Filename extension	`.pdf` [note 1]
Internet media type	`application/pdf` ,[1] `application/x-pdf` `application/x-bzpdf` `application/x-gzpdf`
Type code	'PDF '[1] (including a single space)
Uniform Type Identifier (UTI)	com.adobe.pdf
Magic number	`%PDF`
Developed by	ISO Originally Adobe
Initial release	15 June 1993; 26 years ago
Latest release	2.0
Extended to	PDF/A, PDF/E, PDF/UA, PDF/VT, PDF/X
Standard	ISO 32000–2
Open format?	Yes
Website	www.iso.org/standard/63534.html

Inside PDF

zhuyie
zhuyie@gmail.com

Agenda

- Introduction
- File Structure
- Objects
- Document Structure
- Text Basics
- Glyph Selection
- Unicode Mapping

Introduction

- The Portable Document Format (**PDF**) is a file format developed by **Adobe** in the 1990s to present documents, including text formatting and images, in a manner **independent** of application software, hardware, and operating systems.
- Based on the **PostScript** language.
- In the early years PDF was popular mainly in **desktop publishing** workflows, and competed with a variety of formats such as **DjVu**, Envoy, Common Ground Digital Paper, Farallon Replica.

PostScript

- Researchers at Xerox PARC had developed the first **laser printer** and had recognized the **need** for a standard means of **defining page images**.
- John Warnock and Chuck Geschke founded **Adobe Systems** in 1982. They created **PostScript** in 1984. At about this time they were visited by **Steve Jobs**, who urged them to adapt PostScript to be used as the language for driving laser printers.
- In March 1985, the **Apple LaserWriter** was the first printer to ship with PostScript, sparking the desktop publishing (DTP) revolution in the mid-1980s.

"Hello world!" in PostScript

```
%!PS
/Courier                    % name the desired font
20 selectfont               % choose the size in points and establish
                            % the font as the current one
72 500 moveto               % position the current point at
                            % coordinates 72, 500 (the origin is at the
                            % lower-left corner of the page)
(Hello world!) show         % stroke the text in parentheses
showpage                    % print all on the page
```

DjVu

- DjVu (/ˌdeɪʒɑːˈvuː/) is a computer file format designed primarily to store **scanned documents**, especially those containing a combination of text, line drawings, indexed color images, and photographs.
- DjVu has been promoted as providing **smaller** files than PDF for most scanned documents.
- The DjVu technology was originally developed by **Yann LeCun**, Léon Bottou, Patrick Haffner, and Paul G. Howard at AT&T Labs from 1996 to 2001.

File Structure

- A PDF file is a 7-bit **ASCII** file, except for certain elements that may have **binary content**.
- A PDF contains 4 sections:
  - **Header**, defines the version of PDF specification.
  - **Body**, the actual content that will be displayed.
  - **Cross-reference table**, a table for PDF viewers to quickly access different objects.
  - **Trailer**, defines other meta info of a PDF file.

A blank PDF

```
1   %PDF-1.7                                    Header
2   1 0 obj
3     << /Type /Catalog
4        /Pages 2 0 R
5     >>
6   endobj
7   2 0 obj
8     << /Type /Pages
9        /Kids [3 0 R]
10       /Count 1                               Body
11    >>
12  endobj
13  3 0 obj
14    << /Type /Page
15       /Parent 2 0 R
16       /MediaBox [0 0 600 400]
17       /Resources << >>
18    >>
19  endobj
```

```
20  4 0 obj
21    << /ModDate (D:20200505153537+08'00')
22    >>
23  endobj
24  xref
25  0 5
26  00000000000 65535 f
27  00000000010 00000 n        xref
28  00000000068 00000 n
29  00000000139 00000 n
30  00000000246 00000 n
31  trailer
32    << /ID [<55666dff9ee999338eb5f5e5e11fa18e>
33       /Root 1 0 R
34       /Info 4 0 R
35       /Size 5
                                trailer
36    >>
37  startxref
38  307
39  %%EOF
```

Objects

- Boolean values
- Integer and real numbers
- Strings
- Names
- Arrays
- Dictionaries
- Streams
- Null
- Indirect Objects

Simple Objects

- Boolean: `true false`
- Numbers
  - Integer: `123 43445 +17 -98 0`
  - Real: `34.5 -3.62 +123.6 4. -.002 0.0`
- Strings
  - As a sequence of literal characters enclosed in parentheses ( )
    - `(This is a string)`
  - As hexadecimal data enclosed in angle brackets < >
    - `<4E6F762073686D6F7A206B6120706F702E>`

Simple Objects (cont.)

- Name
  - A name object is an atomic symbol uniquely defined by a sequence of characters.
  - A slash character (/) introduces a name.
  - `/ASomewhatLongerName`
- Null
  - The null object has a type and value that are unequal to those of any other object. There is only one object of type null, denoted by the keyword `null`.

Array Objects

- One-dimensional collection of objects arranged sequentially.
- Heterogeneous.
- Enclosed in square brackets ([and]).
- `[549 3.14 false (Ralph) /SomeName]`

Dictionary Objects

- A sequence of key-value pairs enclosed in double angle brackets (<< … >>).
- The key must be a name, the value can be any kind of object, including another dictionary.

```
<< /Type /Example /IntegerItem 12 /StringItem (a string)
/Subdictionary << /Item1 true >> >>
```

Stream Objects

- A stream consists of a **dictionary** that describes a sequence of bytes, **followed by zero or more lines of bytes** bracketed between the keywords **stream** and **endstream**.

dictionary
stream
…Zero or more lines of bytes…
endstream

- Can read incrementally, can be of unlimited length. Objects with potentially large amounts of data, such as images and page descriptions, are represented as streams.

Stream Objects (cont.)

- Every stream dictionary has a **Length** entry that indicates how many bytes of the PDF file are used for the stream's data.
- Stream's data can be encoded, and must be decoded before it is used. These encoders/decoders are called stream **filters**.
- Commonly used filters
  - ASCIIHexDecode: data encoded in an ASCII hexadecimal representation.
  - LZWDecode: data encoded using the LZW compression method.
  - FlateDecode: data encoded using the zlib/deflate compression method.

A stream object example

```
    << /Length 534
       /Filter [/ASCII85Decode /LZWDecode]
    >>
stream
J..)6T`?p&<!J9%_[umg"B7/Z7KNXbN'S+,*Q/&"OLT'F
LIDK#!n`$"<Atdi`\Vn%b%)&'cA*VnK\CJY(sF>c!Jnl@
RM]WM;jjH6Gnc75idkL5]+cPZKEBPWdR>FF(kj1_R%W_d
&/jS!;iuad7h?[L–F$+]]0A3Ck*$I0KZ?;<)CJtqi65Xb
Vc3\n5ua:Q/=0$W<#N3U;H,MQKqfg1?:lUpR;6oN[C2E4
ZNr8Udn.'p+?#X+1>0Kuk$bCDF/(3fL5]Oq)^kJZ!C2H1
'TO]Rl?Q:&'<5&iP!$Rq;BXRecDN[IJB`,)o8XJOSJ9sD
S]hQ;Rj@!ND)bD_q&C\g:inYC%)&u#:u,M6Bm%lY!Kb1+
":aAa'S`ViJglLb8<W9k6Yl\\0McJQkDeLWdPN?9A'jX*
al>iG1p&i;eVoK&juJHs9%;Xomop"5KatWRT"JQ#qYuL,
JD?M$0QP)lKn06l1apKDC@\qJ4B!!(5m+j.7F790m(Vj8
8l8Q:_CZ(Gm1%X\N1&u!FKHMB~>
endstream
```

Indirect Objects

- Any object in a PDF file may be labeled as an indirect object. This gives the object a **unique object identifier** by which other objects can **refer to** it.
- The definition of an indirect object in a PDF file consists of its **object number** and **generation number**, **followed by the value of the object itself** bracketed between the keywords **obj** and **endobj**.
-

```
12  0  obj
     (Brillig)
endobj
```

Indirect Objects (cont.)

- The object can then be **referred to** from elsewhere in the file by an **indirect reference** consisting of the **object number**, the **generation number**, and the keyword **R**.
- `12 0 R`

Document Structure

- A PDF document is orgnized in a **tree hierarchy**. The root of the tree is called **Document Catalog** and is specified by the `/Root` entry in trailer.
- The catalog contains references to other objects defining the document's **contents**, **outline**, article threads, named destinations, and other attributes.

/Catalog

```
1 0 obj
  << /Type /Catalog
     /Pages 2 0 R
     /PageMode /UseOutlines
     /Outlines 3 0 R
  >>
endobj
```

- `/Type` must be set to /Catalog.
- `/Pages` must be set to specify the document's page tree.

/Pages

```
2 0 obj
  << /Type /Pages
     /Kids [ 4 0 R 5 0 R ]
     /Count 2
  >>
endobj
```

- `/Type` must be set to /Pages.
- `/Kids` array should be set to specify the child pages.
- `/Count` must be set to specify the number of leaf nodes (page objects).

/Page

```
3 0 obj
  << /Type /Page
     /Parent 2 0 R
     /MediaBox [0 0 600 400]
     /Resources << >>
  >>
endobj
```

- `/Type` must be set to /Page.
- `/MediaBox` is a rectangle on the page to store media contents.
- `/Resources` contains any resources (e.g. fonts) that are required by this page.

Content Streams

- A content stream is a PDF stream object whose data consists of a sequence of **instructions** describing the graphical elements to be painted on a page.
- These instructions use a "**postfix**" notation, in which an operator is preceded by its operands.
- An **operand** is a ***direct*** object belonging to any of the basic PDF data types except a stream.
- An **operator** is a PDF keyword that specifies some action to be performed.

Content Streams (cont.)

- Each **page's content** is represented by one or more content streams. Content streams are also used to represent self-contained graphical elements, such as **forms, fonts**, etc.
- An operator may needs to refer to a object that is defined outside the content stream, such as a **font dictionary**. This can be accomplished by defining such objects as **named resources** and referring to them **by name** from within the content stream.
- A content stream's named resources are defined by a **resource dictionary**.

A content stream example

```
4 0 obj
<< /Contents 5 0 R /MediaBox [ 0 0 842 595 ] /Parent 3 0 R /Resources 6 0 R /Type /Page >>
endobj
5 0 obj
<< /Length 312 >>
stream
q Q q /Cs1 cs 0 0 0 sc q 0.24 0 0 0.24 18 434.2 cm BT 300 0 0 300 225 -11
Tm /TT1 1 Tf (H) Tj ET Q q 0.24 0 0 0.24 18 434.2 cm BT 0.0007 Tc 300 0 0 300 442 -11
Tm /TT1 1 Tf [ (el) -1 (l) -1 (o,) 2 ( ) 5 (w) -1 (orl) -1 (d!) ] TJ ET Q
q 0.24 0 0 0.24 18 434.2 cm BT 300 0 0 300 1875 -11 Tm /TT1 1 Tf ( ) Tj ET
Q Qendstream
endobj
6 0 obj
<< /ColorSpace << /Cs1 7 0 R >> /Font << /TT1 8 0 R >> /ProcSet [ /PDF /Text ] >>
endobj
```

Text in PDF

- A **character** is an abstract symbol.
- A **glyph** is a specific graphical rendering of a character.
- Glyphs are organized into **fonts**.
- A font's **encoding** is the association between character codes (obtained from text strings that are shown) and glyph descriptions.

Basics of Showing Text

```
BT
    /F13  12  Tf
    288  720  Td
    (ABC)  Tj
ET
```

- Begin a text object.
- Set the font and font size to use.
- Specify a starting position on the page.
- Paint the glyphs for a string of characters there.
- End the text object.

Text-showing operators

OPERANDS	OPERATOR	DESCRIPTION
string	**Tj**	Show a text string.
string	'	Move to the next line and show a text string. This operator has the same effect as the code T* *string* Tj
a_w a_c *string*	"	Move to the next line and show a text string, using a_w as the word spacing and a_c as the character spacing (setting the corresponding parameters in the text state). a_w and a_c are numbers expressed in unscaled text space units. This operator has the same effect as the code a_w Tw a_c Tc *string* '
array	**TJ**	Show one or more text strings, allowing individual glyph positioning (see implementation note 40 in Appendix H). Each element of *array* can be a string or a number. If the element is a string, this operator shows the string. If it is a number, the operator adjusts the text position by that amount; that is, it translates the text matrix, T_m. The number is expressed in thousandths of a unit of text space (see Section 5.3.3, "Text Space Details," and implementation note 41 in Appendix H). This amount is *subtracted* from the current horizontal or vertical coordinate, depending on the writing mode. In the default coordinate system, a positive adjustment has the effect of moving the next glyph painted either to the left or down by the given amount. Figure 5.11 shows an example of the effect of passing offsets to **TJ**.

PDF Font Types

TYPE	SUBTYPE VALUE	DESCRIPTION
Type 0	**Type0**	*(PDF 1.2)* A *composite* font—a font composed of other fonts, organized hierarchically (see Section 5.6, "Composite Fonts")
Type 1	**Type1**	A font that defines glyph shapes by using a special encoded format (see Section 5.5.1, "Type 1 Fonts")
	MMType1	A *multiple master* font—an extension of the Type 1 font that allows the generation of a wide variety of typeface styles from a single font (see "Multiple Master Fonts" on page 319)
Type 3	**Type3**	A font that defines glyphs with streams of PDF graphics operators (see Section 5.5.4, "Type 3 Fonts")
TrueType	**TrueType**	A font based on the TrueType font format (see Section 5.5.2, "TrueType Fonts")
CIDFont	**CIDFontType0**	*(PDF 1.2)* A CIDFont whose glyph descriptions are based on Type 1 font technology (see Section 5.6.3, "CIDFonts")
	CIDFontType2	*(PDF 1.2)* A CIDFont whose glyph descriptions are based on TrueType font technology (see Section 5.6.3, "CIDFonts")

PDF Font Types (cont.)

- Simple Fonts: Type 1, Type 3, TrueType
- Composite Fonts: Type 0
- CIDFont
  - not actually a font, cannot be used as the operand of the Tf operator.
  - used only as a descendant of a Type 0 font.

Font Data Structures

- A font is represented in PDF as a **dictionary** specifying the type of font, its PostScript name, its encoding, and information that can be used to provide a substitute when the font program is not available.
- The font program itself can be **embedded** as a stream object.
- Two classes of font-related objects to support **CID-Keyed Fonts**:
  - CIDFonts
  - CMaps

A TrueType Font dictionary

```
17  0  obj
    <<  /Type  /Font
        /Subtype  /TrueType
        /BaseFont  /NewYork,Bold
        /FirstChar  0
        /LastChar  255
        /Widths  23 0 R
        /FontDescriptor  7 0 R
        /Encoding  /MacRomanEncoding
    >>
endobj

23  0  obj
    [ 0  333  333  333  333  333  333  333  0  333  333  333  333  333  333  333  333  333
      …Omitted data…
      803  790  803  780  780  780  340  636  636  636  636  636  636  636  636  636  636
    ]
endobj
```

CID-Keyed Fonts (Type 0)

- Provide a convenient and efficient method for defining **multiple-byte character encodings**, fonts with a large number of glyphs, and fonts that incorporate glyphs obtained from other fonts.
- **CID** (character identifier) numbers are used to index and access the glyph descriptions in the font.
- Must explicitly reference the **character collection** on which its CID numbers are based. A character collection is uniquely identified by the **Registry**, **Ordering**, and **Supplement**.
- A **CMap** (character map) file specifies the correspondence between character codes and the CID numbers used to identify characters.

Glyph Selection

- `font name` from the current text state.
- `character codes` from text-showing instruction.
- How to select the correct glyphs to rendering the text?

Type 1 Font Encoding

- A Type 1 font program's glyph descriptions are keyed by **character names**.
- A **character code** is first mapped to **character name** as specified by the font's Encoding. The character name is then used to find the **glyph description**.

Standard Encodings

D.1 Latin Character Set and Encodings

CHAR	NAME	CHAR CODE (OCTAL) STD	MAC	WIN	PDF	CHAR	NAME	CHAR CODE (OCTAL) STD	MAC	WIN	PDF
A	A	101	101	101	101	Œ	OE	352	316	214	226
Æ	AE	341	256	306	306	Ó	Oacute	—	356	323	323
Á	Aacute	—	347	301	301	Ô	Ocircumflex	—	357	324	324
Â	Acircumflex	—	345	302	302	Ö	Odieresis	—	205	326	326
Ä	Adieresis	—	200	304	304	Ò	Ograve	—	361	322	322
À	Agrave	—	313	300	300	Ø	Oslash	351	257	330	330
Å	Aring	—	201	305	305	Õ	Otilde	—	315	325	325
Ã	Atilde	—	314	303	303	P	P	120	120	120	120
B	B	102	102	102	102	Q	Q	121	121	121	121
C	C	103	103	103	103	R	R	122	122	122	122
Ç	Ccedilla	—	202	307	307	S	S	123	123	123	123
D	D	104	104	104	104	Š	Scaron	—	—	212	227
E	E	105	105	105	105	T	T	124	124	124	124
É	Eacute	—	203	311	311	Þ	Thorn	—	—	336	336

Type 3 Font Encoding

- Basically the same as Type 1.

TypeType Font Encoding

- A TrueType font program's "**cmap**" table contains information which maps character codes to glyph descriptions.
- The "cmap" table consists of one or more subtables.
- If a (3, 1) "cmap" subtable (Microsoft Unicode) is present, it is used as follows: A **character code** is first mapped to a **character name** as specified by the font's Encoding. The character name is then mapped to a **Unicode value** by consulting the Adobe Glyph List. Finally, the Unicode value is mapped to a **glyph description** according to the (3, 1) subtable.

TypeType Font Encoding (cont.)

- If no (3, 1) subtable is present but a (1, 0) subtable (Macintosh Roman) is present, it is used as follows: A **character code** is first mapped to a **character name** as specified by the font's Encoding. The character name is then mapped back to a **character code** according to MacRomanEncoding. Finally, the code is mapped to a **glyph description** according to the (1, 0) subtable.
- In either of the cases above, a **character code** is first mapped to a **character name** as specified by the font's Encoding, the character name is looked up in the font program's "post" table (if one is present) and the associated **glyph description** is used.

TypeType Font Encoding (cont. 2)

- When the font **has no Encoding entry**, or the font descriptor's **Symbolic flag is set**, the following occurs:
  - If the font contains a (3, 0) subtable, each two-bytes from the string are used to look up the associated glyph descriptions from the subtable.
  - Otherwise, if the font contains a (1, 0) subtable, single bytes from the string are used to look up the associated glyph descriptions from the subtable.
- If a character cannot be mapped in any of the ways described above, the results are **implementation-dependent**.

Type 0 Font Encoding

- The **Encoding** entry of a Type 0 font dictionary must be **a predefined CMap**, or **a CMap stream** object.
- The number of bytes extracted from the string for each successive character is determined exclusively by the codespace ranges in the CMap (delimited by **begincodespacerange** and **endcodespacerange**).
- The code extracted from the string is then looked up in the character code mappings for codes of that length. (These are the mappings defined by **begincidchar**, **endcidchar**, and corresponding operators for ranges.)

Glyph Selection in CIDFontType0

- The program itself contains glyph descriptions that are identified by **CIDs**.
- If the font program has a Top DICT that uses CIDFont operators: The CIDs are used to determine the GID value using the charset table. The GID value is then used to look up the glyph procedure using the CharStrings INDEX table.
- Otherwise, the CIDs are used directly as GID values, and the glyph procedure is retrieved using the CharStrings INDEX.

Glyph Selection in CIDFontType2

- The program is actually a TrueType font program, which has no native notion of CIDs.
- If the TrueType font program is **embedded**, the Type 2 CIDFont dictionary must contain a **CIDToGIDMap** entry that maps **CIDs** to the **glyph indices** for the appropriate glyph descriptions in that font program.
- Otherwise, the viewer application selects glyphs by translating characters from the encoding specified by the **predefined CMap** to one of the encodings given in the TrueType font's "cmap" table. The results are **implementation-dependent**.

Extraction of Text Content

- In addition to displaying text, consumer applications sometimes need to determine the information content of text.
- This need arises during operations such as **searching**, **indexing**, and **exporting** of text to other applications.
- How to mapping character codes to Unicode values?

Unicode Mapping

- If the font dictionary contains a **ToUnicode CMap**, use that CMap to convert the character code to Unicode.
  - The CMap file must contain begincodespacerange and endcodespacerange operators to define the source character code range.
  - It must use the beginbfchar, endbfchar, beginbfrange, and endbfrange o tors to define the mapping from character codes to Unicode character sequences expressed in **UTF-16BE** encoding.

A ToUnicode CMap example

```
begincmap
/CIDSystemInfo
<< /Registry (Adobe)
/Ordering (UCS)
/Supplement 0
>> def
/CMapName /Adobe–Identity–UCS def
/CMapType 2 def
1 begincodespacerange
<0000> <FFFF>
endcodespacerange
2 beginbfrange
<0000> <005E> <0020>
<005F> <0061> [<00660066> <00660069> <00660066006C>]
endbfrange
1 beginbfchar
<3A51> <D840DC3E>
endbfchar
endcmap
```

ligatures ff, fi, and ffl.

U+2003E in a surrogate pair

Ideograph bui5
CJK Unified Ideographs Extension B

U+2003E
𠀾

Unicode Mapping (cont.)

If the font is a simple font that uses one of the predefined encodings **MacRomanEncoding**, **MacExpertEncoding**, or **WinAnsiEncoding**, or that has an encoding whose **Differences** array includes only character names taken from the Adobe standard Latin character set and the set of named characters in the Symbol font (see Appendix D):

1. Map the character code to a character name according to Table D.1 on page 996 and the font's **Differences** array.
2. Look up the character name in the *Adobe Glyph List* (see the Bibliography) to obtain the corresponding Unicode value.

Unicode Mapping (cont. 2)

If the font is a composite font that uses one of the predefined CMaps listed in Table 5.15 on page 442 (except Identity–H and Identity–V) or whose descendant CIDFont uses the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, or Adobe-Korea1 character collection:

1. Map the character code to a character identifier (CID) according to the font's CMap.
2. Obtain the registry and ordering of the character collection used by the font's CMap (for example, Adobe and Japan1) from its **CIDSystemInfo** dictionary.
3. Construct a second CMap name by concatenating the registry and ordering obtained in step 2 in the format *registry–ordering*–UCS2 (for example, Adobe–Japan1–UCS2).
4. Obtain the CMap with the name constructed in step 3 (available from the ASN Web site; see the Bibliography).
5. Map the CID obtained in step 1 according to the CMap obtained in step 4, producing a Unicode value.

Adobe-GB1-UCS2 CMap

```
31  begincmap
32
33  /CIDSystemInfo 3 dict dup begin
34    /Registry (Adobe) def
35    /Ordering (Adobe_GB1_UCS2) def
36    /Supplement 4 def
37  end def
38
39  /CMapName /Adobe-GB1-UCS2 def
40
41  /CMapVersion 6.001 def
42  /CMapType 1 def
43
44  /XUID [1 10 25335 1212] def
45
46  /WMode 0 def
47
48  1 begincodespacerange
49    <0000> <FFFF>
50  endcodespacerange
51
52  100 beginbfchar
53  <0000> <FFFD>
54  <0063> <00B7>
55  <0064> <02C9>
56  <0065> <02C7>
57  <0066> <00A8>
58  <0067> <3003>
59  <0068> <3005>
60  <0069> <2014>
```

That's all.