PDF internals

callas

Table of Contents

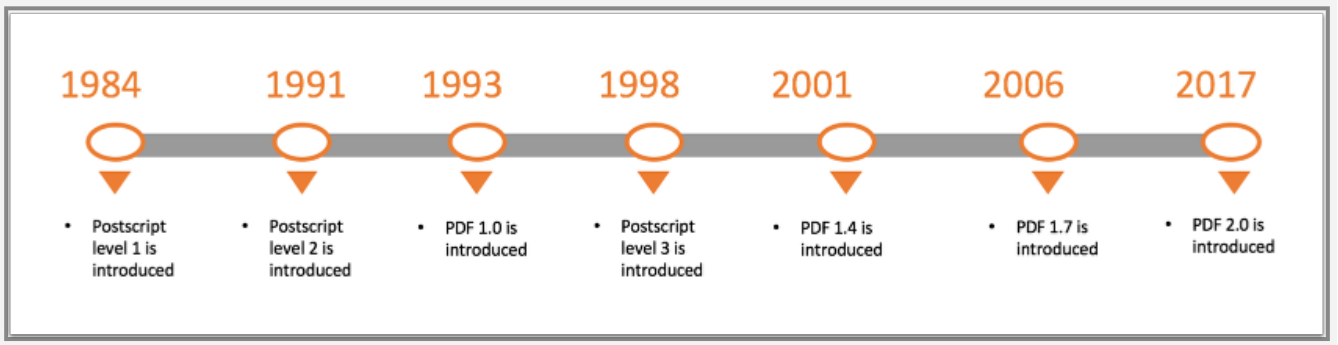
1. Creating a simple PDF file

1.1 How to create a simple PDF file

Brief history of PDF

The beginning - John Warnock, an engineer at Xerox, developed a language called 'Interpress' that could be used to control Xerox laser printers. He along with his boss, Charles M. Geschke, tried for two years to convince Xerox to turn Interpress into a commercial product. When this failed, they decided to leave Xerox and try it on their own - by founding Adobe.

PDF started off as an internal project at Adobe by John Warnock to create a file format so that documents could be spread throughout the company and displayed on any computer using any operating system. The engineers at Adobe enhanced two technologies: Postscript and Adobe Illustrator and created both a new file format (PDF, which is really a kind of optimized PostScript) and a set of applications to create and visualize these files.



The internal structure of PDF

PDF files use a fixed structure and always contain 4 sections:

- A header, which contains information on the PDF-specifications the file adheres to. This line looks like this:

```
%PDF-1.7
```

- The body area which contains a description of the various elements that are placed on the pages.

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

- A cross-reference table which refers to all the elements from the body that are used on the pages of the PDF-file. In other words, the table is mainly a list of the addresses of each object in the body section.

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

The first number after xref says that this list starts at object 0, the object number of the first object in this subsection. The second number after xref is a count of how many objects (6) are in this table and that the remaining five entries are for objects with object numbers 1, 2, 3, 4 and 5. Here object #1 is at offset 10 and is 'in use' (n).

Please note that the first ten digits (0000000000) of the first entry for object 0 points to the next free object, which is, the first object itself.

- A trailer which tells applications or RIPs where to find the cross-reference table and always ends with '%%EOF'. If this line is missing, the PDF-file is not complete and can probably not be processed by any RIP or application. This is not the case with PostScript files. If the last few lines of a PostScript file are missing (because of a lost connection while transferring the file or a computer crash) you can often still print most of the pages. With a PDF-file, you'll lose everything.

```
trailer
```

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

The end of a PDF file is read first by the PDF reading application. The trailer holds information about the location and details of the Cross-reference table. The trailer has three parts. The first part has the keyword *trailer* followed by a dictionary that holds values for certain fields.

The second part has the keyword *startxref*, and in the next line, a number. The number denotes how far (in bytes) the keyword *xref* (of the last section of the cross-reference table) is from the start of the file. The very next line has the value %%EOF to denote the end of the file.

Start with a simple PDF

```
%PDF-1.4
1 0 obj
<<
        /Length 51
>>
stream
1 0 0 RG
5 w
36 144 m
180 144 l
180 36 l
36 36 l
s
endstream
endobj
2 0 obj
<<
        /Type /Catalog
        /Pages 3 0 R
>>
```

```
endobj
3 0 obj
<<
        /Type /Pages
        /Kids [4 0 R ]
        /Count 1
>>
endobj
4 0 obj
<<
        /Type /Page
        /Parent 3 0 R
        /MediaBox [0 0 612 792]
        /Contents 1 0 R
>>
endobj


xref
0 4
0000000000 65535 f
0000000010 00000 n
0000000113 00000 n
0000000165 00000 n
0000000227 00000 n
trailer
<<
        /Size 4
        /Root 2 0 R
>>
startxref
344
%%EOF
```

Copy this code in a text editor and create your own PDF file.

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

2. Fonts explained

2.1 Introduction to Fonts

Important terms

- Glyph: A ‘glyph’ is the shape of a character in a font
- .notdef glyph: A ‘.notdef’ glyph is a particular character that is used when glyph selection fails
- Empty glyph: An ‘empty glyph’ is a glyph without contour (shape), e.g. a blank

Font basics

Each block of text in a PDF document consists of four sets of data.

- The **encoded characters** which are sequences of bytes that represent the individual character codes that make up the text in the page
- The **font data** which is a group of glyphs (character visualizations) accessed by a unique number called a **Glyph ID (present in the font)**
- A map that links the encoded character codes to Glyph IDs (This mapping can be called encoding or glyph selection or glyph mapping)
- An optional map that links the character codes to Unicode values. This map is not needed when displaying the PDF but is required to allow the user to extract text content from the document (Mapping to Unicode: explained below)

How a page description in PDF makes use of fonts

- Page description (content stream) selects font through an internal name: /F25 12 Tf
- Internal name points to font resource object (for current page):

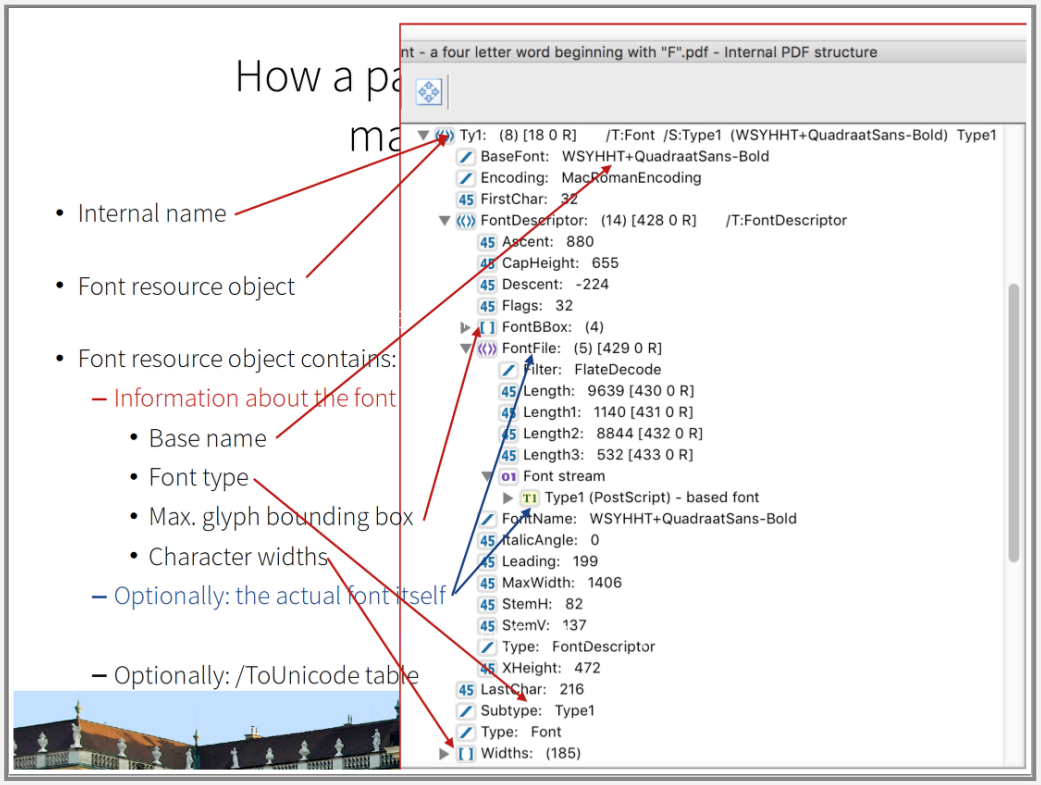
/Resources << /Font << /F25 12 0 R >> >>

- Font resource object contains information about the font, for example:

1. Base name (example: /Courier)

2. Font type (example: /TrueType)
3. Character widths (example: [593 615 572 ...])

- Optionally: the actual font itself (= “font is embedded”)
  - Data stream (for example in /FontFile2 ...)
- Optionally: /ToUnicode table for mapping to Unicode



Mechanisms to include fonts in a PDF

By preference any fonts that are used in a layout are also included in the PDF file itself. This makes sure that the file can be viewed and printed as it was created by the designer. There are two mechanisms to include fonts in a PDF:

- Full embedding – A full copy of the entire character set of a font is stored in the PDF.
- Subsetting – Only those characters that are actually used in the layout are stored in the PDF. If the “$” character doesn’t appear anywhere in the text, that character is not included in the font. This means that PDF files with subsetted fonts are smaller than PDF files with embedded fonts.

Font types

- Simple fonts: 1 byte character codes that can address max. 256 glyphs

1. Type 1: 20 years old hence ancient in technology terms. Type 1 fonts are not cross-platform.
2. Compact Font Format (**CFF** font), very similar to Type 1
3. Type 3: Uses PDF drawing commands to define the glyph outlines. This font format allows greater flexibility over the appearance of the glyphs but does not include a **hinting** mechanism resulting in reduced visual quality for small text or low resolutions.
4. TrueType: Outline fonts which means that they can deliver output at any resolution or size.
5. OpenType: OpenType fonts internally resemble either TrueType fonts or Type 1 font data

- Composite fonts: 1 or 2 byte character codes that can address max. 65535 glyphs

1. Type 0: (Type 1 or OpenType/Type 1/CFF based)
2. Type 2: (TrueType or OpenType/TrueType based)

Mapping to Unicode

- Not required for displaying the page, but for text search or text copy.
- Options of mapping to Unicode:

1. Implicitly by standard (glyph selection) encodings in the PDF (example: MacRomanEncoding)
2. Encoding information in the actual font ('ToUnicode" cmap in TrueType fonts)
3. By glyph names in the actual font (in Type 1 fonts) via the Adobe Glyph List (maps glyph names to Unicode)
4. Explicitly by a ToUnicode table in the PDF data structure