object 2. The letters indicate whether an object is free (f) or in-use (n). In a brand-new PDF file, for objects other than 0, the generation number will be 0 and the letter will be “n.” Using PDF's built-in mechanism to non-destructively overwrite information will increase the generation numbers of outdated objects and change their letters to “f.”

The same objects, this time divided into two subsections

Line

```
xref
0 2
0000000000 65535 f
0000000018 00000 n
2 3
0000000077 00000 n
```

```
0000000178 00000 n
0000000457 00000 n
```

This example shows that an xref table may contain multiple subsections, each started by a header line that consists of the starting object number and the total number of objects in the subsection. This particular table has two subsections: one starting at object 0 and one starting at object 2. The first subsection contains 2 objects, and the second contains 3.

Again, consult the specification for more details:

- The “Cross-Reference Table” section under the “File Structure” section of the “Syntax” chapter
- The “Updating Example” from the “Example PDF Files” annex

The trailer

```
trailer
<<  /Root 1 0 R
    /Size 5
>>
startxref
565
%%EOF
```

The trailer dictionary must contain at least the **`Root`** and **`Size`** entries. The **`Root`** must be an indirect **`Catalog`** dictionary. This **`Catalog`** tells the PDF viewer where to find the various **`Pages`** objects that make up the document's displayable content. The **`Size`** must *not* be indirect, and is just a count of the number of indirect objects in the file. The **`startxref`** tells the PDF viewer the byte offset of the most recent **`xref`**.

Line endings

In general any of the following may be used:

- `0x0a` “line feed”
- `0x0d` “carriage return” (not allowed as the EOL directly after the **`stream`** keyword)
- `0x0d0a` “CR + LF”

(See “Newline” on Wikipedia and the “Character Set” subsection of the spec.)

In the xref table one of the following must be used:

- `0x200a` “space + LF”
- `0x200d` “space + CR”
- `0x0d0a` “CR + LF”

Counting bytes correctly

0-indexed byte counts are required for the **`xref`** table, the **`startxref`**, and the **`Length`** of streams. Which bytes are included in the count and where do the counts start?

Indirect objects

For indirect objects, the correct byte to use is the first byte of the **`obj`** line, which is often the object number.

Example:

```
1 0 obj
```

Character:	1		0		o	b	j	\n
Byte offset:	18	19	20	21	22	23	24	25

Indirect object number 1 is located at the 18th byte of the file.

Stream **`Length`**

The correct count (from the spec.):

> The number of bytes from the beginning of the line following the keyword **`stream`** to the last byte just before the keyword **`endstream`**. (There may be an additional EOL marker, preceding **`endstream`**, that is not included in the count and is not logically part of the stream data.)

Also:

> There should be an end-of-line marker after the data and before **`endstream`**; this marker shall not be included in the stream length.

Example (assuming the EOL is `0x0a`):

	Cumulative byte count
`stream`	0
`BT`	3
`/F1 18 Tf`	13
`0 0 Td`	20
	21
	22
	23
`(Hello hello.) Tj`	41
`ET`	44
	45
	46
	47
	47
`endstream`	47

The correct count is 47. Notice in particular that the last end-of-line before **`endstream`** is not counted.

To make things a little easier, one can calculate the length by subtracting the byte offset of the first character on the line after **`stream`** (the "B" of "BT" in this example) from the byte offset of the last character before the end-of-line that precedes `endstream`, and then adding 1.

How to find byte offsets

The lazy way

Leave out the `xref` section, estimate stream lengths, and use pdftk to fix the PDF file. Install `pdftk` and do

```
pdftk foo.pdf output fixedfoo.pdf
```

It might not be able to handle every case.

Use an editor of some kind

- In Vim, open the file with `vim -b`, and do something like…

  ```
  :set rulerformat=%30(%4l,%4c,%4o%12P%)
  ```

  …to put the current line, character, byte offset, and file position in the ruler. Vim calls the first byte 1, so subtract 1 from the counts to get correct numbers. For some reason, when I used Vim on certain pre-made PDF files it didn't include every newline in the count.

- Pipe the file through…

  ```
  hexdump -v -e '/1 "%_p"'
  ```

  (thanks to this manual; also see a few more hexdump examples). This will print the whole file on one line with each byte represented by one *printing* character so character counts can be used as byte counts.

Links

http://blog.idrsolutions.com/?s=%22Make+your+own+PDF+file%22
A series of posts that explains how to write PDF files from scratch.

Portable Document Format: An Introduction for Programmers
Another short introduction to writing PDF files.

Found a mistake?

Submit a comment or correction

Comments

Jim Nickerson, 16 Aug 2015 19:56:52 -0400

> The Link Portable Document Format: An Introduction for Programmers
> http://www.mactech.com/articles/mactech/Vol.15/15.09/PDFIntro/Portable
> has become http://www.mactech.com/articles/mactech/Vol.15/15.09/PDFIntro/index.html

> Kurt Pfeifle (@pdfkungfu), 01 May 2014 21:12:08 -0400
>
> When using VIm to edit or view a PDF file, always use "vim -b file.pdf".
>
> This "-b" opens the file in binary mode. Only in binary mode VIm's byte counting for binary (and ASCII) files will work correctly.
>
> This way you can easily jump to any offset (like the ones you may read from the xref sections) by simply typing ": goto NNN" (where NNN is the byte offset integer number you want).

Updates

10 Jul 2017	Correct `xref` table object count description from “four” to “five” to account for the 24 Dec 2012 edit.
13 Sep 2015	Correct link to “Portable Document Format: An Introduction for Programmers”
03 May 2014	Mention `vim -b`, thanks to @pdfkungfu's good suggestion!
08 Jan 2013	Comments link
24 Dec 2012	Update to match the corrected minimal PDF file: all streams must be indirect objects.
27 Jan 2012	small wording and formatting changes
26 Jan 2012	corrected the string drawing command to Tj
04 Sep 2011	proofreading, xref table subsections, updated ps2pdf and Ghostscript links
07 Apr 2011	xref table line numbers were misaligned
22 Jan 2011	using ps2pdf to decompress PDF files
02 Dec 2010	a little more about the trailer, some markup change around code snippets
20 Nov 2010	clean up
07 Oct 2010	Change revisions from a `dl` to a `table`
03 Oct 2010	Minor formatting change
13 Sep 2010	First-ish version