

PDF = PestesDateiFormat

Imagine being able to send full text and graphics documents (...) [that] could be viewed on any machine (...). This capability would truly change the way information is managed.



Portable Document Format



Max page size of PDF: 381 km × 381 km

Structure of this talk

PDF introduction:

- PDF history
- PDF standards
- PDF file format

Text rendering:

- PDF text display
- TTF file format

(My) new ideas:

- PDF & TTF writer
- Reflection

PDF History and Standards

PostScript

Somewhat the predecessor of PDF, created by the same company, Adobe. Presented in the 80s.

```
%!  
/Courier findfont      % Select font  
20 scalefont           % Scale to font size 20  
setfont                % Set it as active font  
50 50 moveto           % Set cursor to (50, 50)  
(Hallo Welt!) show     % Print text at cursor position  
  
showpage               % Show page
```

Primarily used for vector graphics, but also turing-complete stack-oriented programming language.

What the printer sees!

PDF

Improvement over PostScript.

No programming language anymore, more rigid structure, more features.

Enables seek (load single page without compiling everything before), Comments, Forms, Video- and Audioplayback, 3D, ...

Version 1.0 released in 1993.

Acrobar Reader: 50\$

Acrobat Distiller (personal version): 695\$

Acrobat Distiller (network version): 2495\$

Releases

	Year	Industry	Notable features
v1.0	1993	Adobe	text, images, pages, hyperlinks , bookmarks
v1.1	1994	Tax	passwords, encryption, device-independent color
v1.2	1996	Printing	radio buttons, checkboxes, forms incl. import/export, mouse events, sound, unicode , color features
v1.3	2000	Printing	digital signatures, color spaces, JavaScript, embedded file streams, image utilities, CIDFonts , prepress support
v1.4	2001		RC4 > 40bits, transparency, better forms, metadata, accessibility, page boundaries, printer marks, predefined CMaps
v1.5	2003		jpeg, multimedia playback, better forms, public key encryption, permissions, view/hide layers, slideshow
v1.6	2004		3D, OpenType , SOAP over http, public key encryption improvements, color spaces
v1.7	2006		3D improvements, public key encryption improvements

Version 1.7 is ISO 32000-1:2008.

Recent developments

PDF 1.7 Extensions:

- PDF 1.7 Extension Level 1 (2008)
- PDF 1.7 Extension Level 3 (2008)
- PDF 1.7 Extension Level 5 (2009)
- PDF 1.7 Extension Level 6 (2009)
- PDF 1.7 Extension Level 8 (2011)

Newest ISO version is ISO 32000-2:2017: PDF 2.0.
Clarified 1.7 specification, some new features.

Even more standards

Specialized use cases = more standards:

- PDF/X for graphic
- PDF/A for archival
- PDF/E for technical documentation
- PDF/VT for dynamic data
- PDF/UA for accessibility

PDF/A Levels:

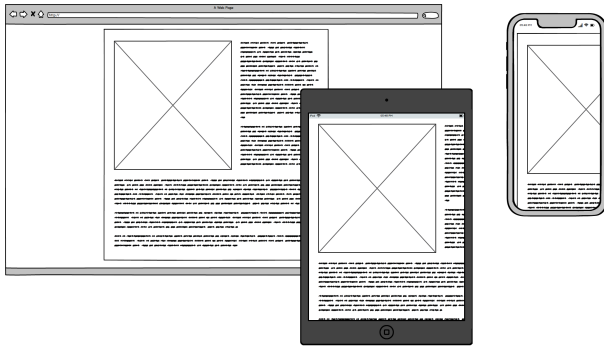
- Level B guarantees visual reproduction.
- Level A additionally guarantees content reproduction.

PDF/A Subversions:

- PDF/A-1 for PDF 1.4 standard
- PDF/A-2 for PDF 1.7 standard
- PDF/A-3 for PDF 2.0 standard

Why is it so popular?

Single purpose, which is archived:
It displays content equally on all devices.



Comparison: How webpages deal with different devices:

- Adapt font size, colors, spacing, ... to screen size
- Adapt layout to aspect ratio / screen size
- Remove or add elements depending on end device
- Test on end devices and/or use resources such as [Can I Use](#)

In short: Its painful and slow.

The file format

Tokens

PDF is a text format. You can open any PDF in your text editor!

Tokens:

```
0 % Numbers
Hello % Strings
5 0 R % References (5 for the object number, 0 for its generation number, R
[2 0 R] % Arrays
<</Key /Value>> % Dictionaries
Image % Names (any two with the same content are "equal")
```

Out of these tokens, the higher-level objects are composed

- Dictionaries
- Streams

```
<</Type /Catalog /Pages 2 0 R>>

stream
BT 1 0 0 1 22 20 cm 1 w /F 12 Tf 14.4 TL (Hello world)Tj ET
endstream
```

Structure (1/2)

Header (asserts PDF version and whether binary data is contained)

```
%PDF-1.7  
%
```

Body (contains the actual content)

```
1 0 obj  
<</Type /Catalog /Pages 2 0 R>>  
endobj  
2 0 obj  
<</Type /Pages /Kids [3 0 R] /Count 1>>  
endobj  
% ...
```

Structure (2/2)

Cross-Reference Table (CRT) (contains the binary offset of objects)

```
xref
0 8
0000000000 65535 f
0000000015 00000 n
0000000062 00000 n
% ...
```

Trailer (contains size of CRT and reader start points)

```
trailer
<</Size 8 /Root 1 0 R /Info 7 0 R>>
startxref
574
%%EOF
```

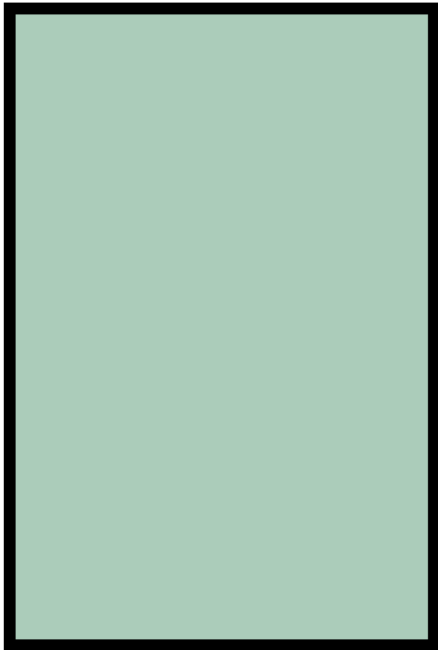
Body

```
1 0 obj
<</Type /Catalog /Pages 2 0 R>>
endobj
2 0 obj
<</Type /Pages /Kids [3 0 R] /Count 1>>
endobj
3 0 obj
<</Type /Page /Parent 2 0 R /MediaBox [0 0 210 297] /Resources 4 0 R /Contents [6 0 R]>>
endobj
4 0 obj
<</Font <</F 5 0 R>> /ProcSet [/PDF /Text]>>
endobj
5 0 obj
<</Type /Font /Subtype /Type1 /BaseFont /Helvetica /Encoding /WinAnsiEncoding>>
endobj
6 0 obj
<</Length 59>>
stream
BT 1 0 0 1 22 20 cm 1 w /F 12 Tf 14.4 TL (Hello world)Tj ET
endstream
endobj
```

Drawing

Drawing (rg = background color, re = rectangle, b = painting mode)

```
stream  
1 0 0 1 40 20 cm 0.5 w 0.67 0.8 0.73 rg 0 0 20 30 re b  
endstream
```



Include Image

Stream with binary image data

```
5 0 obj
<</Length 570 /Type /XObject /Subtype /Image /Width 25 /Height 16 /Filter /
stream
???JFIF???#####'++'6;4;6PJCCJPzW]W]Wz?s?ss?s?????????
??S???o?o????????????????#####'++'6;4;6PJCCJPzW]W]Wz?s?ss?s?????????%????
??S???o?o?????????????????????????????????????????????????????????????°?N?????????
endstream
endobj
```



Print image

Declare image alias (for 4 0 R referenced as Resources in Page):

```
4 0 obj
<</XObject <</I 5 0 R>> /ProcSet [/PDF /ImageC]>>
endobj
5 0 obj
(image stream)
```

Print image (Do = print referenced image):

```
6 0 obj
<</Length 28>>
stream
20 0 0 20 20 20 cm 1 w /I Do
endstream
endobj
```

Enough foreplay: Render text!

Or: What is the most complicated approach to "support" unicode?

Crashcourse encoding

Character: Letters / numbers ("what you see"); like 2, a, A, {, \$

Encoding: Defines how bits are mapped to characters

Character	UTF-8 Encoding	UTF-16 Encoding	(decimal)
2	0010 1000	0000 0000 0010 1000	32
a	0011 1101	0000 0000 0011 1101	61
A	0010 1001	0000 0000 0010 1001	41

Unicode: Maps decimals to characters. *huge*: March 2020, 143,859 characters defined.

Text with default encoding / fonts

Choose predefined font (Helvetica) & predefined encoding (WinAnsiEncoding).

```
4 0 obj
<</Font <</F 5 0 R>> /ProcSet [/PDF /Text]>>
endobj
5 0 obj
<</Type /Font /Subtype /Type1 /BaseFont /Helvetica /Encoding /WinAnsiEncoding>>
endobj
```

Then print text like (w = line width, Tf = font, TL = leading, Tj = text)

```
6 0 obj
<</Length 59>>
stream
BT 1 0 0 1 22 20 cm 1 w /F 12 Tf 14.4 TL (Hello world)Tj ET
endstream
```

How about unicode?

Default encodings are all single byte encoding. What if I need a multi-byte / non-european character?

Embedd .ttf file:

```
8 0 obj
<</Type /FontDescriptor /FontName /OpenSansRegular /Flags 6
/FontBBox [96 0 561 953] /ItalicAngle 0 /Ascent 1094 /Descent -300
/CapHeight 0 /StemV 0 /FontFile3 9 0 R>>
endobj
9 0 obj
<</Length 4274 /Subtype /OpenType>>
stream
(...)
endstream
endobj
```

Still have to define encoding...

Roll your own encoding

Define text encoding:

```
7 begincodespacerange  
<6c> <6c>  
<61> <68>  
endcodespacerange
```

Define how text maps to glyph indexes:

```
7 begincidrange  
<6c> <6c> 2  
<61> <67> 4  
<68> <68> 11  
endcidrange
```

Both these steps require deep knowledge about fonts!

The TTF format

Introduction

History:

- Adobe Type-1 (1984)
- Apple TrueType (1991)
- Microsoft & Adobe OpenType (1996)

OpenType flavours:

- TrueType-flavoured OpenType (.ttf)
- PostScript-flavoured OpenType (.otf)

Omnipresent.

Table directory

File starts like this:

```
0x00010000 || 12 || (...) || tables
```

0x00010000 for TTF (or 0x4F54544F for OTF)

12 is number of tables

tables specify content

Many tables, many functions.

Have you tried typesetting مرحبا بالعالم?

What about sufficient?

Name	Purpose
cmap	character maps
maxp	memory requirements (#characters, #points, ...)
head	global information (max bounding box, font style, ...)
OS/2	windows font metrics
name	multilingual strings about font (copyright notice, font name, ...)
cvt	list of values for instructions
fpgm	program run upon first usage of font
gasp	"preferred rasterization on gray-capable devices"
prep	program run upon character drawing
GDEF	ligatures
GPOS	"glyph placement in sophisticated text layout"
GSUB	ligatures
hhea	sizing per glyph (ascender, decender, ...)
loca	offsets of glyph data blocks
hmtx	width of glyphs
glyf	list of glyph data blocks

Overview ttf processing

Binary read out:

- Read out table directory to get offset / type of tables
- Read out each table (possibly multiple formats; cmap has 4!)

Interpret:

- Split glyph table as mandated by loca
- Get character widths as defined by hhea and maxp
- Use post and cmap to get unicode per glyph

Write back:

- Create subset with glyphs we actually need
- Correct "summary" tables (like #characters in font)
- Recreate binary file

In short: Read out tables, patch them together differently, pray everything works.

Putting it all together

TFF preprocessing:

- parse TFF
- transform glyphs & widths into suitable structure

TFF subset generation:

- wait until PDF fully specified
- collect all text within PDF file
- use set of chars to create TTF subset
- save TTF subset, remembering the order of the glyphs

PDF text writing:

- define encoding (pro-tip: use UTF-8)
- reencode all text within PDF to this encoding
- generate PostScript CMap knowing encoding / TFF glyph order
- generate reverse PostScript CMap (why?)

Bonus: How to extend ttf?

Steps:

1. Just do it
2. Wait until other vendors need to adapt
3. Profit

like colored glyphs (emojis!):

- COLR, CPAL (microsoft + mozilla)
- CBDT, CBLC (google)
- sbix (apple)

PDF Writers

State of the art

Overview:

- PHP provides [TCPDF](#) and [FPDF](#) (and "new" [tc-lib-pdf](#)).
- Python has [PyFPDF](#), [PyPDF4](#).
- Some action in the [go](#) and [JavaScript](#) universe.

Libraries are hard to use.

```
private function printH3($text, $startX, $startY)
{
    $this->pdfDocument->SetXY($startX, $startY);
    $this->pdfDocument->SetFontSize(22);
    $this->pdfDocument->SetFont('opensans', 'b');
    $this->pdfDocument->MultiCell(0, 0, $text, 0, 'L', false, 1);
    $this->pdfDocument->Ln(1.6);
}
```

Usually html & CSS to PDF "converters" are used -> spin up chrome headless, and use it to print the page.

Architecture (TCPDF)

Two big tasks:

- Read (and possibly write) .ttf
- Write .pdf

Issues:

- "One File approach": Leads to 24k (pdf) and 3K (ttf) lines of spaget.
- Code quality generally **very** low. [1](#) [2](#) [3](#)
- API unintuitive, badly documented (AddPage vs startPage vs addTOCPage?)
- Resulting code is untestable, hard to follow, large [print table row](#)

You cannot meaningfully improve upon this existing work.

Target

Implementation requirements:

- Add content (text, images, drawings)
- Within Layout (columns, tables, grid, ...)
- Fully testable (what text is printed? which color does it have?)
- No output technology specific details

```
// create printer
$printer = new Printer(new PdfPrinter());
$printer->setColor("dark grey");

// create layout
$layout = new ColumnLayout($printer);
$layout->setColumns(2);
```

Architecture:

- document-generator provides printer & layouts (nothing PDF-specific!)
- pdf-generator implements interfaces required by document-generator
- html-generator, email-generator, word-generator, ...

document-generator

Concept:

- abstracts PDF specific details (can also attach HTML / CSS export)
- Layout (recursively!) defines columns, tables, grid, ...
- Printer allows to print text, images within said layout

```
/**
 * @param string[] $rowContent
 */
private function printTableRow(TableRowLayoutInterface $row, PrintFactoryIn
{
    $printer = $printFactory->getPrinter($row);

    $columnLength = \count($rowContent);
    for ($i = 0; $i < $columnLength; ++$i) {
        $row->setColumn($i);
        $printer->printParagraph($rowContent[$i]);
    }
}
```

pdf-generator

Concept:

- abstract lower-level details continuously to tame complexity
- Printer prints text, images & drawings

Architecture:

- Frontend (to attach to document-generator)
- IR (pdf-agnostic Printer with colors, sizing, fonts, ...)
- Backend (actually writes the PDF; 5 more layers of abstraction)

```
$printer = new Printer(new Document());  
$printer->setCursor(new Cursor($xPosition, $yPosition, 1));  
$printer->printText('Hello World');  
$result = $printer->save();
```

... same architecture for ttf!

Reflection

Reflection

Success factors:

- PDF specification was adapted to customer requests
- TTF can be extended by anyone without breaking readers

What about long term storage (PDF)?

- PDF specification is complex, large, organically grown
- PDF destroys semantic meaning of source
- Might not even be able to recover text!

Better Long-Term storage?

- Word / Excel -> nearly same problems as PDF
- HTML / CSS as structure stays -> but writing a browser even harder