Advanced PDF TRICKS

KURT PFEIFLE
ANGE ALBERTINI

TROOPERS
MAKE THE WORLD A SAFER PLACE

ANGE ALBERTINI

reverse engineering

VISUAL DOCUMENTATIONS

@angealbertini

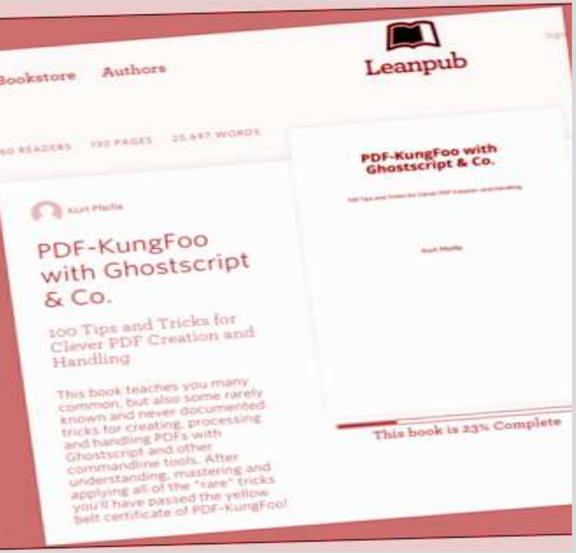
ange@corkami.com

http://www.corkami.com

Kurt Pfeifle

@pdfkungfoo

<kurt.pfeifle@mykolab.com>



Bookstore Authors

Leanpub

60 READERS 132 PAGES 25,497 WORDS

Kurt Pfeifle

PDF-KungFoo with Ghostscript & Co.

100 Tips and Tricks for Clever PDF Creation and Handling

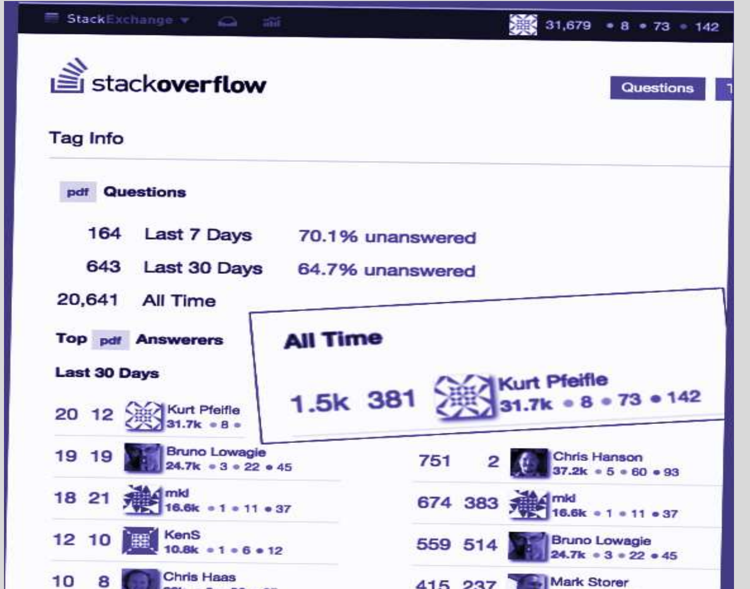
This book teaches you many common, but also some rarely known and never documented tricks for creating, processing and handling PDFs with Ghostscript and other commandline tools. After understanding, mastering and applying all of the "rare" tricks you'll have passed the yellow belt certificate of PDF-KungFoo!

PDF-KungFoo with Ghostscript & Co.

Kurt Pfeifle

This book is 23% Complete

PDF-Answers: https://stackoverflow.com/users/359307/kurt-pfeifle



StackExchange

31,679 • 8 • 73 • 142

stackoverflow

Questions

Tag Info

pdf Questions

164	Last 7 Days	70.1% unanswered
643	Last 30 Days	64.7% unanswered
20,641	All Time	

Top pdf Answerers

Last 30 Days

20	12	Kurt Pfeifle 31.7k • 8 •
19	19	Bruno Lowagie 24.7k • 3 • 22 • 45
18	21	mkl 16.6k • 1 • 11 • 37
12	10	KenS 10.8k • 1 • 6 • 12
10	8	Chris Haas

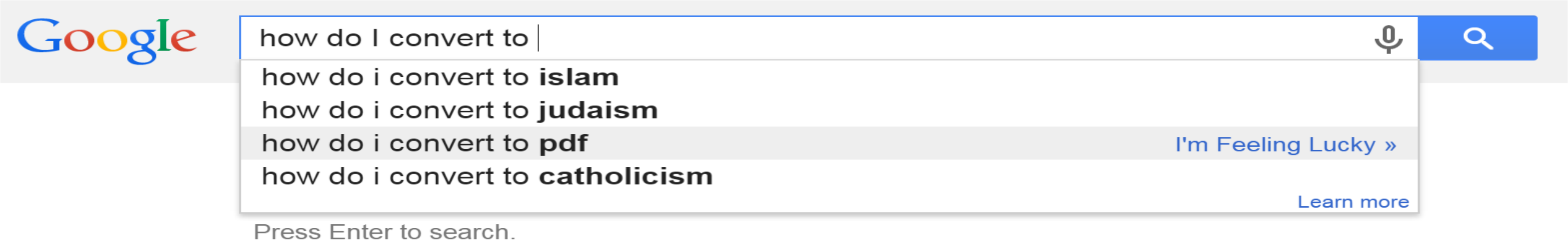
All Time

1.5k	381	Kurt Pfeifle 31.7k • 8 • 73 • 142
751	2	Chris Hanson 37.2k • 5 • 60 • 93
674	383	mkl 16.6k • 1 • 11 • 37
559	514	Bruno Lowagie 24.7k • 3 • 22 • 45
415	237	Mark Storer

PDF-KungFoo with Ghostscript & Co.

100 Tips and Tricks for Clever PDF Creation and Handling

https://leanpub.com/pdfkungfoo/



how do I convert to

how do i convert to **islam**

how do i convert to **judaism**

how do i convert to **pdf**

how do i convert to **catholicism**

Press Enter to search.

Recently, PDF officially became a religion…
… so here we are, Pope and Akuma ;)

Goal:

learn PDF *internals*

("just suck less about the format")

PDF 1.7 spec is 750+ pages, but...
additional *"normative"* references to more than other 80 specs (not all public), 10.000+ pages

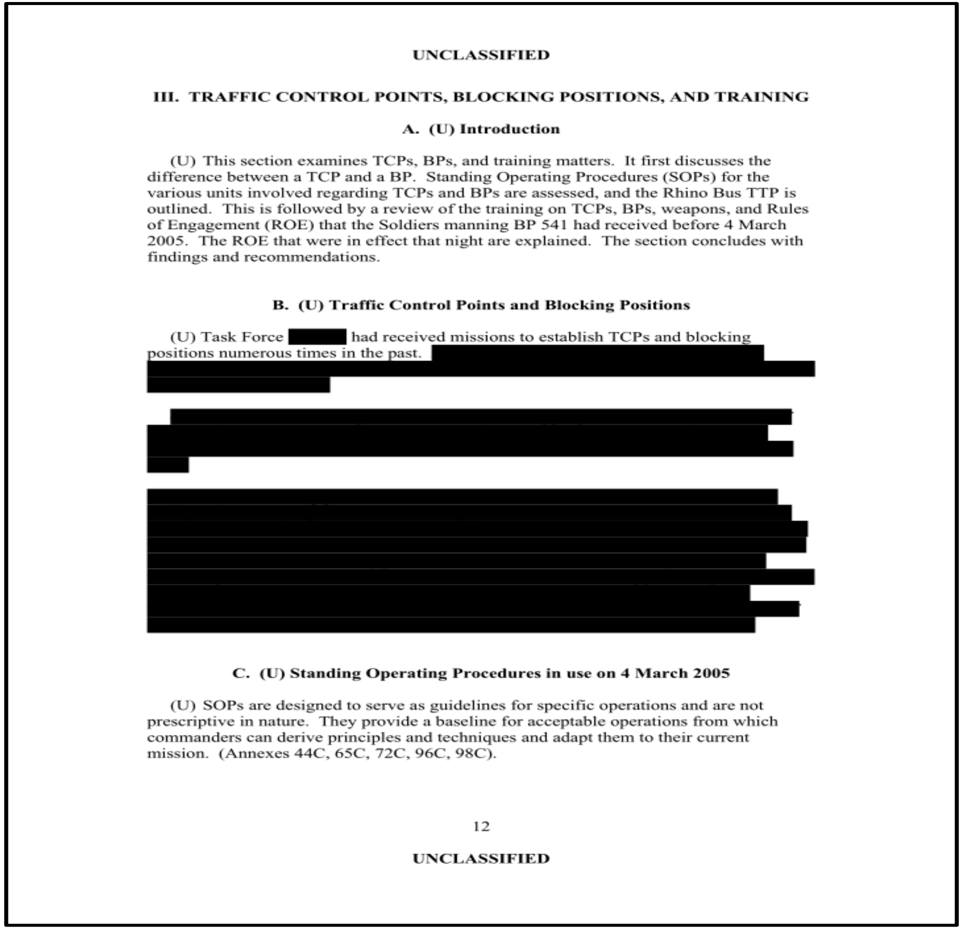
Applications:

watermarks
censorship
edits & tricks...

(create PDFs which don't immediately jump out as amateurish because of their syntax errors
(nothing to do with malicious PDFs analysis or exploitation))

example: hand-written title

A real life example



UNCLASSIFIED

III. TRAFFIC CONTROL POINTS, BLOCKING POSITIONS, AND TRAINING

A. (U) Introduction

(U) This section examines TCPs, BPs, and training matters. It first discusses the difference between a TCP and a BP. Standing Operating Procedures (SOPs) for the various units involved regarding TCPs and BPs are assessed, and the Rhino Bus TTP is outlined. This is followed by a review of the training on TCPs, BPs, weapons, and Rules of Engagement (ROE) that the Soldiers manning BP 541 had received before 4 March 2005. The ROE that were in effect that night are explained. The section concludes with findings and recommendations.

B. (U) Traffic Control Points and Blocking Positions

(U) Task Force [redacted] had received missions to establish TCPs and blocking positions numerous times in the past. [redacted]

[redacted]

[redacted]

C. (U) Standing Operating Procedures in use on 4 March 2005

(U) SOPs are designed to serve as guidelines for specific operations and are not prescriptive in nature. They provide a baseline for acceptable operations from which commanders can derive principles and techniques and adapt them to their current mission. (Annexes 44C, 65C, 72C, 96C, 98C).

12

UNCLASSIFIED

But the 'redactor' guy really botched that job !

Seen in its metadata: *“EmailSubject (Another Redact Job For You)”*
http://download.repubblica.it/pdf/rapportousacalipari.pdf

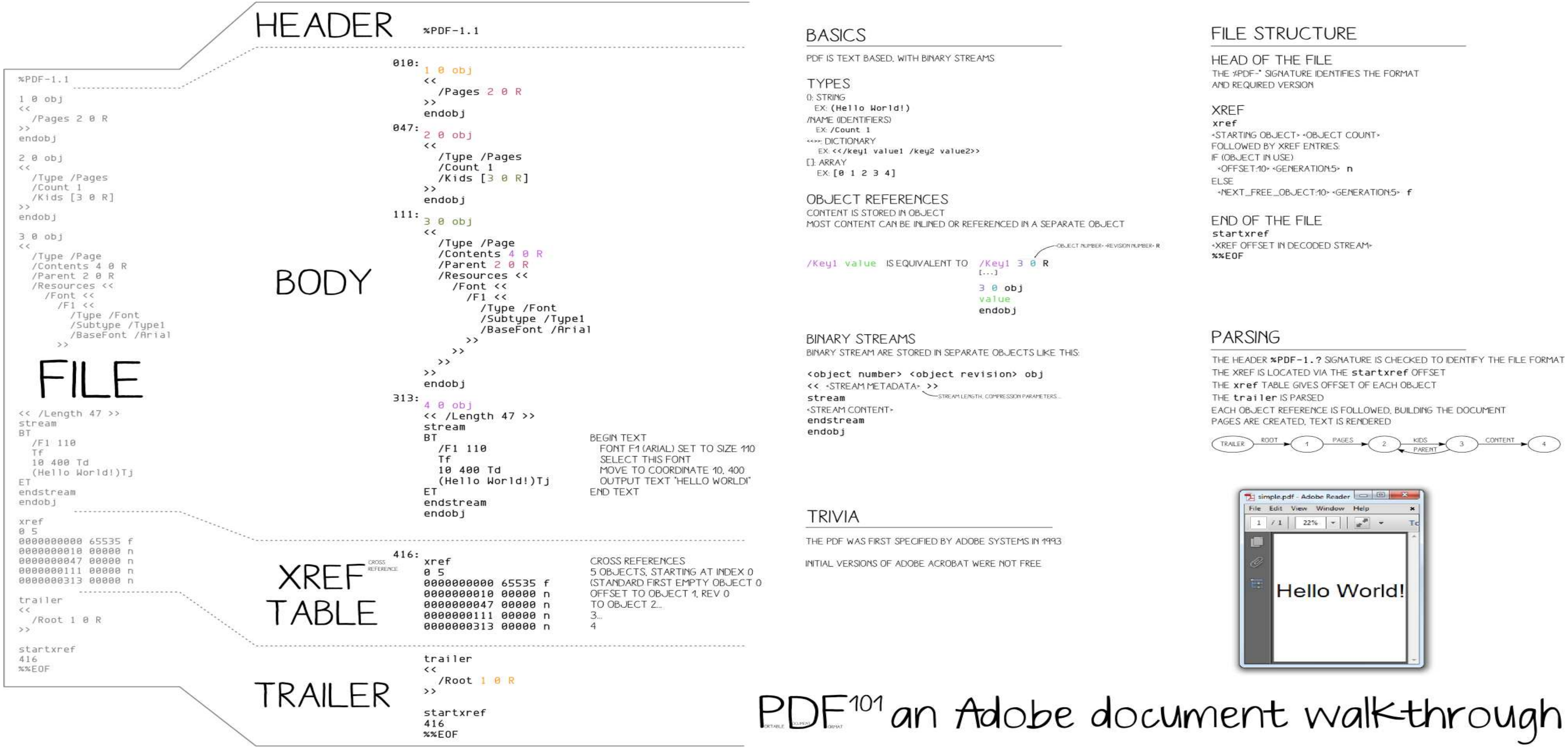
Preamble

this presentation is supplemented by many more hands-on examples, that you can find at:

http://pdf101.corkami.com

PDF 101

basics of the PDF file format
Part I / II



FILE

```
%PDF-1.1

1 0 obj
<<
  /Pages 2 0 R
>>
endobj

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

3 0 obj
<<
  /Type /Page
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 <<
        /Type /Font
        /Subtype /Type1
        /BaseFont /Arial
      >>

<< /Length 47 >>
stream
BT
  /F1 110
  Tf
  10 400 Td
  (Hello World!)Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000047 00000 n
0000000111 00000 n
0000000313 00000 n

trailer
<<
  /Root 1 0 R
>>

startxref
416
%%EOF
```

HEADER

```
%PDF-1.1
```

BODY

```
010:
     1 0 obj
     <<
       /Pages 2 0 R
     >>
     endobj
047:
     2 0 obj
     <<
       /Type /Pages
       /Count 1
       /Kids [3 0 R]
     >>
     endobj
111:
     3 0 obj
     <<
       /Type /Page
       /Contents 4 0 R
       /Parent 2 0 R
       /Resources <<
         /Font <<
           /F1 <<
             /Type /Font
             /Subtype /Type1
             /BaseFont /Arial
           >>
         >>
       >>
     >>
     endobj
313:
     4 0 obj
     << /Length 47 >>
     stream
     BT
       /F1 110
       Tf
       10 400 Td
       (Hello World!)Tj
     ET
     endstream
     endobj
```

BEGIN TEXT
  FONT F1 (ARIAL) SET TO SIZE 110
  SELECT THIS FONT
  MOVE TO COORDINATE 10, 400
  OUTPUT TEXT "HELLO WORLD!"
END TEXT

XREF TABLE

CROSS REFERENCE

```
416:
     xref
     0 5
     0000000000 65535 f
     0000000010 00000 n
     0000000047 00000 n
     0000000111 00000 n
     0000000313 00000 n
```

CROSS REFERENCES
5 OBJECTS, STARTING AT INDEX 0
(STANDARD FIRST EMPTY OBJECT 0
OFFSET TO OBJECT 1, REV 0
TO OBJECT 2...
3...
4

TRAILER

```
trailer
<<
  /Root 1 0 R
>>

startxref
416
%%EOF
```

BASICS

PDF IS TEXT BASED, WITH BINARY STREAMS

TYPES

(): STRING
  EX: `(Hello World!)`
/NAME (IDENTIFIERS)
  EX: `/Count 1`
<<>>: DICTIONARY
  EX: `<</key1 value1 /key2 value2>>`
[]: ARRAY
  EX: `[0 1 2 3 4]`

OBJECT REFERENCES

CONTENT IS STORED IN OBJECT
MOST CONTENT CAN BE INLINED OR REFERENCED IN A SEPARATE OBJECT

`/Key1 value` IS EQUIVALENT TO

```
/Key1 3 0 R
[...]

3 0 obj
value
endobj
```

<OBJECT NUMBER> <REVISION NUMBER> R

BINARY STREAMS

BINARY STREAM ARE STORED IN SEPARATE OBJECTS LIKE THIS:

```
<object number> <object revision> obj
<< <STREAM METADATA> >>
stream
<STREAM CONTENT>
endstream
endobj
```

STREAM LENGTH, COMPRESSION PARAMETERS...

TRIVIA

THE PDF WAS FIRST SPECIFIED BY ADOBE SYSTEMS IN 1993

INITIAL VERSIONS OF ADOBE ACROBAT WERE NOT FREE

FILE STRUCTURE

HEAD OF THE FILE

THE %PDF-* SIGNATURE IDENTIFIES THE FORMAT
AND REQUIRED VERSION

XREF

`xref`
<STARTING OBJECT> <OBJECT COUNT>
FOLLOWED BY XREF ENTRIES:
IF (OBJECT IN USE)
  <OFFSET:10> <GENERATION:5> `n`
ELSE
  <NEXT_FREE_OBJECT:10> <GENERATION:5> `f`

END OF THE FILE

`startxref`
<XREF OFFSET IN DECODED STREAM>
`%%EOF`

PARSING

THE HEADER `%PDF-1.?` SIGNATURE IS CHECKED TO IDENTIFY THE FILE FORMAT
THE XREF IS LOCATED VIA THE `startxref` OFFSET
THE `xref` TABLE GIVES OFFSET OF EACH OBJECT
THE `trailer` IS PARSED
EACH OBJECT REFERENCE IS FOLLOWED, BUILDING THE DOCUMENT
PAGES ARE CREATED, TEXT IS RENDERED

PDF^{101} an Adobe document walkthrough

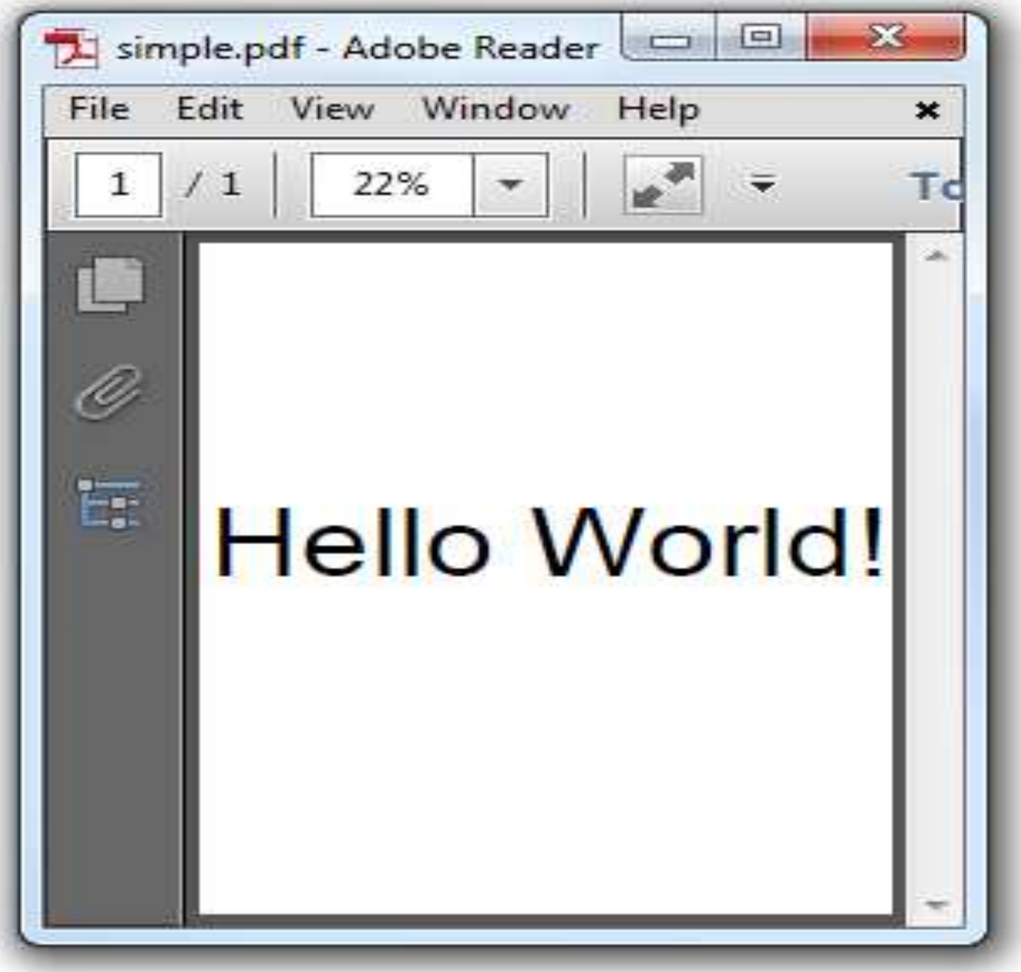
PORTABLE DOCUMENT FORMAT

My poster on the PDF format (free to print, reuse...) http://pics.corkami.com
to order a print: http://prints.corkami.com

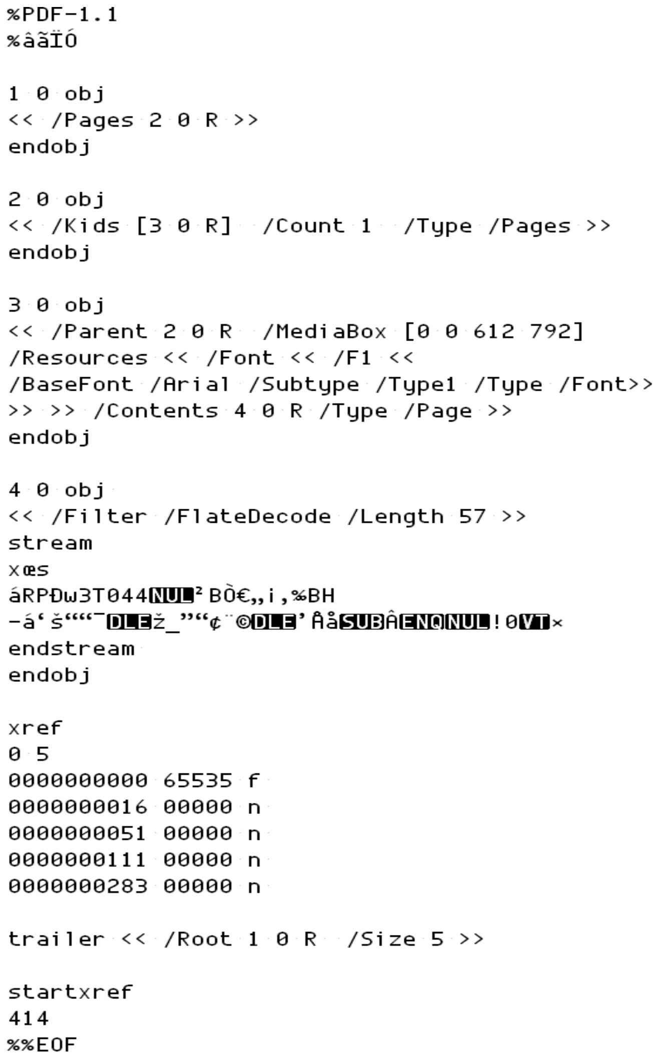
A simple example

helloworld_bin.pdf

reminder: this is simplified, PDF is actually much more complex



Hello World!

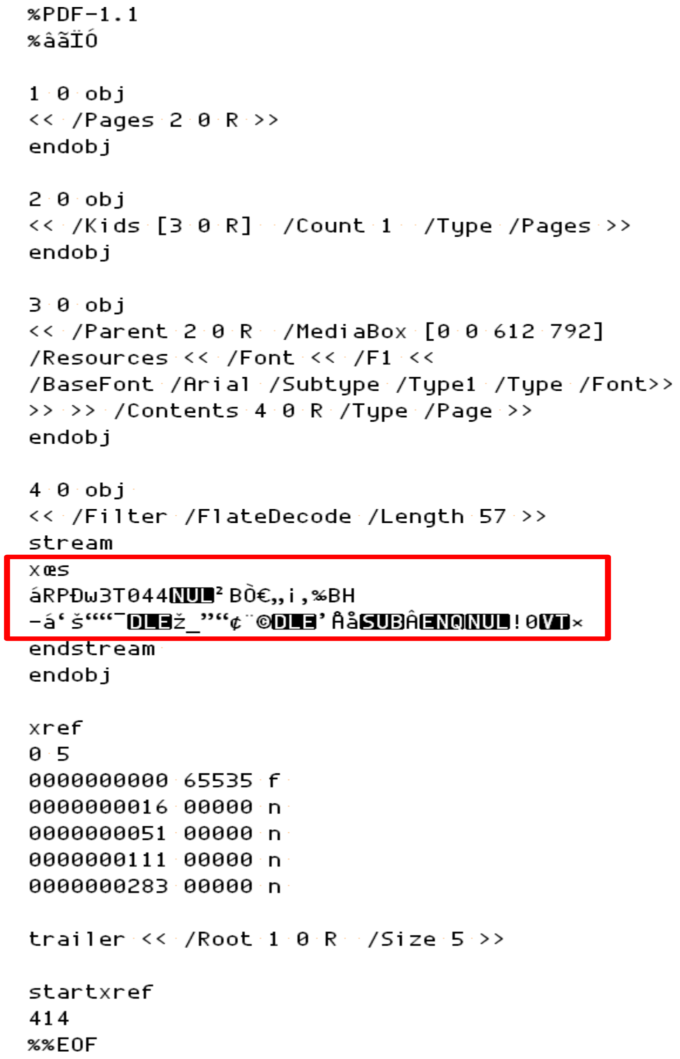


```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPĐw3T044NUL²BÒ€„i,%BH
-á‘š““““¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

text

binary

text



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPĐw3T044NUL²BÒ€„i,‰BH
–á‘š““‘¯DLEž_”‘‘¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

A PDF file is

- text-based
    - white-space tolerant
- with binary streams

→ it can be edited with any decent text editor
(that keeps binary and EOLs intact)

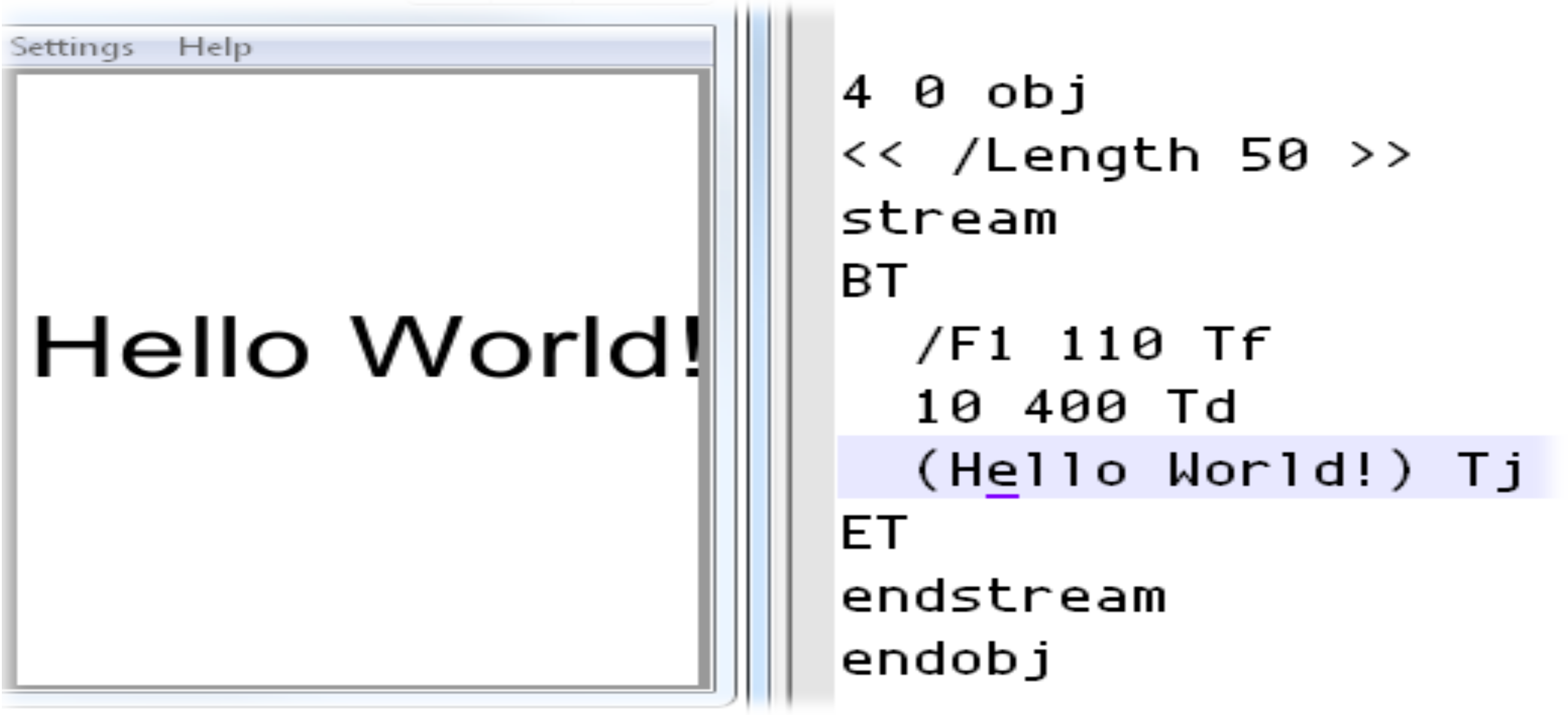
Recommended environment

- Text editor
- Evince/Sumatra/MuPDF/Zathura
  - lightweight
  - updates on the fly
- Tool to decompress streams and unbundle object streams
  - (explanations later)
- Check for mistakes with

  `qpdf --check` *or* **`pdfinfo`** *or* **`Ghostscript`**

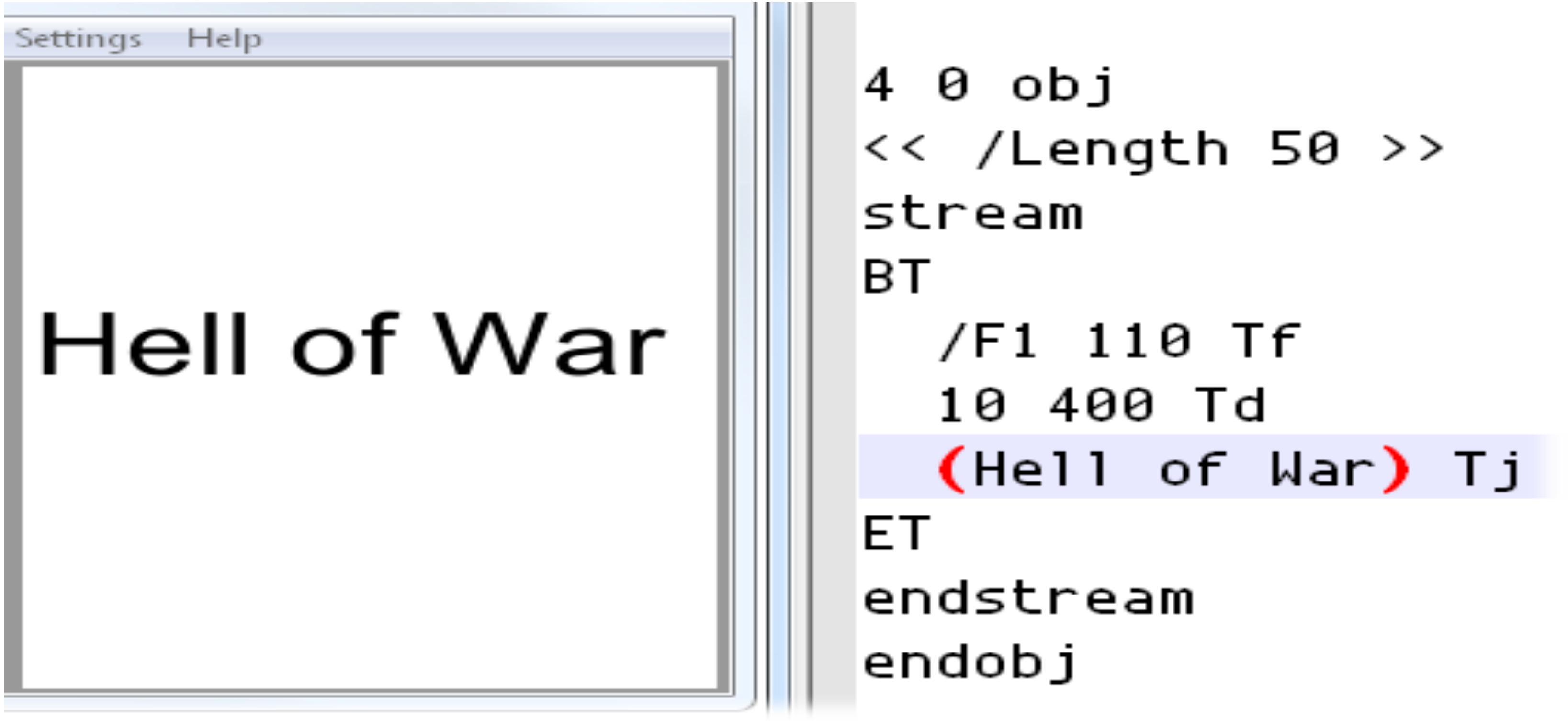
Exercise: manipulate content

whitespace doesn’t change anything
(well, at least in most cases...)



```
4 0 obj
<< /Length 50 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj
```

Update content, save...



```
4 0 obj
<< /Length 50 >>
stream
BT
   /F1 110 Tf
   10 400 Td
   (Hell of War) Tj
ET
endstream
endobj
```

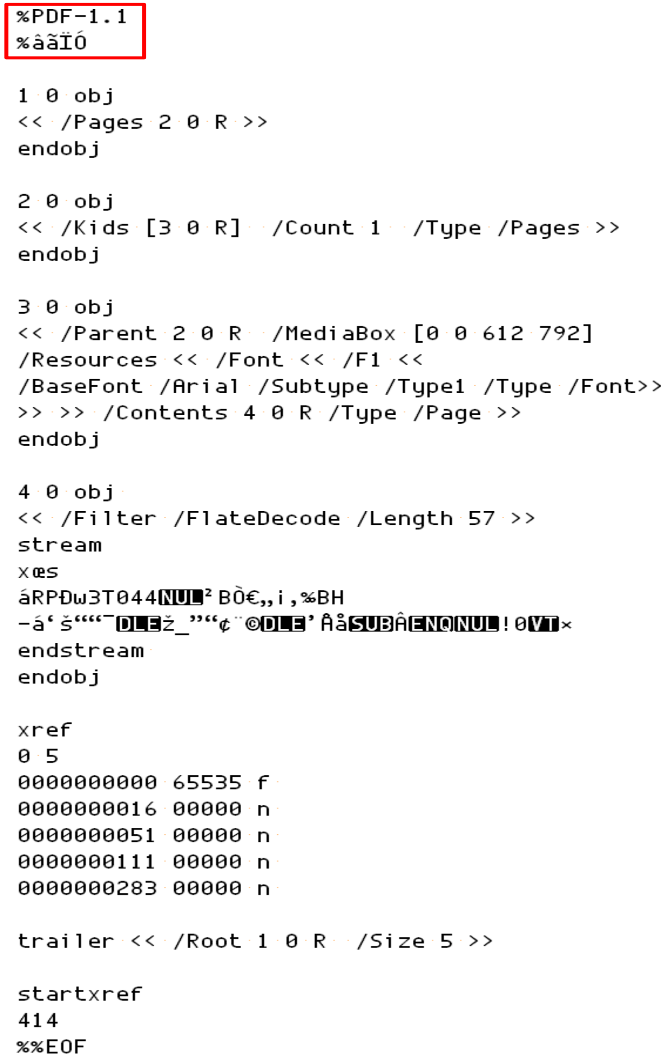
...and you see the result straight away.

Basic PDF structure

1. header
   - signature
2. body
   - made up of "*indirect* objects"
3. cross-reference table
4. trailer
   - cross-reference table
   - trailer dictionary
   - startxref pointer
   - end of file signature

Signature (2 lines)

1. PDF signature
   - `%PDF-1.0 ... %PDF-1.7`
2. Charset identifier
   - not required
   - tells tools file is not ASCII
   - 4 non-ASCII chars in a comment



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL²BÒ€„i,%BH
-á‘š““‘¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Body

made of objects

```
<number> <generation> obj
   <content>
   endobj
```

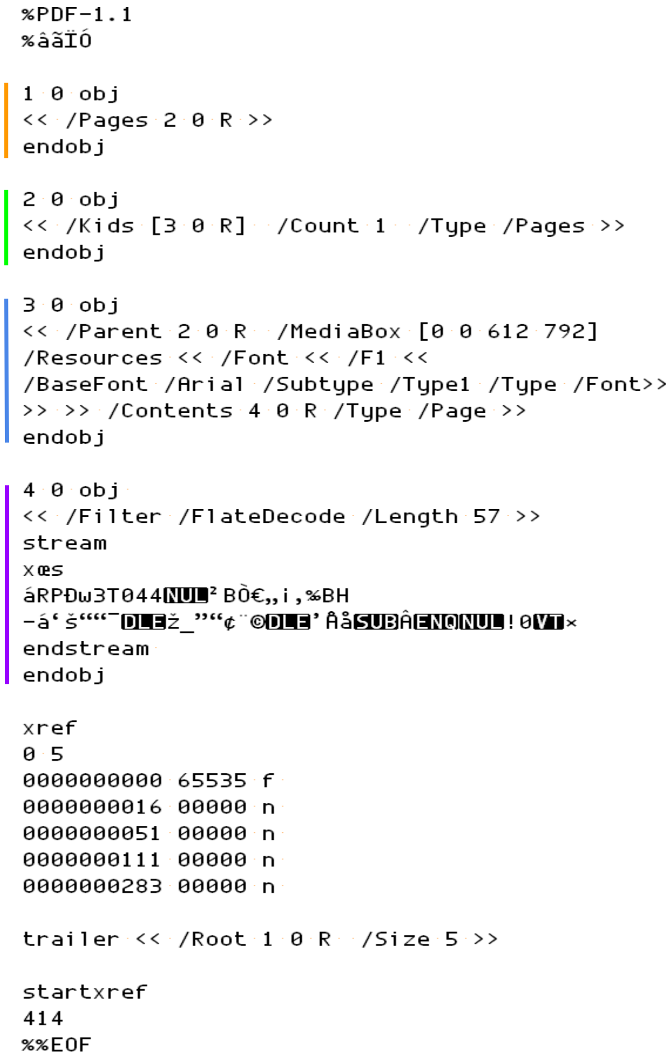
# <generation> in most cases '0'.

<content> frequently composed of:

```
<<..dictionary..>>
```
(double angle brackets)

and optionally

```
stream              # (start keyword)
<streamcontent>     # (can be anything)
endstream           # (end keyword)
```



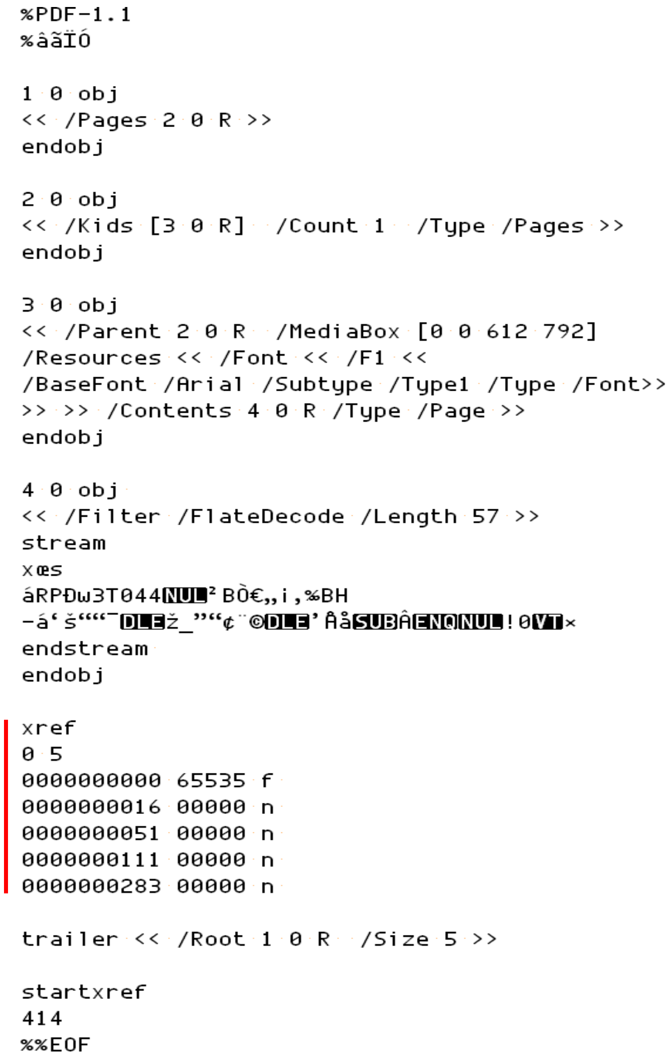
```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL²BÒ€„i,‰BH
–á‘š““¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Xref

- table
- byte offsets for each object

```
xref
0 5                          5 objects, starting at 0
0000000000 65535 f           obj #0: always null (dummy obj)
0000000016 00000 n           obj #1: offset 16 from filestart
0000000051 00000 n           obj #2: offset 51
0000000111 00000 n           ...
0000000283 00000 n
```

- each line = 20 chars exactly!
    - EOL char = <CR> *or* <LF> *or* <CR><LF>
    - if EOL is single byte (<CR> or <LF>), then use extra 1 space before EOL!

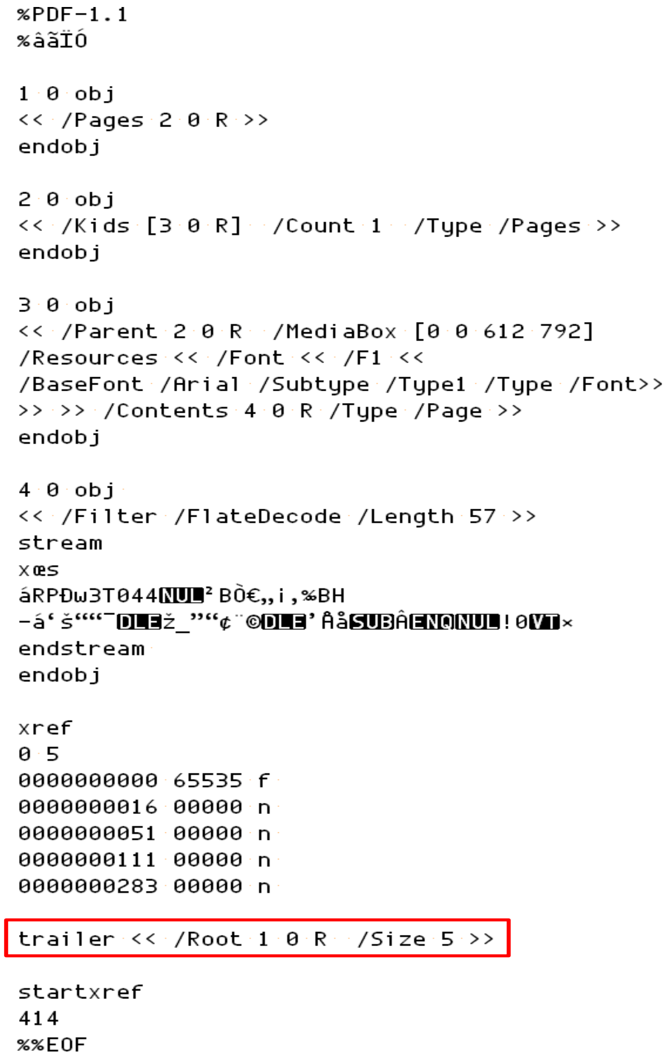


```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPĐw3T044NUL² BÒ€„i,‰BH
–á‘š““¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Trailer 1/2

- structure
  a. “*trailer*”
  b. dictionary (like most objects)
  c. "*startxref*" info
  d. "*%%EOF*"

- dict points to “root” object
  - `/Size` = #(xref elements)
  - `/Root` (can be any number)



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL²BÒ€„i,‰BH
–á‘š““‘¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

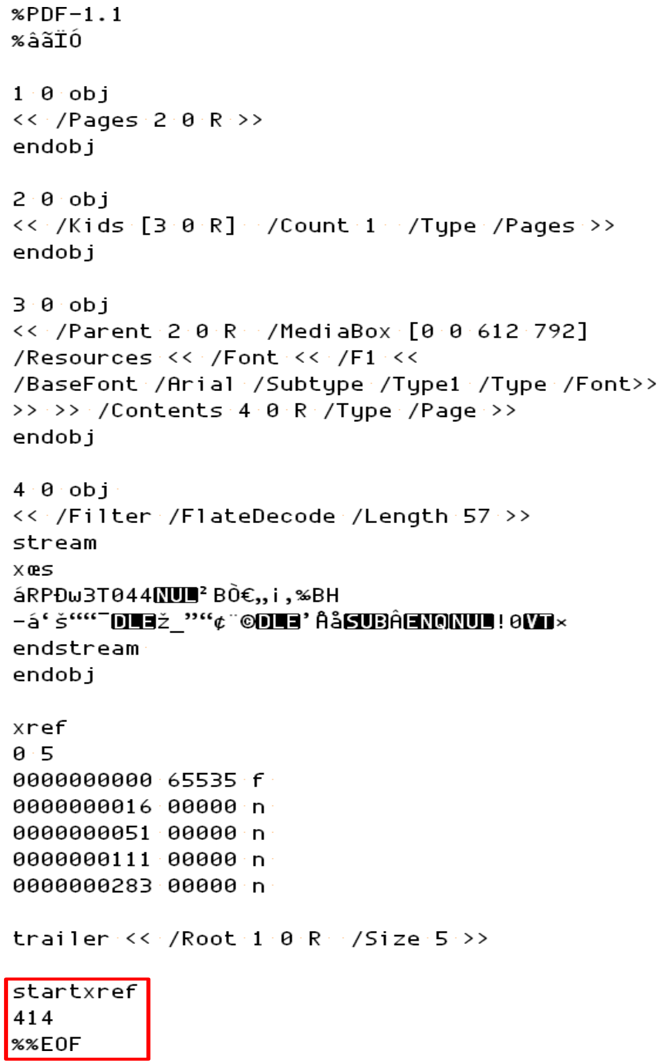
Trailer 2/2

1. pointer to xref
   a. “startxref”
   b. offset to "xref"
      - (decimal)
2. End Of File marker
   a. "%%EOF"

Note: Some real world files after PDF-1.5 may use a *'cross reference stream'* instead of an xref table. Compressed, not directly readable. Not discussed in this talk.

To turn them into a standard cross reference table, use:

```
   qpdf --qdf --object-streams=disable \
        in.pdf uncompressed.pdf
```



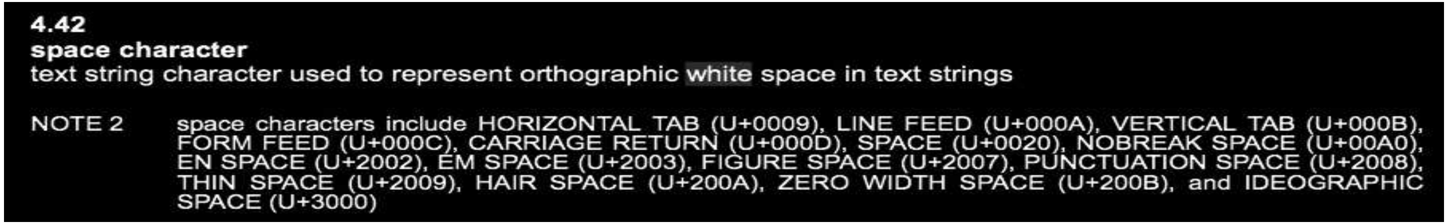
```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPĐw3T044NUL²BÒ€„i,%BH
-á‘š““‘¯DLEž_”“¢¨©DLE’ÂåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Basic types

boolean, *numbers*, strings, names, arrays, dictionaries, streams, *null*...

(Basic types often separated from each other by whitespace. Sometimes no whitespace *required* because of specific delimiters assigned to the respective basic types...)

Space, Whitespace & Delimiters



4.42
space character
text string character used to represent orthographic white space in text strings

NOTE 2 space characters include HORIZONTAL TAB (U+0009), LINE FEED (U+000A), VERTICAL TAB (U+000B), FORM FEED (U+000C), CARRIAGE RETURN (U+000D), SPACE (U+0020), NOBREAK SPACE (U+00A0), EN SPACE (U+2002), EM SPACE (U+2003), FIGURE SPACE (U+2007), PUNCTUATION SPACE (U+2008), THIN SPACE (U+2009), HAIR SPACE (U+200A), ZERO WIDTH SPACE (U+200B), and IDEOGRAPHIC SPACE (U+3000)

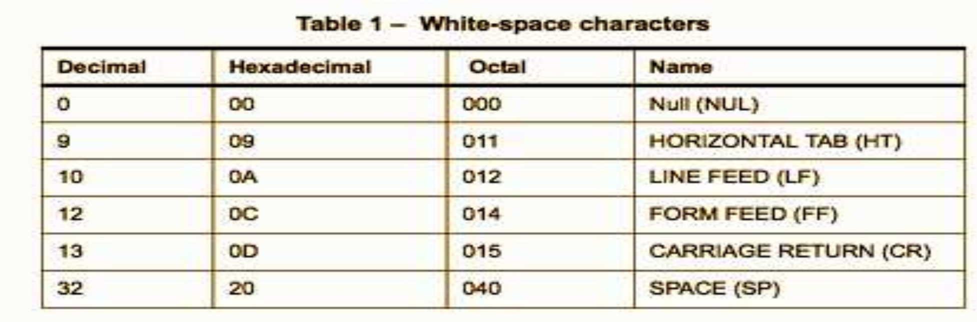


Table 1 – White-space characters

Decimal	Hexadecimal	Octal	Name
0	00	000	Null (NUL)
9	09	011	HORIZONTAL TAB (HT)
10	0A	012	LINE FEED (LF)
12	0C	014	FORM FEED (FF)
13	0D	015	CARRIAGE RETURN (CR)
32	20	040	SPACE (SP)

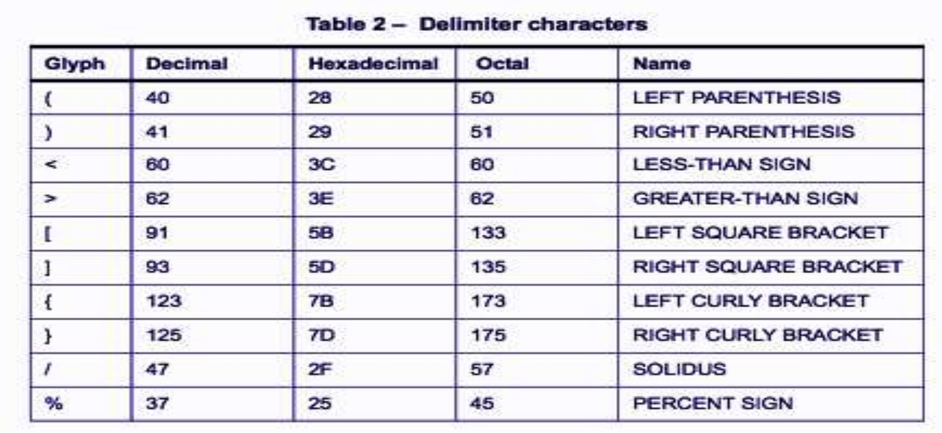


Table 2 – Delimiter characters

Glyph	Decimal	Hexadecimal	Octal	Name
(	40	28	50	LEFT PARENTHESIS
)	41	29	51	RIGHT PARENTHESIS
<	60	3C	60	LESS-THAN SIGN
>	62	3E	62	GREATER-THAN SIGN
[	91	5B	133	LEFT SQUARE BRACKET
]	93	5D	135	RIGHT SQUARE BRACKET
{	123	7B	173	LEFT CURLY BRACKET
}	125	7D	175	RIGHT CURLY BRACKET
/	47	2F	57	SOLIDUS
%	37	25	45	PERCENT SIGN

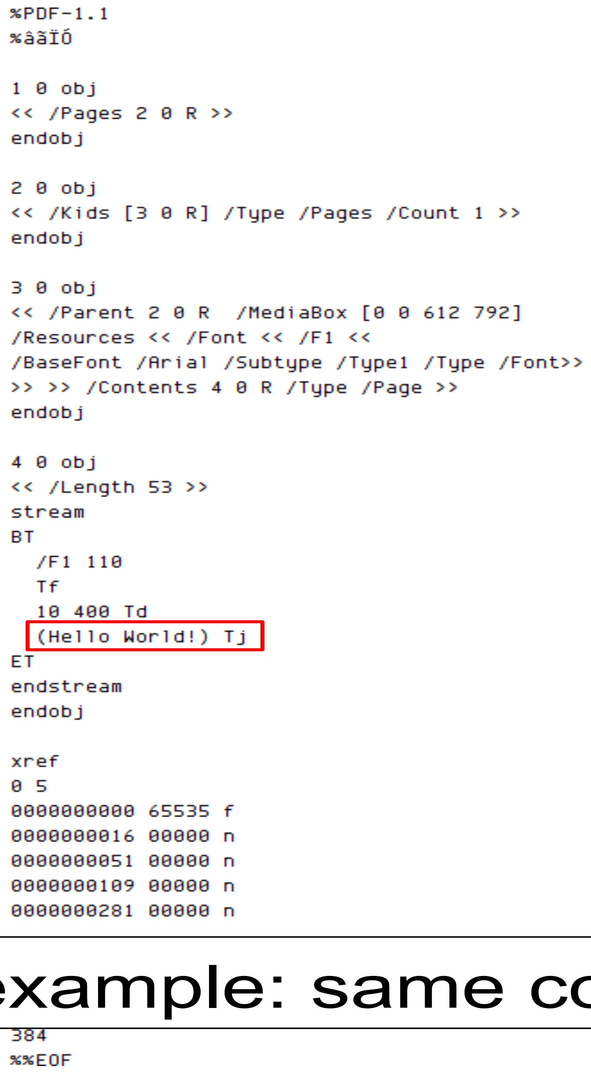
Strings/Literals

`%comment until line return`

- `(string)` ⇐ ASCII
- `(\163\164\162\151\156\147)` ⇐ octal
- `(str\151ng)` ⇐ mix of octal & ASCII
- `<686578>` ⇐ hex

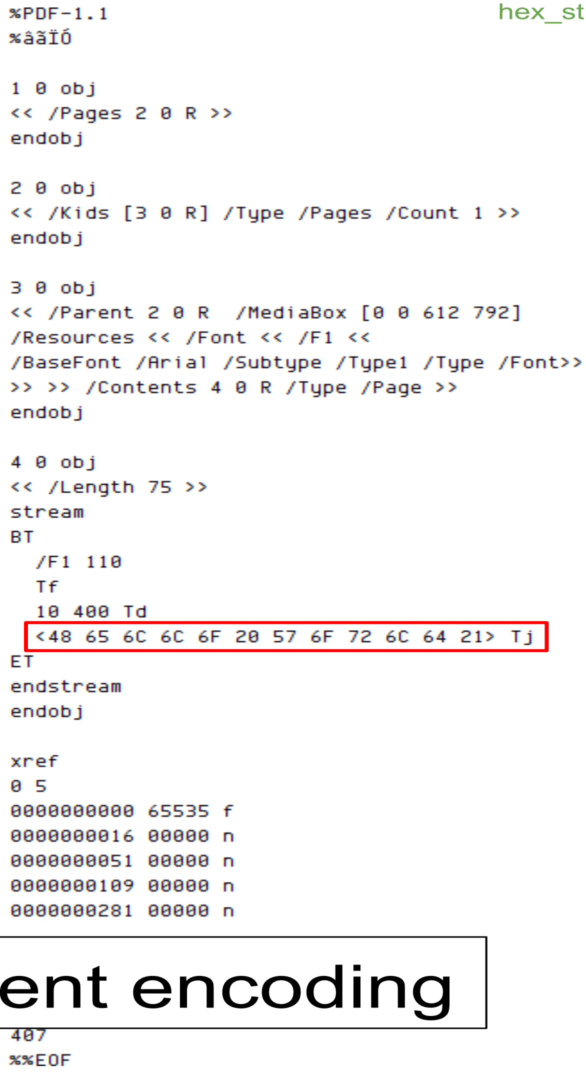
-
  ```
  <686 5   7
    8>
  ```
  ⇐ separated nibbles

(PDF is *quite* f*cked up)



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110
  Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

384
%%EOF
```

hex_string.pdf



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 75 >>
stream
BT
  /F1 110
  Tf
  10 400 Td
  <48 65 6C 6C 6F 20 57 6F 72 6C 64 21> Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

407
%%EOF
```

example: same content, different encoding

1 0 R

an important fact to know when you read PDF

Object reference

the declaration

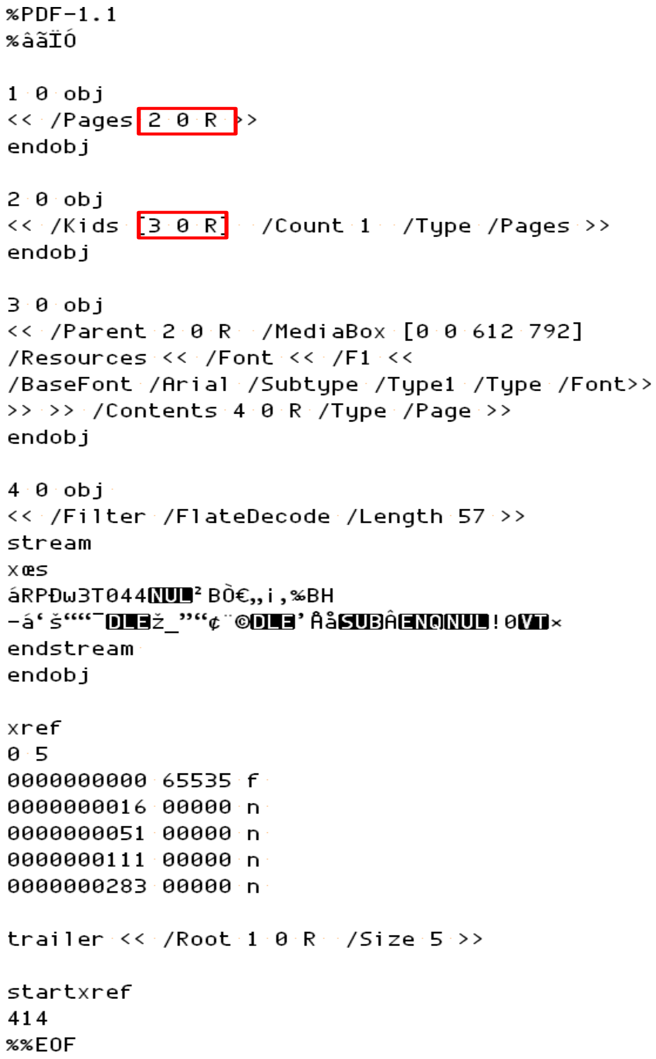
- `<object> <generation> R`

refers to

- the actual contents of the object

some objects CAN'T be inlined

`<generation>` is *very rarely* non-zero



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL² BÒ€„i,‰BH
–á‘š““““¯DLEž_”““¢¨©DLE’ÂåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Object reference - example

```
/Count 1 ...
```

```
/Count 5 0 R ...

5 0 obj
1
endobj
```

2 equivalent examples via object reference

Object references: syntax

It's odd, but critical to understand

- `3 0 1` ⇒ 3 elements (3 numbers):
  a. `3`
  b. `0`
  c. `1`
- `3 0 R` ⇒ 1 element:
  a. reference to "`3 0`"
     - object `3`
     - generation `0`

Other PDF syntax rules follow common-sense

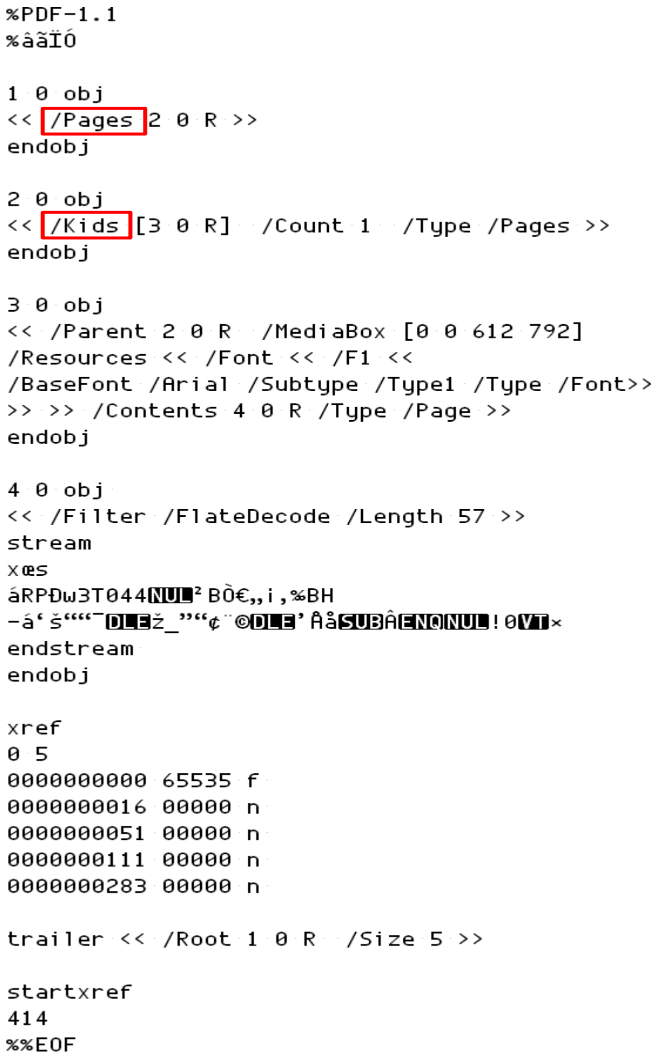
Name objects

- “reserved keywords”
  - like symbols in Ruby
- starts with /
  - "/Pages" , "/Kids" ...
- case sensitive
  - CamelCase by default
  - undefined names are ignored

⇒ /pages != /Pages

but /Pages == /P#61ges ☺

(useful to disable or to obfuscate things...)



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL² BÒ€„i,‰BH
-á‘š““““¯DLEž_”““¢¨©DLE’ÂåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Exercise:
identify basic types

boolean, *numbers*, strings, names,
arrays, dictionaries, streams, *null*...

Exercise:
add/edit names

bogus names ignored, case sensitivity
the reader may fall back to default values

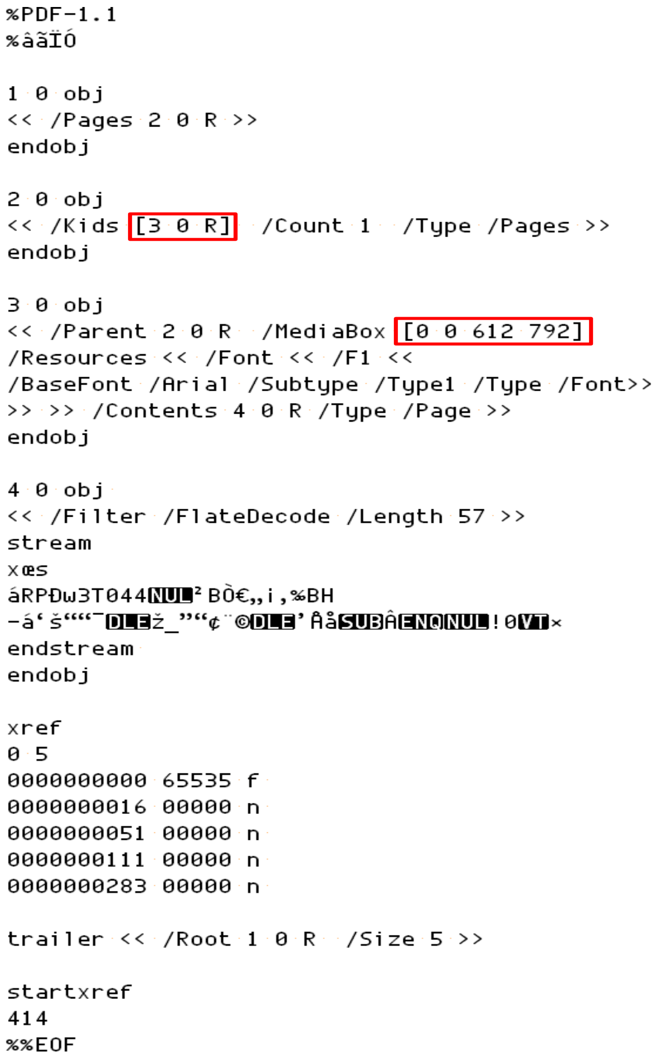
Arrays

Syntax

- [<values>*]

Examples:

- [3 0 R] = 1 value
  a. “3 0 R”
- [0 0 612 792] = 4 values
  a. “0”
  b. “0”
  c. “612”
  d. “792”



```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL² BÒ€„i,‰BH
–á‘š““““¯DLEž_”““¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Dictionaries

Syntax:

- **<<** [<key> <value>]* **>>**

# <keys> must be "names", must follow the rules for "names", which is why...

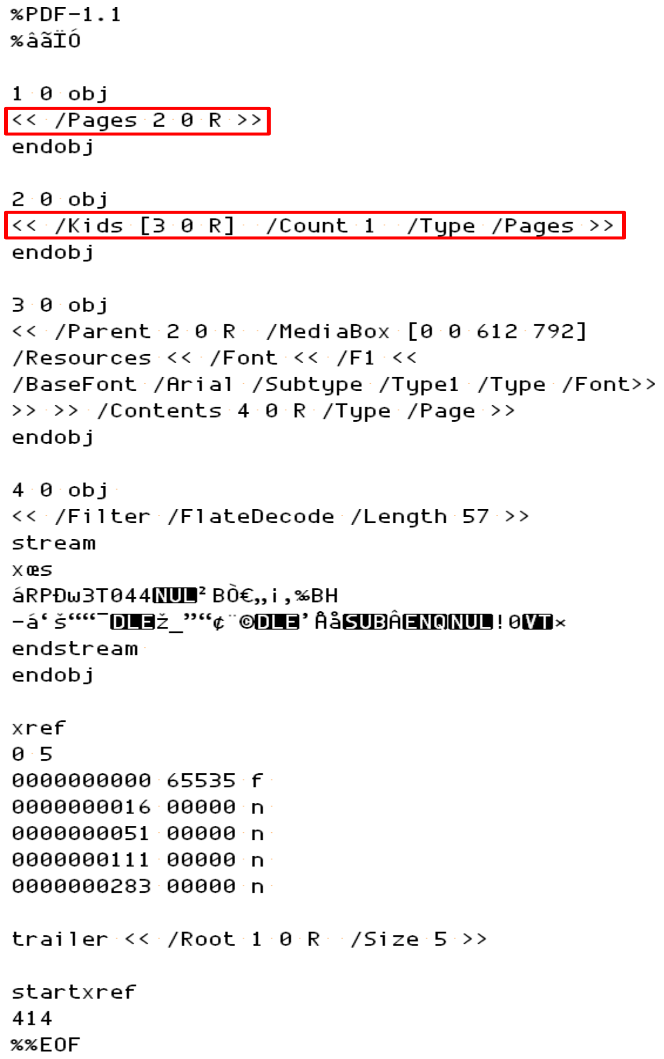
# ...<keys> always start with forward slashes: /Name1, /Something, /Kids, /Type,...

Object 1 sets:

1. */Pages* to "*2 0 R*" # (to an obj reference)

Object 2 sets:

1. */Kids* to "*[3 0 R]*" # (to an array)
2. */Count* to "*1*" # (to an integer)
3. */Type* to "*/Pages*" # (to a name)



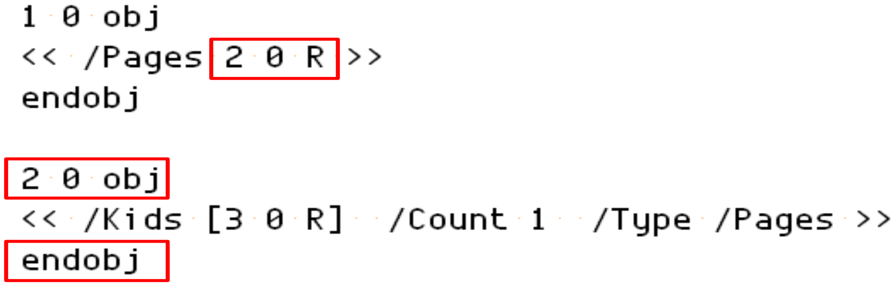
```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL²BÒ€„i,‰BH
–á‘š““‘¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Object reference

```
/Pages 2 0 R
```

is “equivalent” to

```
/Pages <<
    /Kids [3 0 R]
    /Count 1
    /Type /Pages
>>
```



```
1 0 obj
<< /Pages 2 0 R >>
endobj

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

and then ”3 0 R“ is a further reference…

Binary streams

parameters, filters...

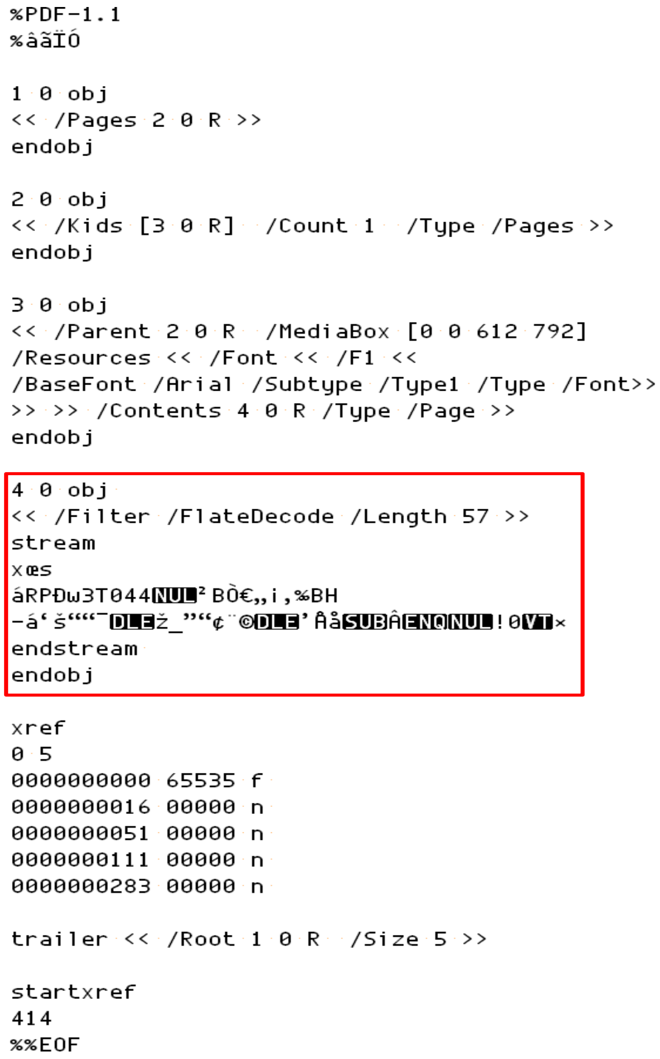
Streams

(Streams are only places where in PDF binary chars can appear -- other than in comments…)

Syntax:

1. usual obj declaration
2. stream params in dictionary
   (must include /Length !)
   (if encoded, includes /Filter !)
3. stream (keyword)
   + EOL character(s)
4. stream data
5. endstream (keyword)
   + EOL character(s)
6. usual endobj

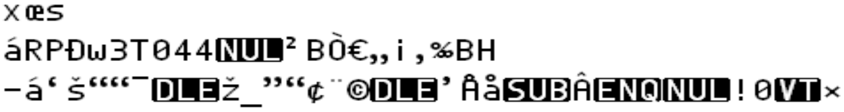
stream data is not interpreted (at object level)



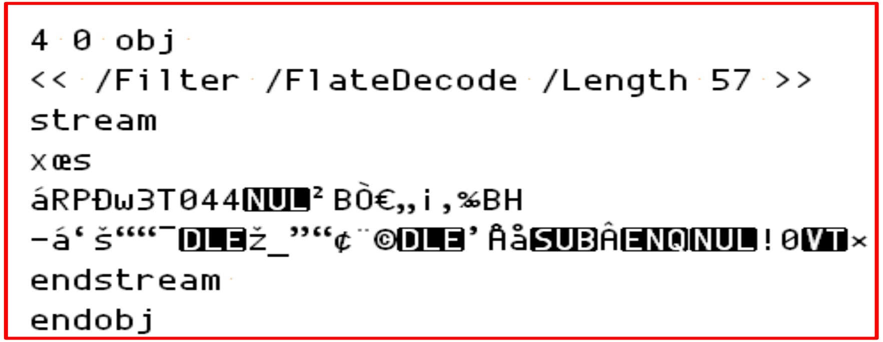
```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPÐw3T044NUL²BÒ€„i,‰BH
–á‘š““‘¯DLEž_””“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000111 00000 n
0000000283 00000 n

trailer << /Root 1 0 R  /Size 5 >>

startxref
414
%%EOF
```

Example

- stream parameters:
  - `/Filter = /FlateDecode`
  - `/Length = 57`
- stream content (binary):



```
xœs
áRPĐw3T044NUL² BÒ€„i,‰BH
–á‘š““““¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
```



```
4 0 obj
<< /Filter /FlateDecode /Length 57 >>
stream
xœs
áRPĐw3T044NUL² BÒ€„i,‰BH
–á‘š““““¯DLEž_”“¢¨©DLE’ÅåSUBÂENQNUL!0VT×
endstream
endobj
```

Binary streams

- can be stored with different encodings or compression schemes
  - `/Filter`
  - encodings/compressions can be cascaded
- content is decoded
  - after each filter

only the final (de-coded) data matters

What’s in a stream?

Typical contents of (filtered/encoded/binary) streams are:

- Embedded font files
- Images
- ICC profiles
- Page /Contents

PDF-1.5 and later: bundle "indirect objects" into streams: “`/Type /ObjStm`”

(Some stream contents may be "binary-as-original", without extra /Filter applied. Example: font files.)

Streams don’t enforce encodings

as long as the result is correct
once decoded by the filters

```
<< /Length 53 >>

stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
```

```
<< /Length 57
   /Filter /FlateDecode
>>

stream
xœs
áRPÐw3T044 ²BÒ€„¡,‰BH
-á‘š““¯ž_”“¢¨©’ÅåÂ !0×
endstream
```

the 2 streams above are equivalent -- they just use a different encoding
(Flate = ZIP compression)
(/FlateDecode = Use ZIP uncompression to unpack the stream)

```
<< /Length 170
   /Filter [
      /ASCIIHexDecode
      /FlateDecode] >>
stream
78 9C 73 0A E1 52 50 D0 77 33 54 30 34
34 00 B2 42 D2 80 84 A1 81 82 89 81 81
42 48 0A 90 AD E1 91 9A 93 93 AF 10 9E
5F 94 93 A2 A8 A9 10 92 C5 E5 1A C2 05
00 21 30 0B D7
endstream
```

```
<< /Length 57
   /Filter /FlateDecode
>>

stream
xœs
áRPÐw3T044 ²BÒ€„¡‚‰BH
-á‘š““¯ž_”“¢¨©’ÅåÂ !0×
endstream
```

/ASCIIHexDecode will decode ASCII Hex to binary, then Deflating will decompress the result

Exercise:
stream decoding

via mutool, pdftk, qpdf, podofouncompress, peepdf, pdf-parser.py, ...

Main filters

- `<none>`: direct raw binary stream in the file
- `/FlateDecode` : ZIP's deflate (de)compression

  → smaller
- `/ASCIIHexDecode:` turns hex <=> binary
  - `41 0A` ⇒ "`A\n`"

  → easy text editing (but binary is very common)

  `mutool` has a specific option for that

- `/ASCII85Decode:` hex <=> ASCII base 85

Other filters

Images

- `/DCTDecode` to store JPEG **files** directly
  - not just the data, even the header!
  - may work for any data, including JavaScript
- `/LZWDecode`, `/CCITTFaxDecode`, `/JBIG2Decode`, `/JPXDecode`

Encryption

- `/Crypt`
  - RC4 or AES

Let’s put it all together

how is the file actually parsed?

Parsing 1/7

1. Signature is checked

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Parsing 2/7

2. %%EOF is located

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Parsing 3/7

3. `xref` is located via `startxref`

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Parsing 4/7

4. `xref` gives the byte offset adresses for each object

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Parsing 5/7

5. `trailer` is parsed
   → gives `/Root` object

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Parsing 6/7

6. objects are parsed
   a. /Root object contains /Pages
   b. /Pages contains page array
      - /Kids
   c. each /Page has:
      - size: /MediaBox (*)
      - /Contents
        - as stream object
      - /Resources
        - defines the /Font dictionary

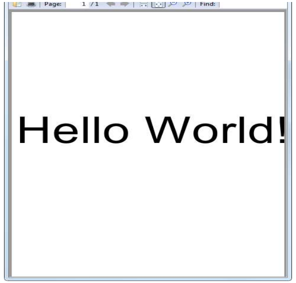
(*) If all /MediaBox sizes are identical, can also be set in /Pages obj and "inherited" in individual /Page objects without setting them there.

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Parsing 7/7

7. the page is rendered

a.	`BT`	BeginText
b.	`<name> <size> Tf`	select font
c.	`<x> <y> Td`	move cursor
d.	`<string> Tj`	display string
e.	`ET`	EndText



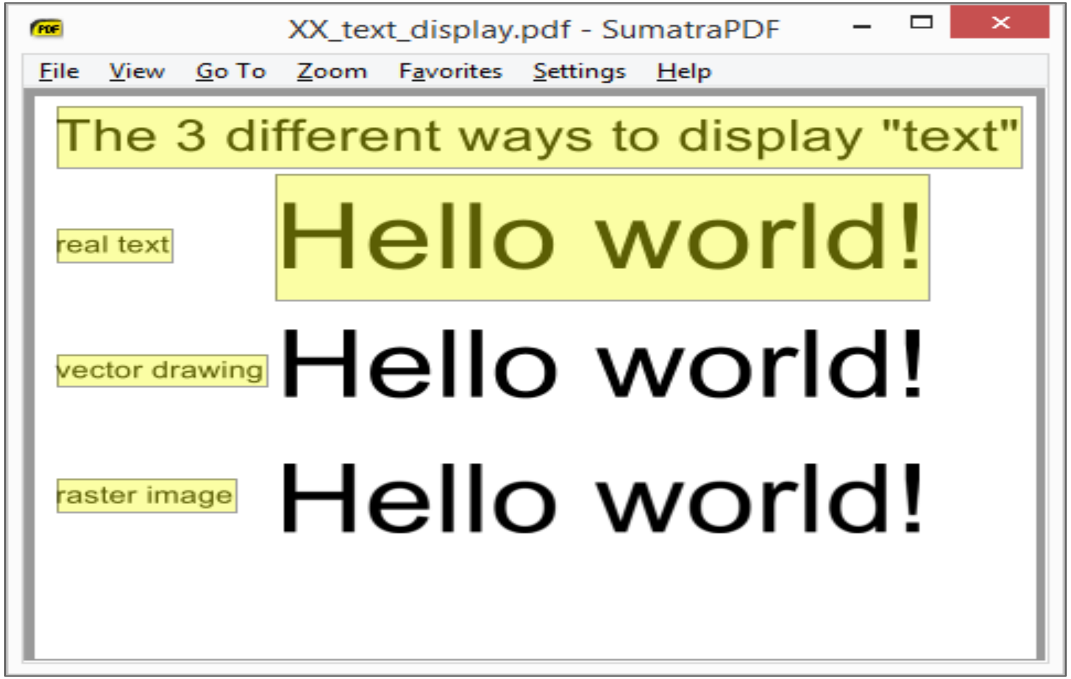
```
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
```

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 53 >>
stream
BT
 /F1 110 Tf
 10 400 Td
 (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000051 00000 n
0000000109 00000 n
0000000281 00000 n

trailer << /Root 1 0 R /Size 5 >>

startxref
384
%%EOF
```

Page contents

3 basic types

- Real Text
- Raster Images
- Vector Drawing Elements



The 3 different ways to display "text"

real text Hello world!

vector drawing Hello world!

raster image Hello world!

Exercise:
Text representations

‘text’ / drawing / image

In practice

- that was the ‘strict’ minimum
- a typical PDF embeds more information
  - fonts
  - font encodings
  - metadata
  - raster images
  - ICC profiles
  - …

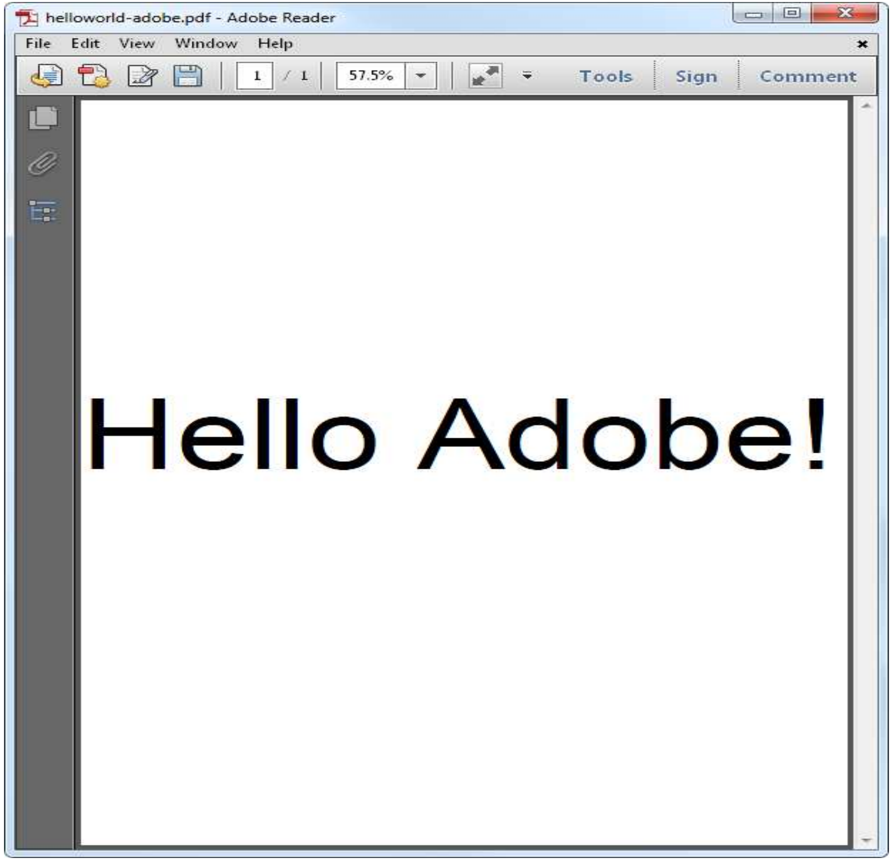
a generated *Hello World* typically weights >10 KB

In practice - in the malware world

- Most readers accept malformed files
  - many elements may be missing:
    - EOF, startxref, xref, /Length, endobj, endstream
    - /MediaBox /Font
- Each reader has its own weirdness
  - see my “Schizophrens” talks and PoCs

- …so much for the so-called “standard”

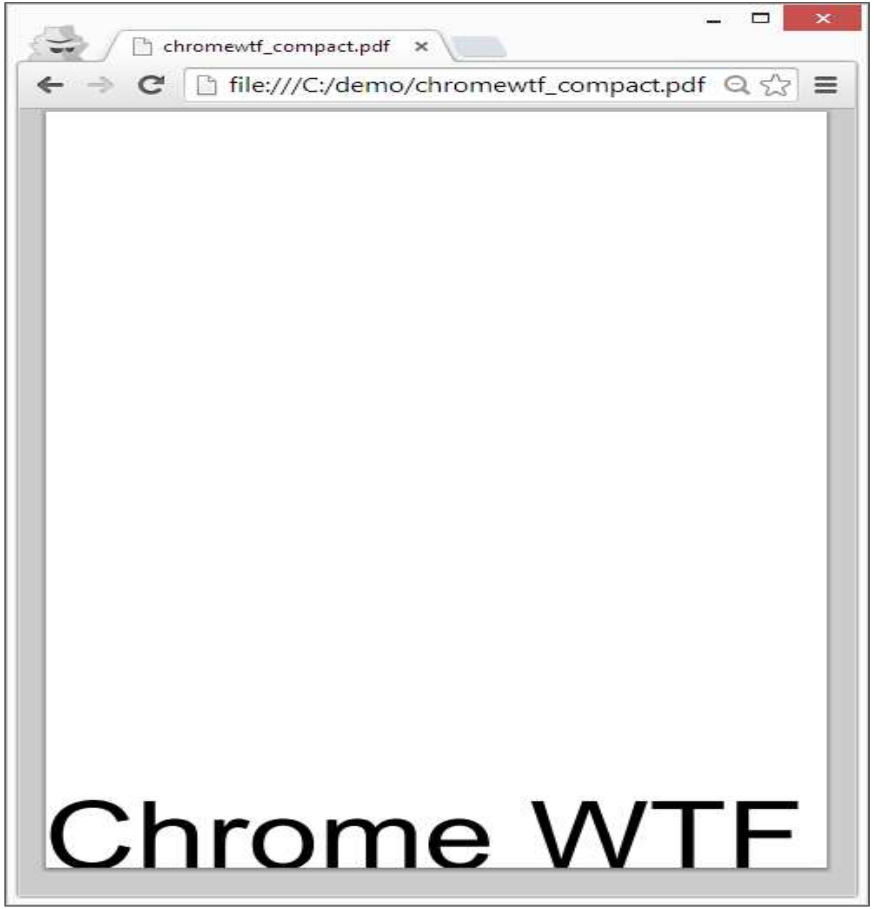
```
%PDF-\0
1 0 obj<</Kids[<</Parent
1 0 R/Contents[2 0 R]>>]
/Resources<<>>   >>
2 0 obj<<>>
stream\n
BT/F1 105 Tf 0 400 Td
(Hello Adobe!)Tj ET
endstream\n
endobj\n
trailer<<
/Root<</Pages 1 0 R>>>>
```


helloworld-adobe.pdf - Adobe Reader
File Edit View Window Help
1 / 1
57.5%
Tools
Sign
Comment
Hello Adobe!

A "Hello World" for Adobe, in 179 bytes

(demo with Adobe Reader XI [works] and Acrobat Pro [crashes] on Mac)

```
%PDF\n
1 0 obj<<
/W[[]1/]
/Root 1 0 R
/Pages
<</Kids[<</Contents<<>>
stream\n
BT{99 Tf
(Chrome WTF)'
endstream
>>]>>>>
stream\n
endobj
%startxref%1234567
```



“Chrome WTF”, in a funky tweet

Reminders on syntax

Basic ones

```
% comment until line end
```

```
<string in hex>
(standard string)
```

Equivalent examples:

```
(Hello Loop)
<48 65 6C 6C 6F 20 4C 6F 6F 70>
<4 86 56 C6C6 F20 4 C 6 F6F 7>
```

(Spec says: if odd no. of characters, 'hex' string should be padded with 0.)

Check your viewers now...

Dictionaries (key/value pairs)

```
<< [/name <value>]* >>          # << >> are dictionary delimiters
                                # [ & ] not part of syntax -- here to denote "pair"
<< /Size 637 >>                 # sets /Size to 637
```

Ex:

```
<</Creator(Ange Albertini)>>    # No whitespace: Why? Optional!
                                #   (other delimiters already present)
```

sets /Creator to "Ange Albertini"

(/name must comply to syntax rules for "Name tokens”)
(<value> can be anything -- even another dictionary, or an array)
(order of key/value pairs is NOT significant!)

Arrays (ordered list of elements)

```
[ <element>* ]                    # [ ] are array delimiters!
```

Ex

```
[0 0 612 792]                     # array of 4 elements
```

(`<element>` can be anything -- even another array or dictionary!)

(in arrays the order of elements is significant!)

Binary streams

absolutely ***anything*** between

    stream
    endstream

inside a dedicated object

with stream encoding parameters
in the object’s dictionary

Backward syntax

- Operators and operands in page contents
- Because PDF inherited some elements from PostScript

References

`1 0 R` : refers to object 1 generation 0

refers to what's between

```
1 0 obj
endobj
```

Example:

`[ 1 0 R ]` is an array of one element

element is reference to object "1 0"

Text in page contents

inside a (possibly encoded) stream

- `/F1 110 Tf` : use text font F1 with size 110
- `10 400 Td` : put current point to x=10, y=400
- `(Hello World) Tj` : print Hello World

Walkthrough

helloworld_pretty.pdf

```
%PDF-1.1
%âãÏÓ

1 0 obj <<
  /Pages 2 0 R
>>
endobj

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

```
3 0 obj <<
  /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Resources <<
    /Font <<
      /F1 <<
        /Type /Font
        /Subtype /Type1
        /BaseFont /Arial
      >>
    >>
  >>
  /Contents 4 0 R
>>
endobj
```

```
4 0 obj
<< /Length 51 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj
```

```
xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000053 00000 n
0000000117 00000 n
0000000345 00000 n

trailer <<
  /Root 1 0 R
  /Size 5
>>

startxref
446
%%EOF
```

Embedding an image in a PDF

/Page’s /Resources

```
/Resources
<<
  /XObject <</Im0  5 0 R>>
  ..
>>
```

/Page’s /Contents object stream:

```
q
<width> 0 0 <height> 0 0 cm
/Im0 Do
Q
```

"<width> 0 0 <height> 0 0" : defines the operant, a matrix.
"cm" : is the *"concatenate matrix to current transformation matrix"*-operator.
"Do" : is the operator that calls invokes the rendering of the XObject.
"q" and **"Q"** : are the operators which save and restore the graphics state.

Changing the zeros in the matrix to other numerical values can rotate, skew, scale, translate (and any combination of thereof) the image.

Image object:

```
5 0 obj
<<
  /Type /XObject
  /Subtype /Image
  /Width <width>
  /Height <height>
  /BitsPerComponent 8
  /ColorSpace /DeviceRGB
  /Filter [
    /ASCIIHexDecode
    /DCTDecode   % JPEG compression
  ]
>>
stream
<IMAGE DATA>
endstream
endobj
```

Images = independent objects

They can be dumped by trivial parsing
(<4Kb images can be inlined)

At this point...

We’ve covered the basics of:

- file structure
- objects relation
- file parsing
- page rendering

→ enough to start playing with PDF internals!

How to start using the PDF spec

Link to free/gratis version:

- http://acroeng.adobe.com/PDFReference/ISO32000/PDF32000-Adobe.pdf
  (official specs -- meanwhile belong to ISO ⇒ not free -- costs 198 CHF to buy)

Important starting chapters:

- **Understand `/Contents` stream:**
  Annex A (Operator Summary); also names equivalent PostScript operators

- **Understand other "normative" specs:**
  Chapter 3 (Normative References); lists ~80 more external documents about fonts, encryption, hashes, Unicode, images, compression schemes....

- **Understand text/font encodings:**
  Annex D (Char sets and Encodings)

Hiding/revealing elements

Part II / II

text can (most of the times) be copied

images can be extracted

the “Select All” trick often works,
but not always

even if “Select All” does *not* work, secrets *may* still be recovered

(incrementally updated PDFs!)

hiding/revealing parts of the PDF document

from this point on:

not hiding data in a PDF file (stego)

nothing reader-specific (schizo)

Isn’t copy/paste enough?

- why not editing the file itself ?
  and restoring the secrets perfectly?

want to hide something?

- create your own methods!

Easy PDF editing

1. decompress streams
   - PDFTk , qpdf
   - optional: use ASCIIHex to get an ASCII-only file
2. open in text editor
3. view results via Sumatra

overwrite, or comment (don’t delete)

⇒ no offset to adjust

```
D:\> pdftk "GoogleDoc.pdf" output uncompressed.pdf uncompress
```

```
D:\> qpdf --qdf --object-streams=disable "OpenOffice.pdf" uncompressed.pdf
```

```
D:\> mutool clean -d -i -f "GhostScript.pdf" uncompressed.pdf
```

protected.pdf
unprotected.pdf

Remove PDF "protections"

- PDF feature to prevent printing or copy/paste
- If you can view it, it means it **is** decrypted !
  - it just means that the **user** password is empty
- Permission for copy-paste/printing is just a flag
  - the **owner** password “prevents” to change it

⇒ remove it alltogether:

```
D:\> qpdf --decrypt protected.pdf unprotected.pdf
```

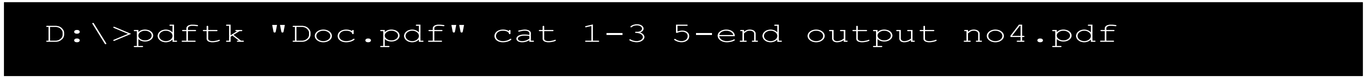
Reminder

technically speaking, a PDF page is:

1. a stream object
2. as the `/Contents` of a `/Type /Page` object
3. in the `/Kids` array of a `/Type /Pages` object
4. as the value of `/Pages` in root object
5. as the value of `/Root` in the trailer

and text on the page are simple *`(string) Tj`*
or *`<hexvalues> Tj`* (or *`TJ`*)

Erasing a page with a tool

- tools such as PDFtk can operate on pages
  - 

```
D:\>pdftk "Doc.pdf" cat 1-3 5-end output no4.pdf
```

but:

- they don’t erase pages!
  - they extract the other pages and write a new file

→ the whole code for page is lost...

...but its image contents (as objects) may still be present + extractable!! (Bug or feature of `pdftk` ?!)

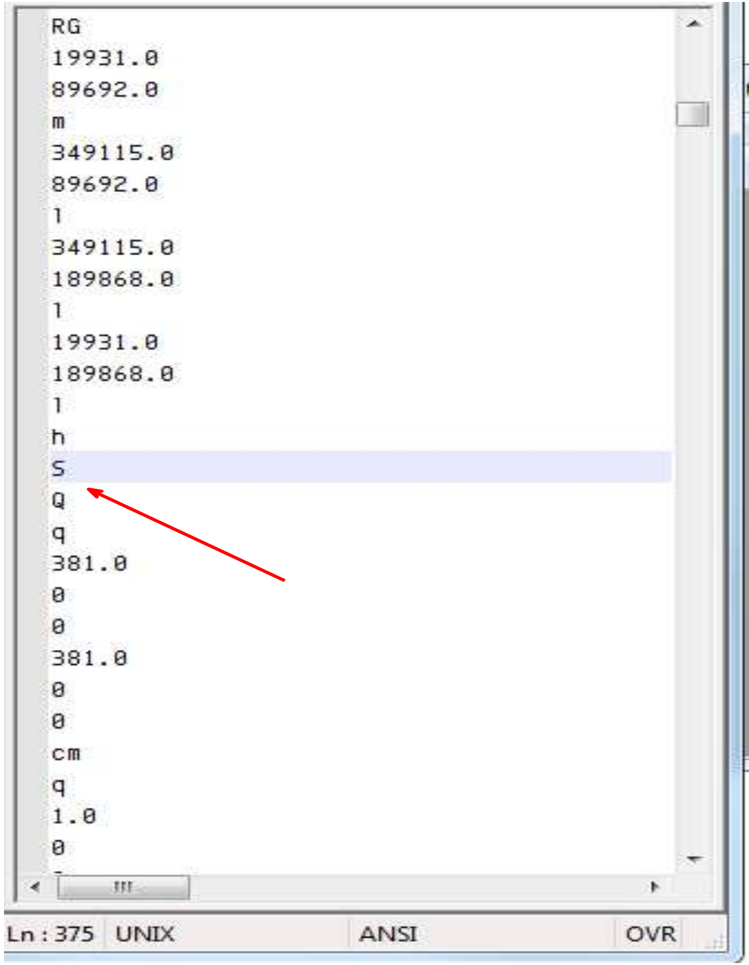
Erase overlapping element?

- remove paint/text operators from binary stream

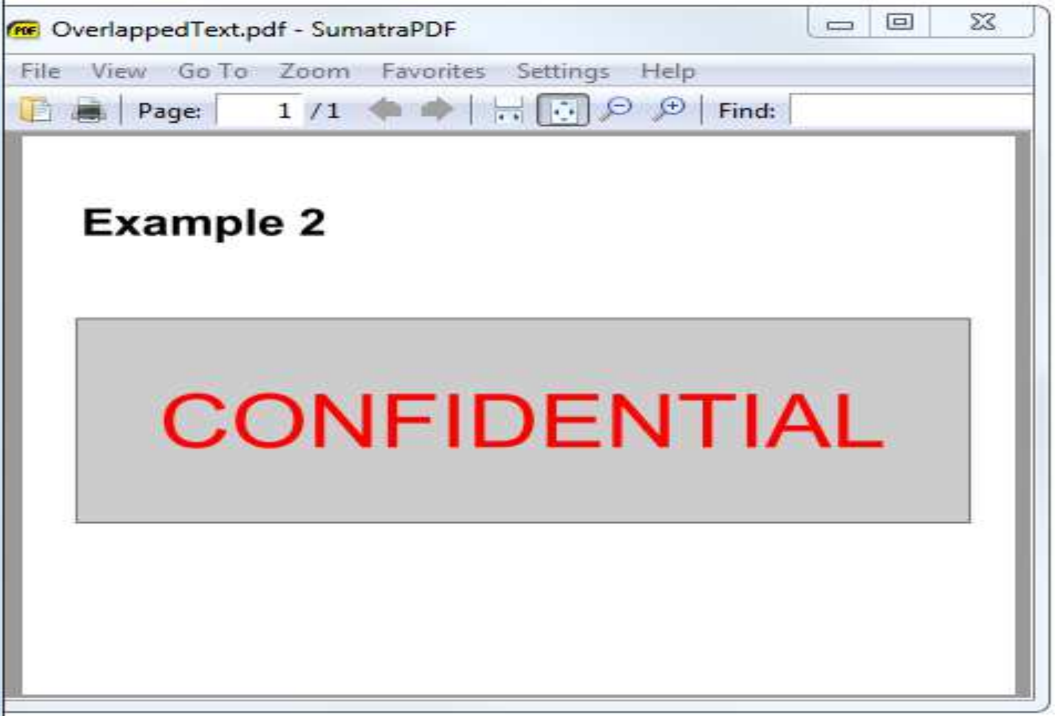
Hints:

Content drawing stream operators operate in their order of appearance inside the stream.
Overlapping elements more likely at the end of the stream, as they were likely added last.

Example:
manually remove overlapping elements



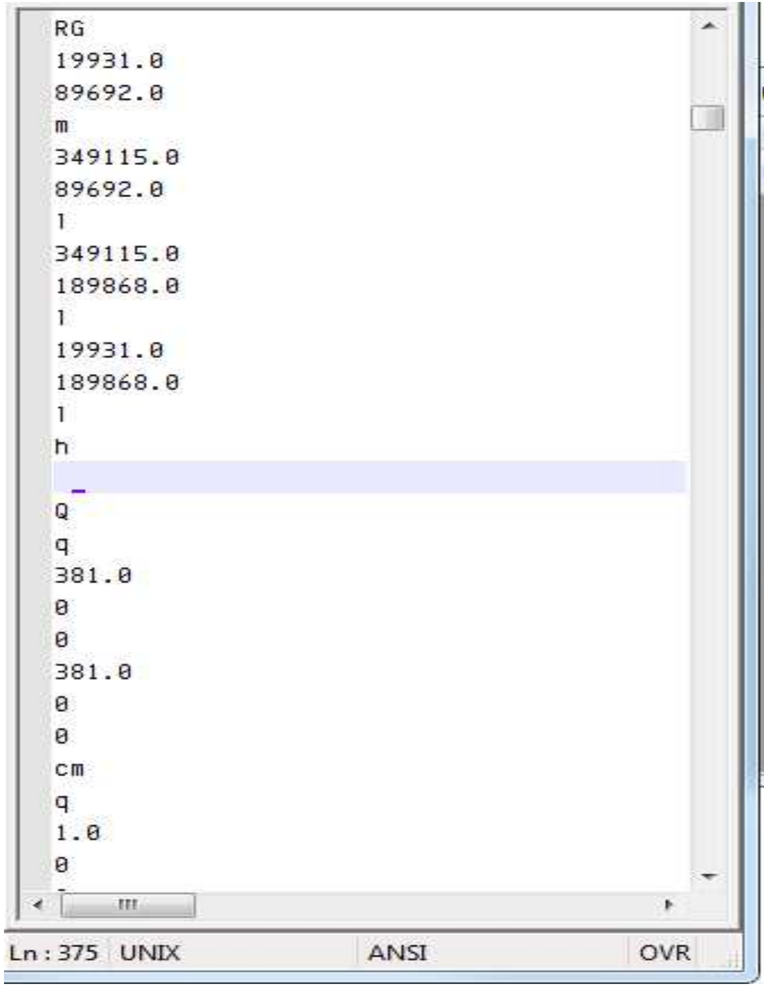
```
RG
19931.0
89692.0
m
349115.0
89692.0
l
349115.0
189868.0
l
19931.0
189868.0
l
h
S
Q
q
381.0
0
0
381.0
0
0
cm
q
1.0
0
```



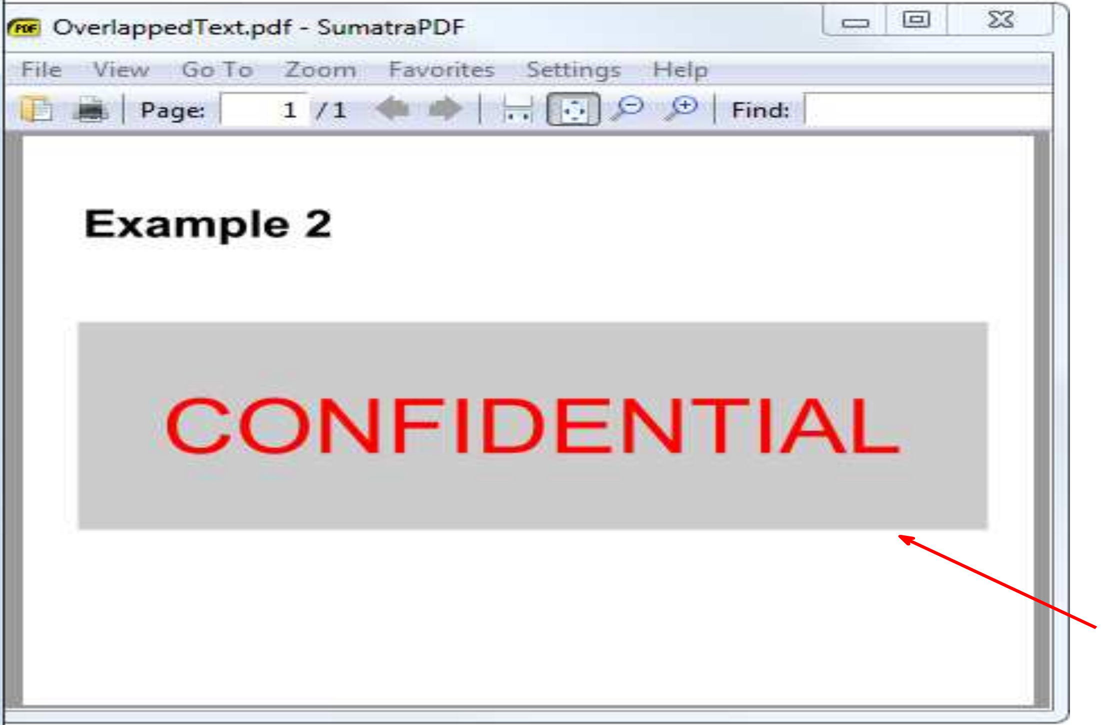
Example 2

CONFIDENTIAL

take the uncompressed PDF
locate the `/Contents` stream object
locate the S (**S**troke path)
(you can search for **\nS\n**)



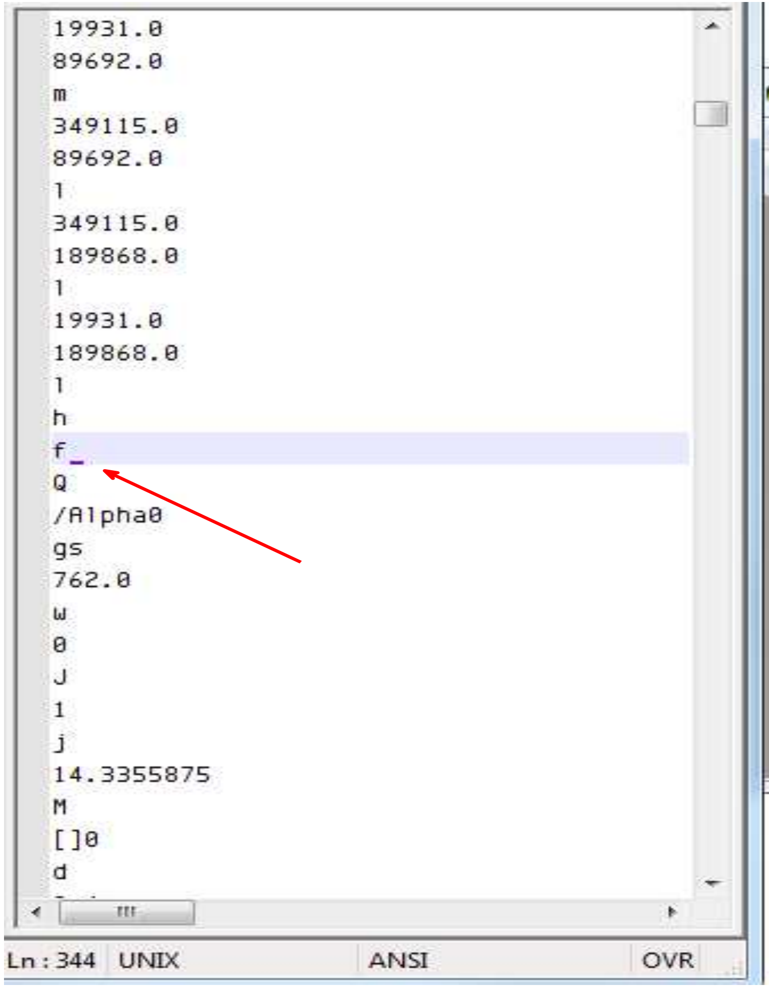
```
RG
19931.0
89692.0
m
349115.0
89692.0
l
349115.0
189868.0
l
19931.0
189868.0
l
h
 
Q
q
381.0
0
0
381.0
0
0
cm
q
1.0
0
```



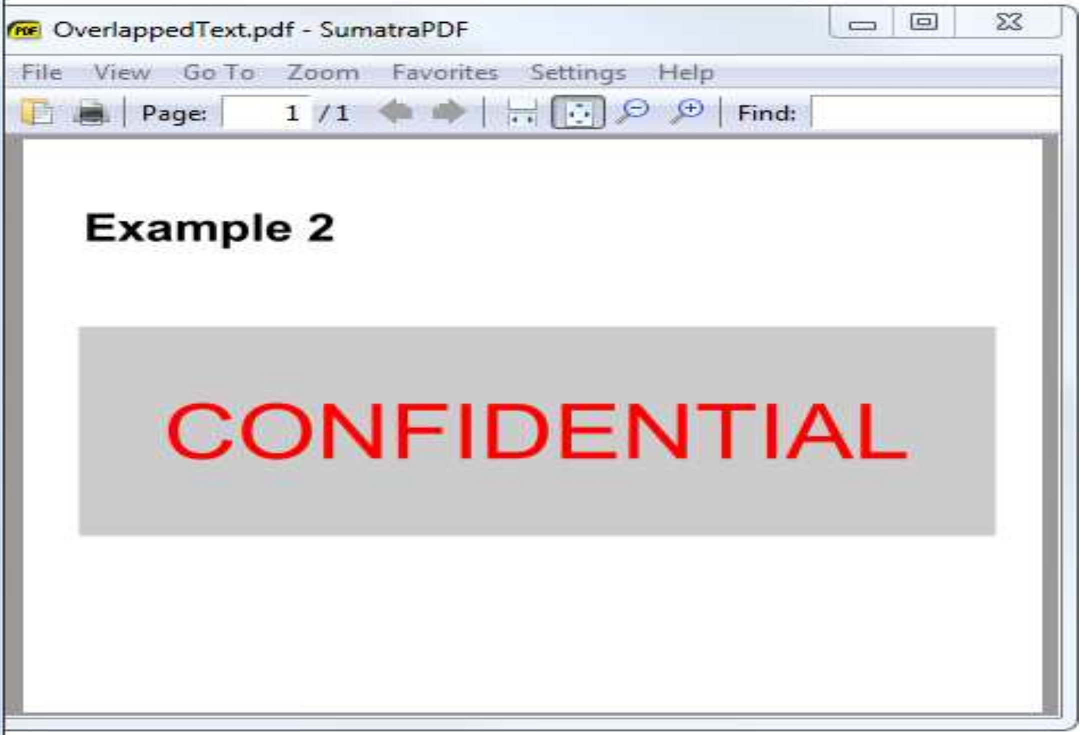
Example 2

CONFIDENTIAL

overwrite the S with a space
⇒ no more black border



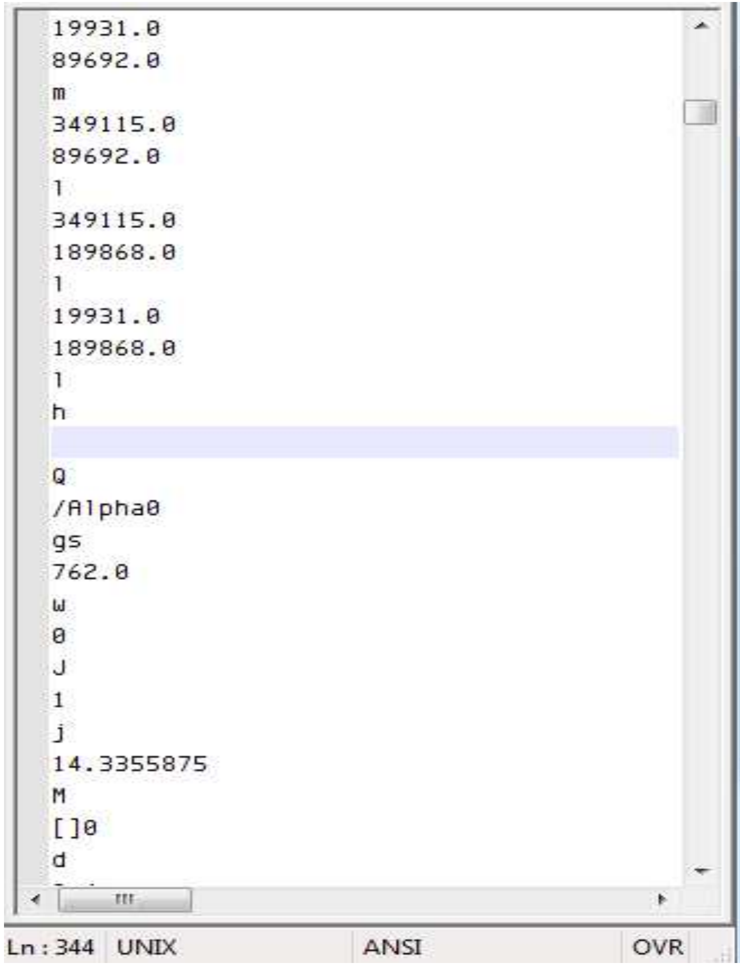
```
19931.0
89692.0
m
349115.0
89692.0
l
349115.0
189868.0
l
19931.0
189868.0
l
h
f_
Q
/Alpha0
gs
762.0
w
0
J
1
j
14.3355875
M
[]0
d
```



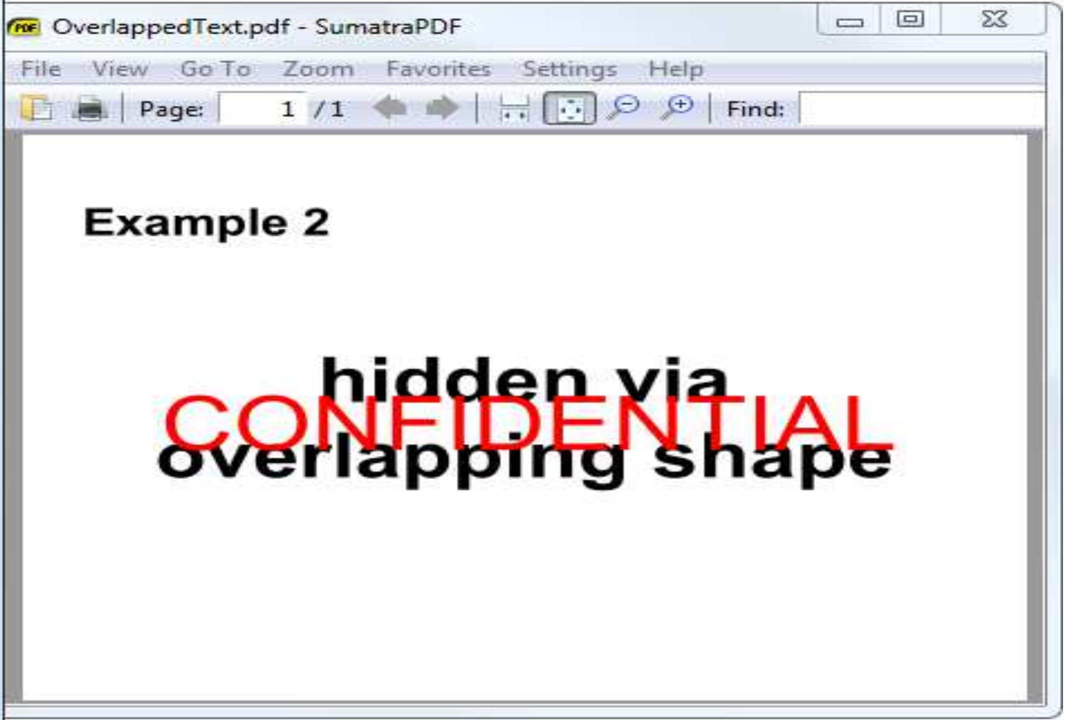
Example 2

CONFIDENTIAL

locate the f (path **F**illing)
overwrite with space too...



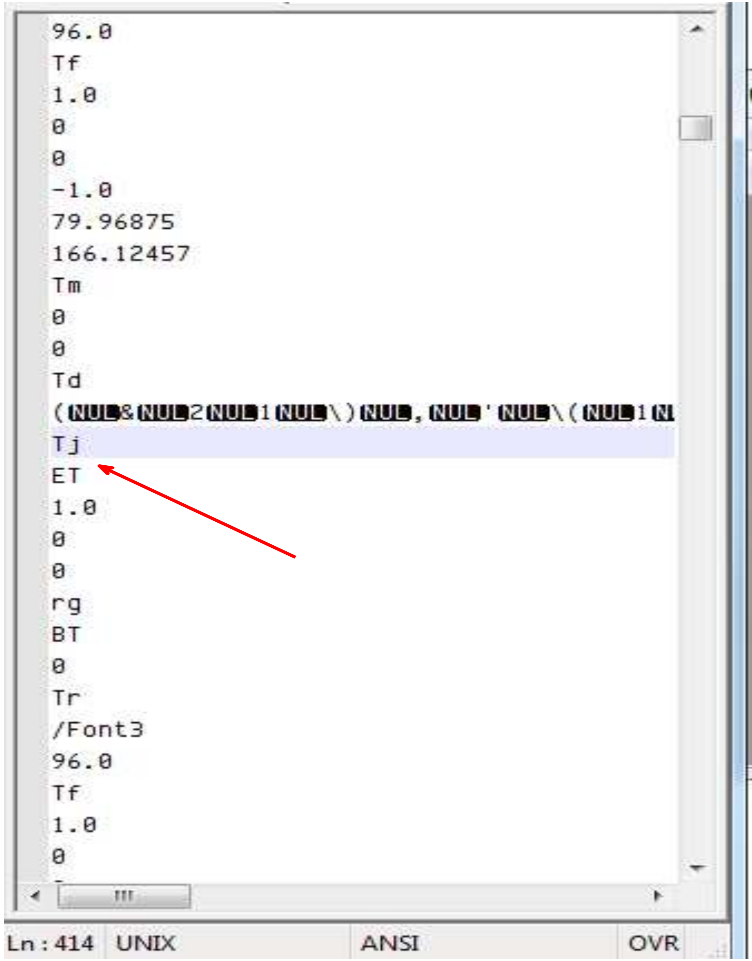
```
19931.0
89692.0
m
349115.0
89692.0
l
349115.0
189868.0
l
19931.0
189868.0
l
h

Q
/Alpha0
gs
762.0
w
0
J
1
j
14.3355875
M
[]0
d
```



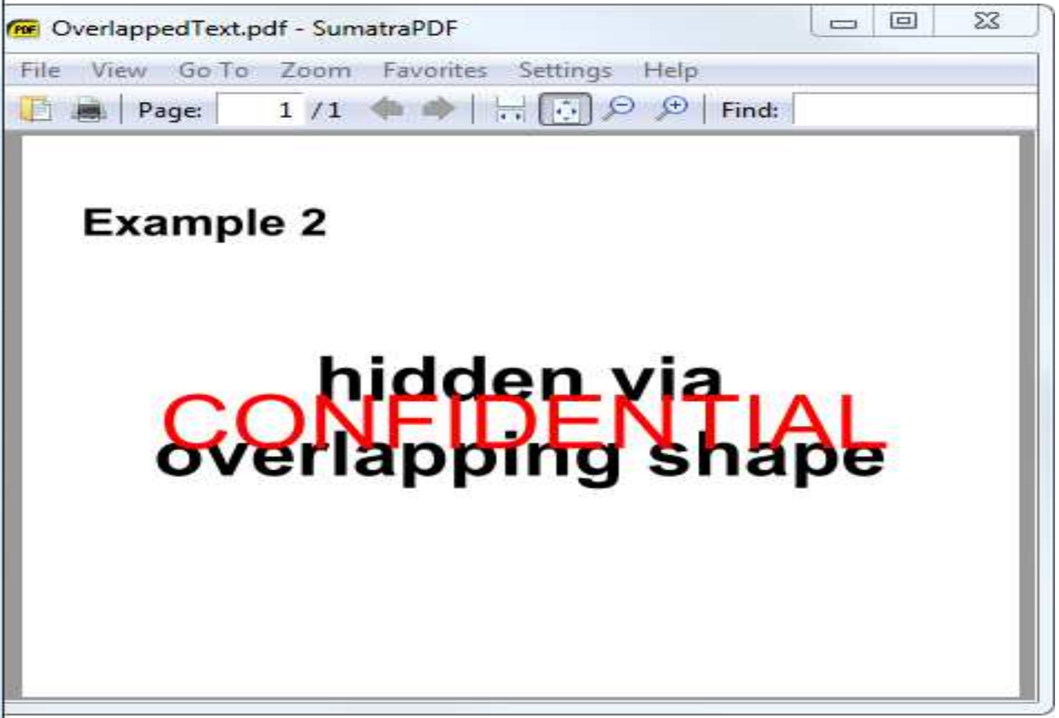
Example 2

hidden via
CONFIDENTIAL
overlapping shape

⇒ no more gray surface



```
96.0
Tf
1.0
0
0
-1.0
79.96875
166.12457
Tm
0
0
Td
(NUL&NUL2NUL1NUL\)NUL,NUL'NUL\(NUL1N
Tj
ET
1.0
0
0
rg
BT
0
Tr
/Font3
96.0
Tf
1.0
0
```



Example 2

hidden via

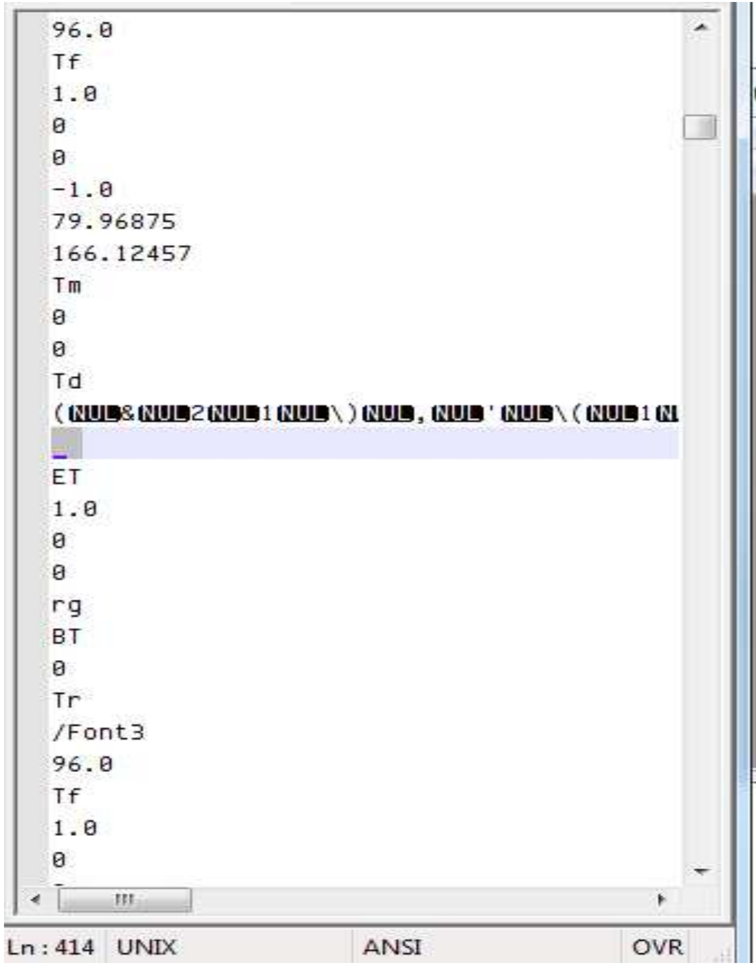
CONFIDENTIAL

overlapping shape

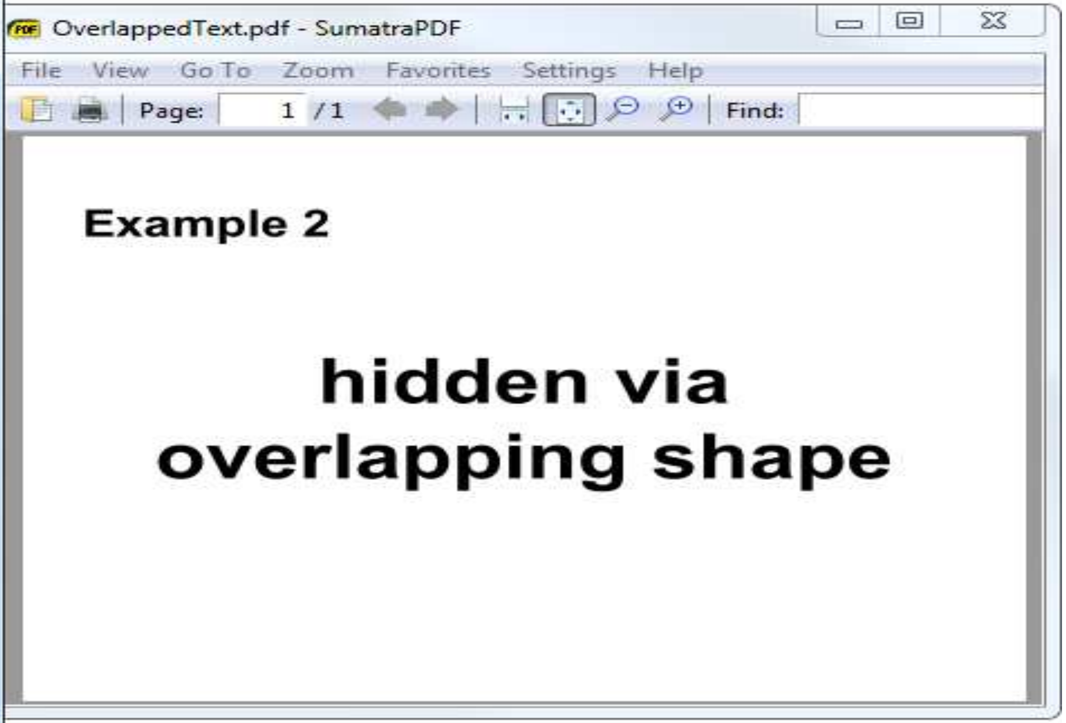
and the “obvious” `Tj` after the string `(...)`

Note: the chars in this PDF are different to letters in rendered text, due to the font mapping:

&→C, 2→O, 1→N...



```
96.0
Tf
1.0
0
0
-1.0
79.96875
166.12457
Tm
0
0
Td
(NUL&NUL2NUL1NUL\)NUL,NUL'NUL\(NUL1NU

ET
1.0
0
0
rg
BT
0
Tr
/Font3
96.0
Tf
1.0
0
```



Example 2

hidden via
overlapping shape

→ no more hidden elements!

bonus: the operation can be easily automated!
(on all pages, etc...)

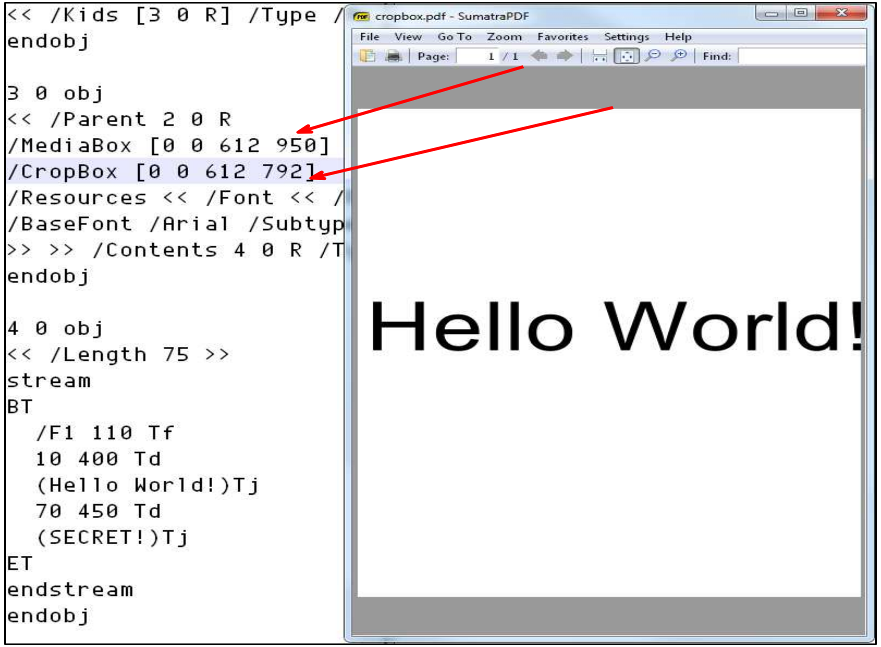
cropbox..pdf

Page size (MediaBox/CropBox) effects

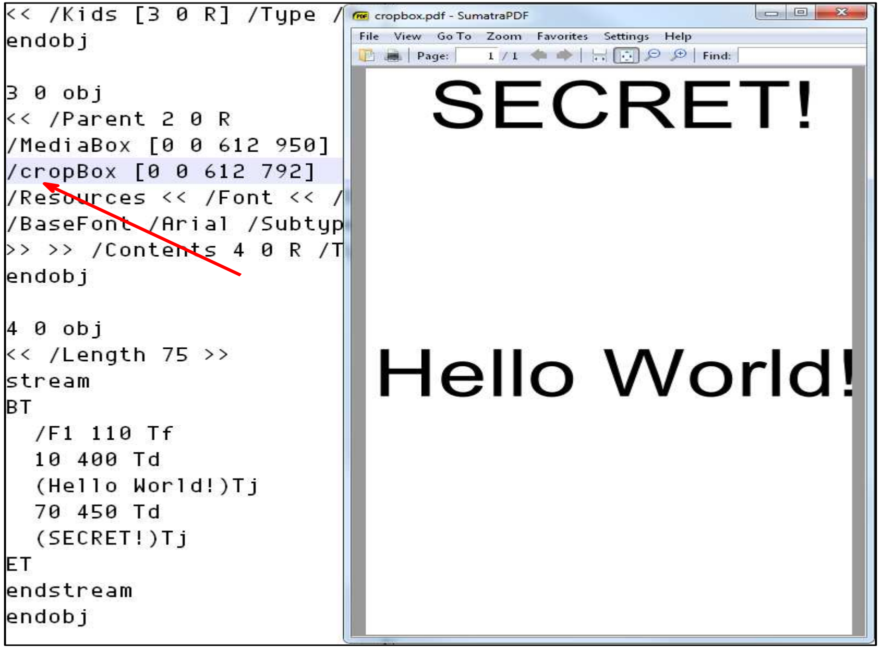
- a page isn’t just a /MediaBox :(
    - PDF is not so simple!
        - CropBox/BleedBox/TrimBox/ArtBox/...

- What you *see* is `/CropBox`
    - Copy/Paste and (some) `pdftotext` respect that

⇒ what is in `MediaBox` (but not `CropBox)  is not` extracted by tools or copy/paste (most times -- some tools/versions do it)

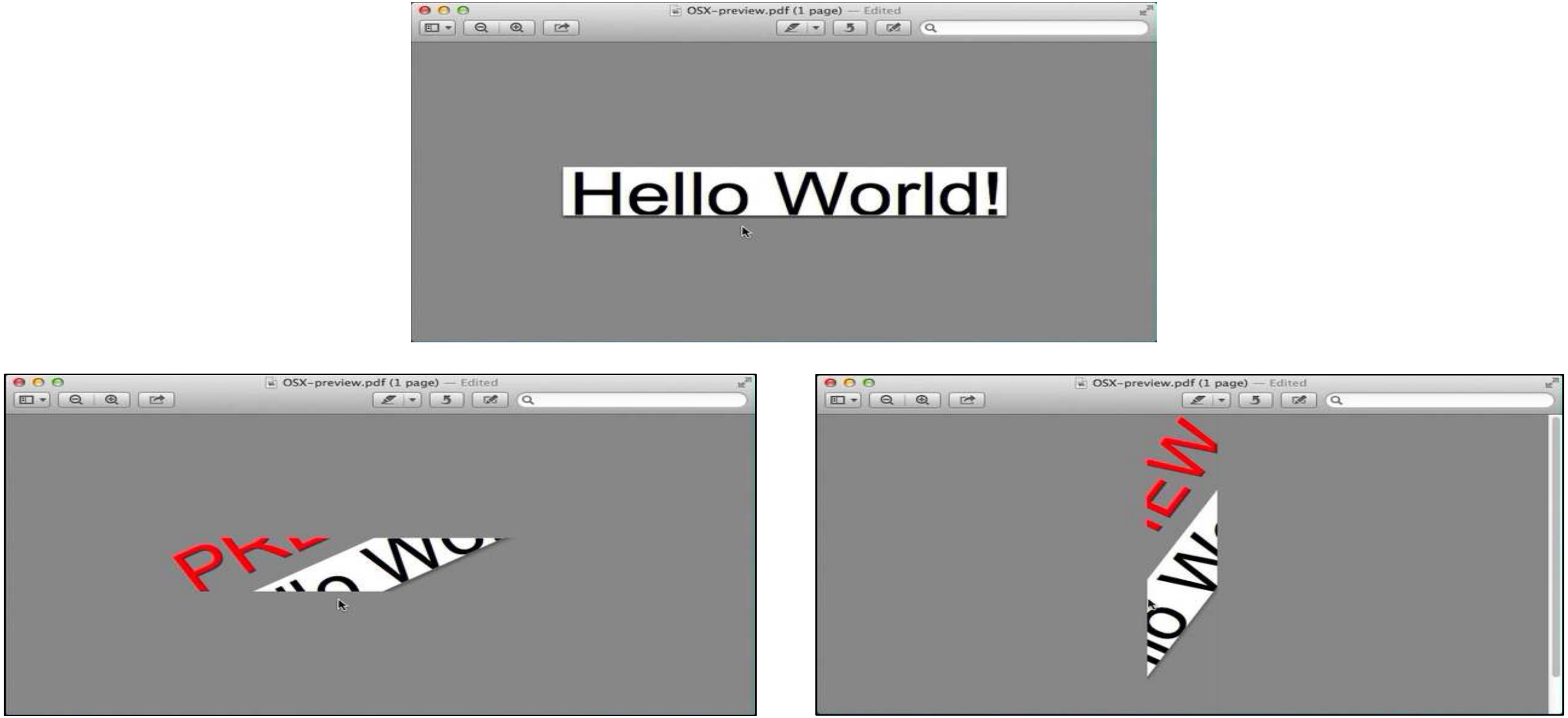


```
<< /Kids [3 0 R] /Type /
endobj

3 0 obj
<< /Parent 2 0 R
/MediaBox [0 0 612 950]
/CropBox [0 0 612 792]
/Resources << /Font << /
/BaseFont /Arial /Subtyp
>> >> /Contents 4 0 R /T
endobj

4 0 obj
<< /Length 75 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!)Tj
  70 450 Td
  (SECRET!)Tj
ET
endstream
endobj
```



```
<< /Kids [3 0 R] /Type /
endobj

3 0 obj
<< /Parent 2 0 R
/MediaBox [0 0 612 950]
/cropBox [0 0 612 792]
/Resources << /Font << /
/BaseFont /Arial /Subtyp
>> >> /Contents 4 0 R /T
endobj

4 0 obj
<< /Length 75 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!)Tj
  70 450 Td
  (SECRET!)Tj
ET
endstream
endobj
```

"mis"-spell and disable /CropBox to see the full contents


OSX-preview.pdf (1 page) — Edited
Hello World!
OSX-preview.pdf (1 page) — Edited
PR
llo Wo
OSX-preview.pdf (1 page) — Edited
EW
llo W

OS-X actually does use a /CropBox when you copy/paste out of a PDF, but full page content is still there. You can see full original content by rotating the page. Or just mis-spell "/cropBox" once more to expose the secret again...

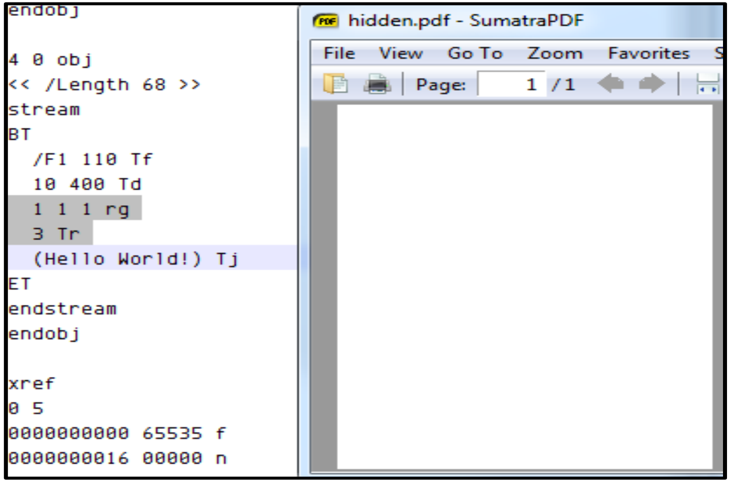
hidden.pdf

Hidden text

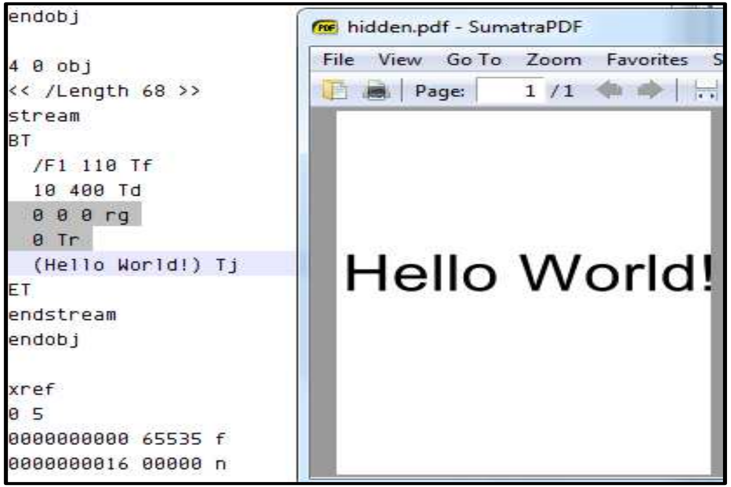
- White color
  - ○ 1 1 1 rg (filling’s color)

- Text rendering mode ('Tr')
  - ○ 3 Tr = invisible
    - ■ OCRs use it to store text, overlayed over scanned image...

(Both of the above work independently from each other. Both allow to still copy'n'paste text...)



```
endobj

4 0 obj
<< /Length 68 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  1 1 1 rg
  3 Tr
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
```



```
endobj

4 0 obj
<< /Length 68 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  0 0 0 rg
  0 Tr
  (Hello World!) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
```

A more ‘deniable’ hiding?

Altering `/Kids` or the page’s `/Contents` works.

But there is another elegant solution:
"incremental updates"

PDF incremental updates

- Not commonly used *on purpose*
  - ...but required for signing
- Supported by readers
- Acrobat incrementally updates after (most) changes when clicking *"Save"* (to avoid this, use *"Save As..."* !)

The concept:

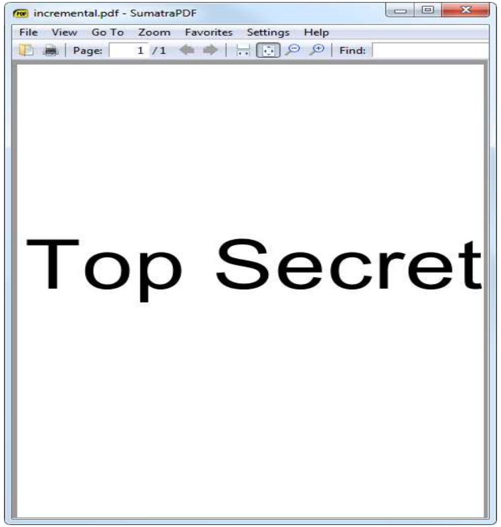
...add another set of objects, xref, trailer, ...

...to update the objects' hierarchy

...while leaving all previous objects in place.

Example

a confidential object
with a secret stream object 4
to be hidden


incremental.pdf - SumatraPDF
File View Go To Zoom Favorites Settings Help
Page: 1 /1
Find:

Top Secret

```
%PDF-1.1
%âãÏÓ

1 0 obj
<< /Pages 2 0 R >>
endobj

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

3 0 obj
<< /Parent 2 0 R  /MediaBox [0 0 612 792]
/Resources << /Font << /F1 <<
/BaseFont /Arial /Subtype /Type1 /Type /Font>>
>> >> /Contents 4 0 R /Type /Page >>
endobj

4 0 obj
<< /Length 50 >>
stream
BT
  /F1 120 Tf
  10 400 Td
  (Top Secret) Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000016 00000 n
0000000052 00000 n
0000000110 00000 n
0000000282 00000 n

trailer << /Size 5 /Root 1 0 R >>

startxref
385
%%EOF
```

New /Contents

append a new object 4

```
4 0 obj
<< /Length 52 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!) Tj
ET
endstream
endobj
```

Extra xref

append a new xref
that references it

```
xref
0 1
0000000000 65535 f
4 1
0000000551 00000 n
```

Extra trailer 1/2

- same `/Size` & `/Root`
- gives byte offset to previous **xref** via `/Prev` (not to previous trailer)

```
trailer <<
    /Size 5
    /Root 1 0 R
    /Prev 385
>>
```

Extra trailer 2/2

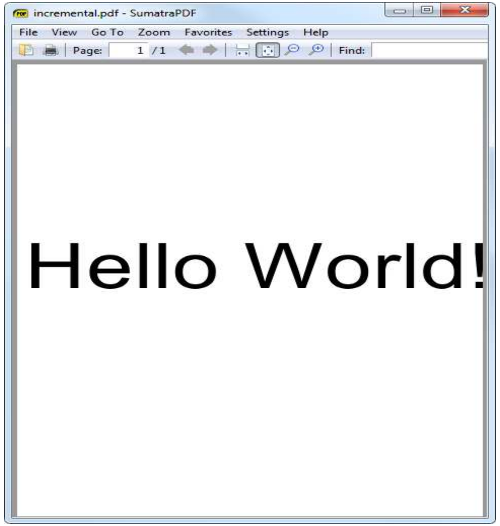
points to the new **xref**

```
startxref
654
%%EOF
```

Result

⇒ different content !

restore content by deleting everything after the first %%EOF:


incremental.pdf - SumatraPDF
File View Go To Zoom Favorites Settings Help
Page: 1 / 1
Find:

Hello World!

Incremental update to hide page

use the same trick

to override /Type /Pages

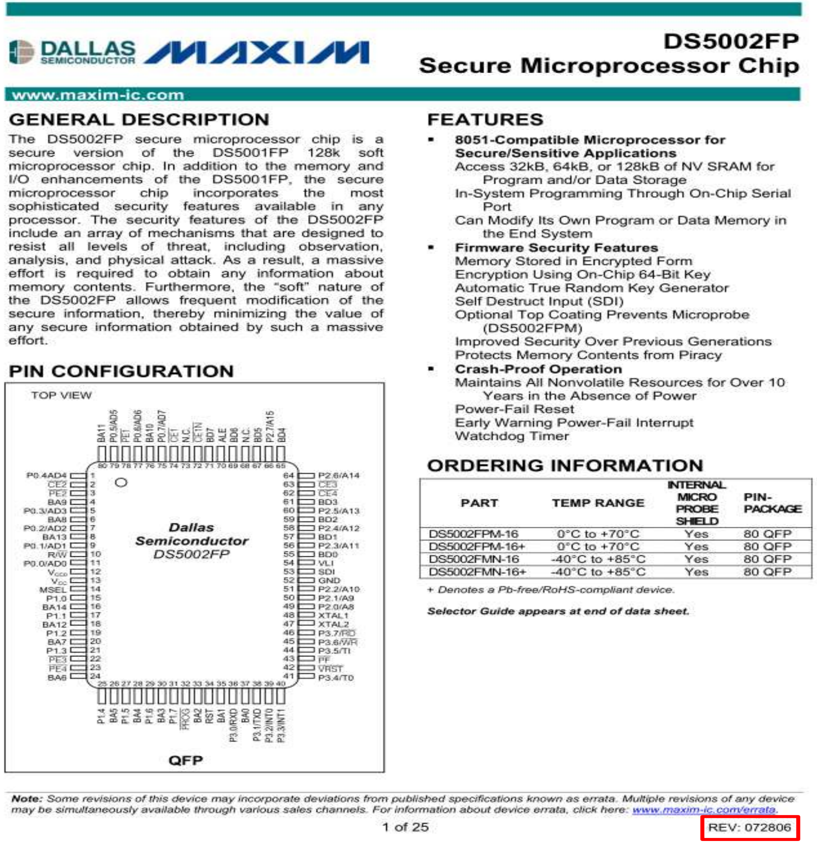
```
...
%%EOF

1 0 obj
<<
/Type /Pages
/Kids [ 6 0 R 21 0 R]
/Count 2
>>
endobj

xref
0 1
0000000000 65535 f
1 1
0000118783 00000 n

trailer << /Size 41 /Root 4
0 R /Prev 117882 >>

startxref
118849
%%EOF
```

Actual accidental leaks in the wild ?

Of course!

In any PDF with `/Prev` in the `trailer`:

- restore each intermediate version...
- ...by truncating after each `%%EOF`, one by one



DALLAS SEMICONDUCTOR MAXIM

DS5002FP Secure Microprocessor Chip

www.maxim-ic.com

GENERAL DESCRIPTION

The DS5002FP secure microprocessor chip is a secure version of the DS5001FP 128k soft microprocessor chip. In addition to the memory and I/O enhancements of the DS5001FP, the secure microprocessor chip incorporates the most sophisticated security features available in any processor. The security features of the DS5002FP include an array of mechanisms that are designed to resist all levels of threat, including observation, analysis, and physical attack. As a result, a massive effort is required to obtain any information about memory contents. Furthermore, the "soft" nature of the DS5002FP allows frequent modification of the secure information, thereby minimizing the value of any secure information obtained by such a massive effort.

PIN CONFIGURATION

FEATURES

- **8051-Compatible Microprocessor for Secure/Sensitive Applications**
  Access 32kB, 64kB, or 128kB of NV SRAM for Program and/or Data Storage
  In-System Programming Through On-Chip Serial Port
  Can Modify Its Own Program or Data Memory in the End System
- **Firmware Security Features**
  Memory Stored in Encrypted Form
  Encryption Using On-Chip 64-Bit Key
  Automatic True Random Key Generator
  Self Destruct Input (SDI)
  Optional Top Coating Prevents Microprobe (DS5002FPM)
  Improved Security Over Previous Generations
  Protects Memory Contents from Piracy
- **Crash-Proof Operation**
  Maintains All Nonvolatile Resources for Over 10 Years in the Absence of Power
  Power-Fail Reset
  Early Warning Power-Fail Interrupt
  Watchdog Timer

ORDERING INFORMATION

PART	TEMP RANGE	INTERNAL MICRO PROBE SHIELD	PIN-PACKAGE
DS5002FPM-16	0°C to +70°C	Yes	80 QFP
DS5002FPM-16+	0°C to +70°C	Yes	80 QFP
DS5002FMN-16	-40°C to +85°C	Yes	80 QFP
DS5002FMN-16+	-40°C to +85°C	Yes	80 QFP

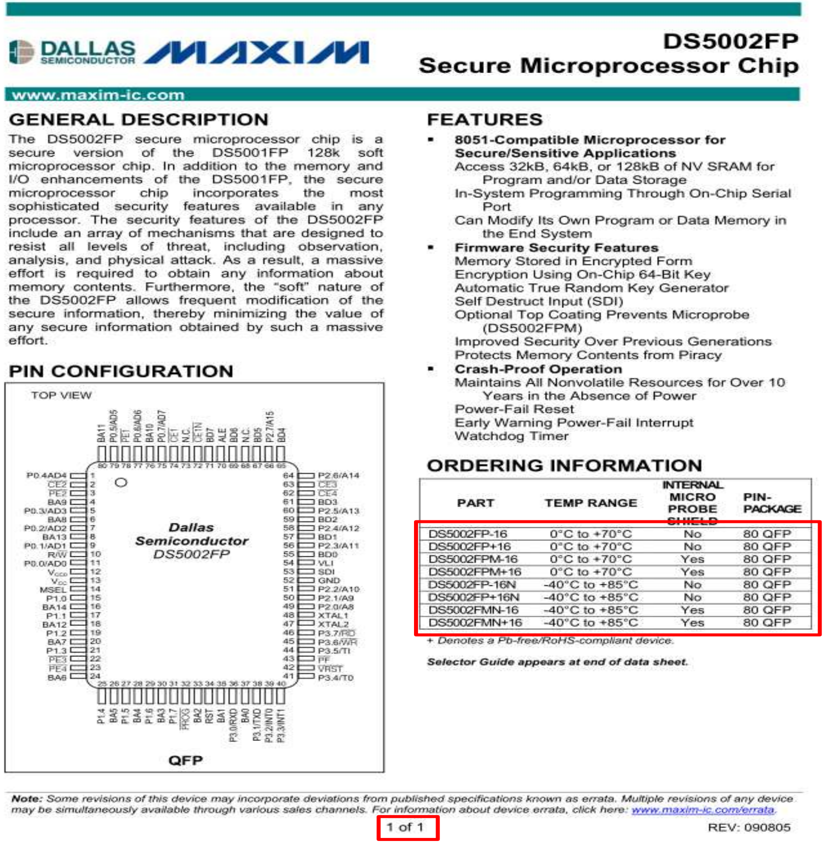
+ Denotes a Pb-free/RoHS-compliant device.

Selector Guide appears at end of data sheet.

Note: *Some revisions of this device may incorporate deviations from published specifications known as errata. Multiple revisions of any device may be simultaneously available through various sales channels. For information about device errata, click here: www.maxim-ic.com/errata.*

1 of 25

REV: 072806



DALLAS SEMICONDUCTOR MAXIM

DS5002FP Secure Microprocessor Chip

www.maxim-ic.com

GENERAL DESCRIPTION

The DS5002FP secure microprocessor chip is a secure version of the DS5001FP 128k soft microprocessor chip. In addition to the memory and I/O enhancements of the DS5001FP, the secure microprocessor chip incorporates the most sophisticated security features available in any processor. The security features of the DS5002FP include an array of mechanisms that are designed to resist all levels of threat, including observation, analysis, and physical attack. As a result, a massive effort is required to obtain any information about memory contents. Furthermore, the "soft" nature of the DS5002FP allows frequent modification of the secure information, thereby minimizing the value of any secure information obtained by such a massive effort.

PIN CONFIGURATION

FEATURES

8051-Compatible Microprocessor for Secure/Sensitive Applications
Access 32kB, 64kB, or 128kB of NV SRAM for Program and/or Data Storage
In-System Programming Through On-Chip Serial Port
Can Modify Its Own Program or Data Memory in the End System
Firmware Security Features
Memory Stored in Encrypted Form
Encryption Using On-Chip 64-Bit Key
Automatic True Random Key Generator
Self Destruct Input (SDI)
Optional Top Coating Prevents Microprobe (DS5002FPM)
Improved Security Over Previous Generations
Protects Memory Contents from Piracy
Crash-Proof Operation
Maintains All Nonvolatile Resources for Over 10 Years in the Absence of Power
Power-Fail Reset
Early Warning Power-Fail Interrupt
Watchdog Timer

ORDERING INFORMATION

PART	TEMP RANGE	INTERNAL MICRO PROBE SHIELD	PIN-PACKAGE
DS5002FP-16	0°C to +70°C	No	80 QFP
DS5002FP+16	0°C to +70°C	No	80 QFP
DS5002FPM-16	0°C to +70°C	Yes	80 QFP
DS5002FPM+16	0°C to +70°C	Yes	80 QFP
DS5002FP-16N	-40°C to +85°C	No	80 QFP
DS5002FP+16N	-40°C to +85°C	No	80 QFP
DS5002FMN-16	-40°C to +85°C	Yes	80 QFP
DS5002FMN+16	-40°C to +85°C	Yes	80 QFP

+ Denotes a Pb-free/RoHS-compliant device.

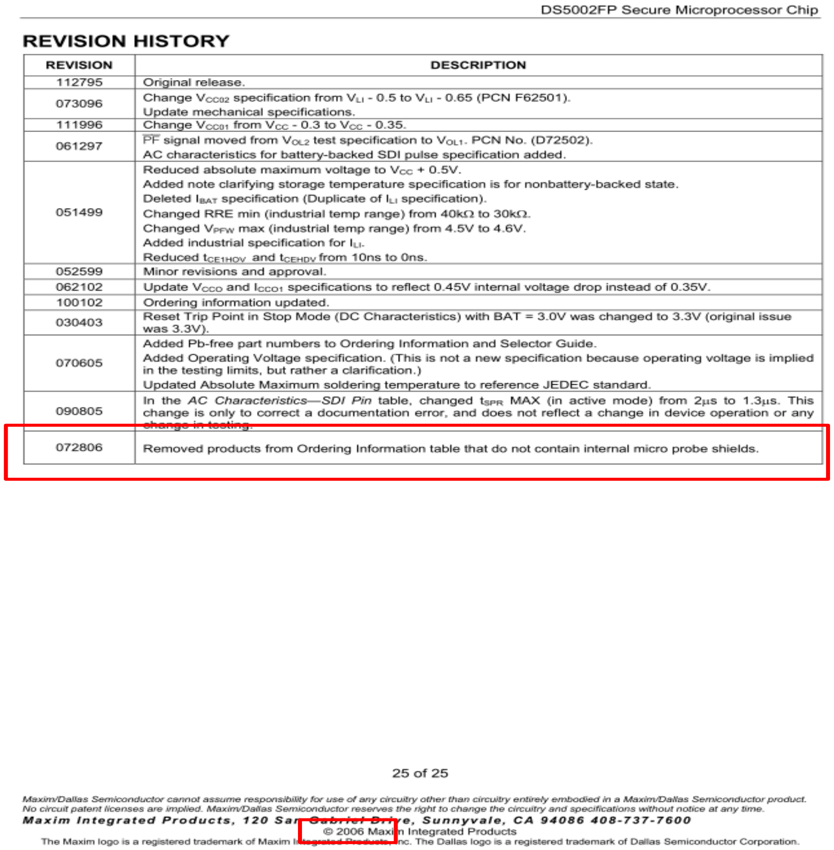
Selector Guide appears at end of data sheet.

Note: Some revisions of this device may incorporate deviations from published specifications known as errata. Multiple revisions of any device may be simultaneously available through various sales channels. For information about device errata, click here: www.maxim-ic.com/errata.

1 of 1

REV: 090805

incrementally updated PDF found in the wild
(removed parts, incorrect page number)

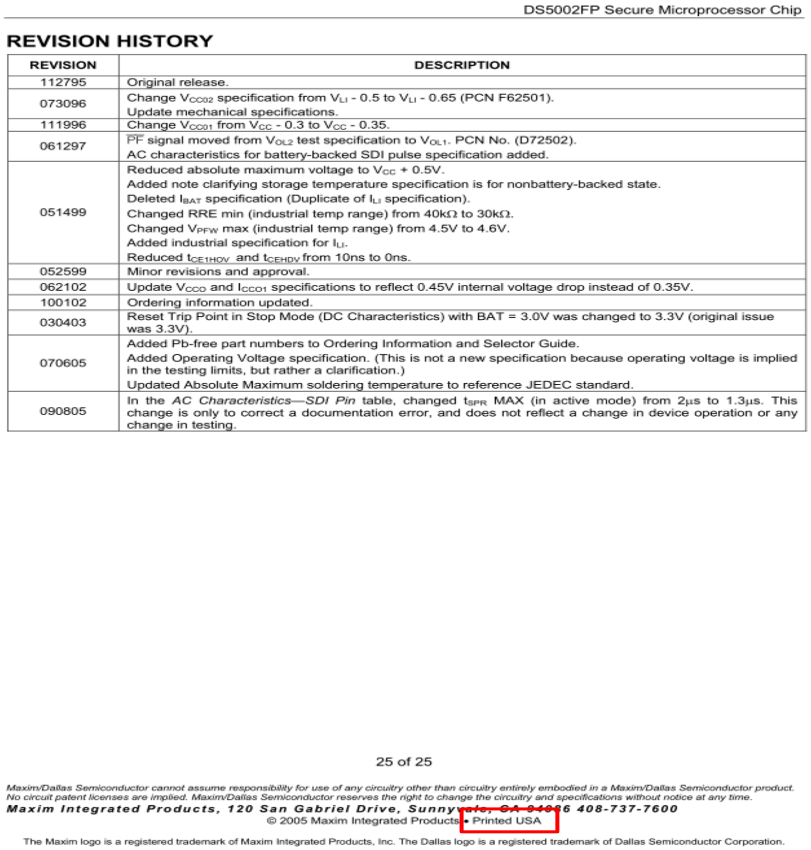


REVISION HISTORY

REVISION	DESCRIPTION
112795	Original release.
073096	Change V_{CC02} specification from V_{LI} - 0.5 to V_{LI} - 0.65 (PCN F62501). Update mechanical specifications.
111996	Change V_{CC01} from V_{CC} - 0.3 to V_{CC} - 0.35.
061297	$\overline{PF}$ signal moved from V_{OL2} test specification to V_{OL1}. PCN No. (D72502). AC characteristics for battery-backed SDI pulse specification added.
051499	Reduced absolute maximum voltage to V_{CC} + 0.5V. Added note clarifying storage temperature specification is for nonbattery-backed state. Deleted I_{BAT} specification (Duplicate of I_{LI} specification). Changed RRE min (industrial temp range) from 40kΩ to 30kΩ. Changed V_{PFW} max (industrial temp range) from 4.5V to 4.6V. Added industrial specification for I_{LI}. Reduced t_{CE1HOV} and t_{CEHDV} from 10ns to 0ns.
052599	Minor revisions and approval.
062102	Update V_{CCO} and I_{CCO1} specifications to reflect 0.45V internal voltage drop instead of 0.35V.
100102	Ordering information updated.
030403	Reset Trip Point in Stop Mode (DC Characteristics) with BAT = 3.0V was changed to 3.3V (original issue was 3.3V).
070605	Added Pb-free part numbers to Ordering Information and Selector Guide. Added Operating Voltage specification. (This is not a new specification because operating voltage is implied in the testing limits, but rather a clarification.) Updated Absolute Maximum soldering temperature to reference JEDEC standard.
090805	In the *AC Characteristics—SDI Pin* table, changed t_{SPR} MAX (in active mode) from 2µs to 1.3µs. This change is only to correct a documentation error, and does not reflect a change in device operation or any change in testing.
072806	Removed products from Ordering Information table that do not contain internal micro probe shields.

Maxim/Dallas Semiconductor cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim/Dallas Semiconductor product. No circuit patent licenses are implied. Maxim/Dallas Semiconductor reserves the right to change the circuitry and specifications without notice at any time.

Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600



REVISION HISTORY

REVISION	DESCRIPTION
112795	Original release.
073096	Change V_{CC02} specification from V_{LI} - 0.5 to V_{LI} - 0.65 (PCN F62501). Update mechanical specifications.
111996	Change V_{CC01} from V_{CC} - 0.3 to V_{CC} - 0.35.
061297	$\overline{PF}$ signal moved from V_{OL2} test specification to V_{OL1}. PCN No. (D72502). AC characteristics for battery-backed SDI pulse specification added.
051499	Reduced absolute maximum voltage to V_{CC} + 0.5V. Added note clarifying storage temperature specification is for nonbattery-backed state. Deleted I_{BAT} specification (Duplicate of I_{LI} specification). Changed RRE min (industrial temp range) from 40kΩ to 30kΩ. Changed V_{PFW} max (industrial temp range) from 4.5V to 4.6V. Added industrial specification for I_{LI}. Reduced t_{CE1HOV} and t_{CEHDV} from 10ns to 0ns.
052599	Minor revisions and approval.
062102	Update V_{CCO} and I_{CCO1} specifications to reflect 0.45V internal voltage drop instead of 0.35V.
100102	Ordering information updated.
030403	Reset Trip Point in Stop Mode (DC Characteristics) with BAT = 3.0V was changed to 3.3V (original issue was 3.3V).
070605	Added Pb-free part numbers to Ordering Information and Selector Guide. Added Operating Voltage specification. (This is not a new specification because operating voltage is implied in the testing limits, but rather a clarification.) Updated Absolute Maximum soldering temperature to reference JEDEC standard.
090805	In the *AC Characteristics—SDI Pin* table, changed t_{SPR} MAX (in active mode) from 2µs to 1.3µs. This change is only to correct a documentation error, and does not reflect a change in device operation or any change in testing.

Maxim/Dallas Semiconductor cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim/Dallas Semiconductor product. No circuit patent licenses are implied. Maxim/Dallas Semiconductor reserves the right to change the circuitry and specifications without notice at any time.

Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600

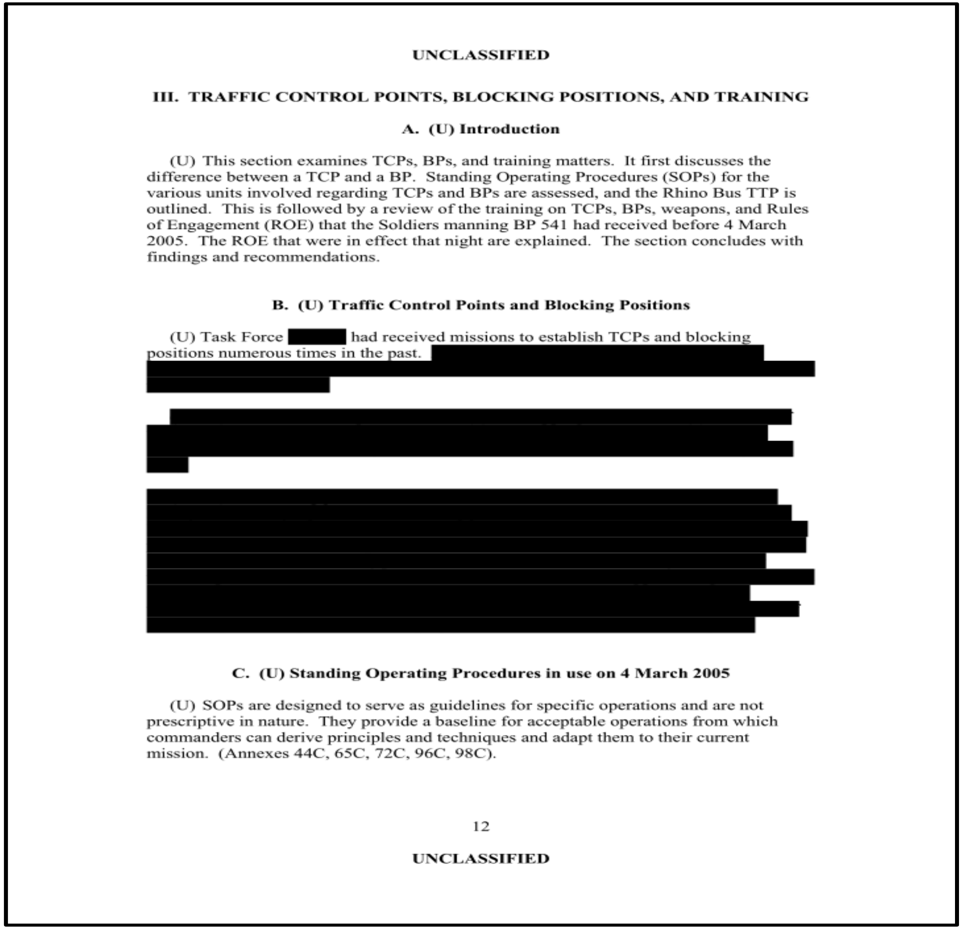
"Printed USA"

real examples

(of info leaks because of f*ck-up)

US Military in Iraq

1. decompress streams
2. locate page
3. locate content
4. locate `re` operators
5. disable `re` operators



UNCLASSIFIED

III. TRAFFIC CONTROL POINTS, BLOCKING POSITIONS, AND TRAINING

A. (U) Introduction

(U) This section examines TCPs, BPs, and training matters. It first discusses the difference between a TCP and a BP. Standing Operating Procedures (SOPs) for the various units involved regarding TCPs and BPs are assessed, and the Rhino Bus TTP is outlined. This is followed by a review of the training on TCPs, BPs, weapons, and Rules of Engagement (ROE) that the Soldiers manning BP 541 had received before 4 March 2005. The ROE that were in effect that night are explained. The section concludes with findings and recommendations.

B. (U) Traffic Control Points and Blocking Positions

(U) Task Force [redacted] had received missions to establish TCPs and blocking positions numerous times in the past. [redacted]

[redacted]

[redacted]

C. (U) Standing Operating Procedures in use on 4 March 2005

(U) SOPs are designed to serve as guidelines for specific operations and are not prescriptive in nature. They provide a baseline for acceptable operations from which commanders can derive principles and techniques and adapt them to their current mission. (Annexes 44C, 65C, 72C, 96C, 98C).

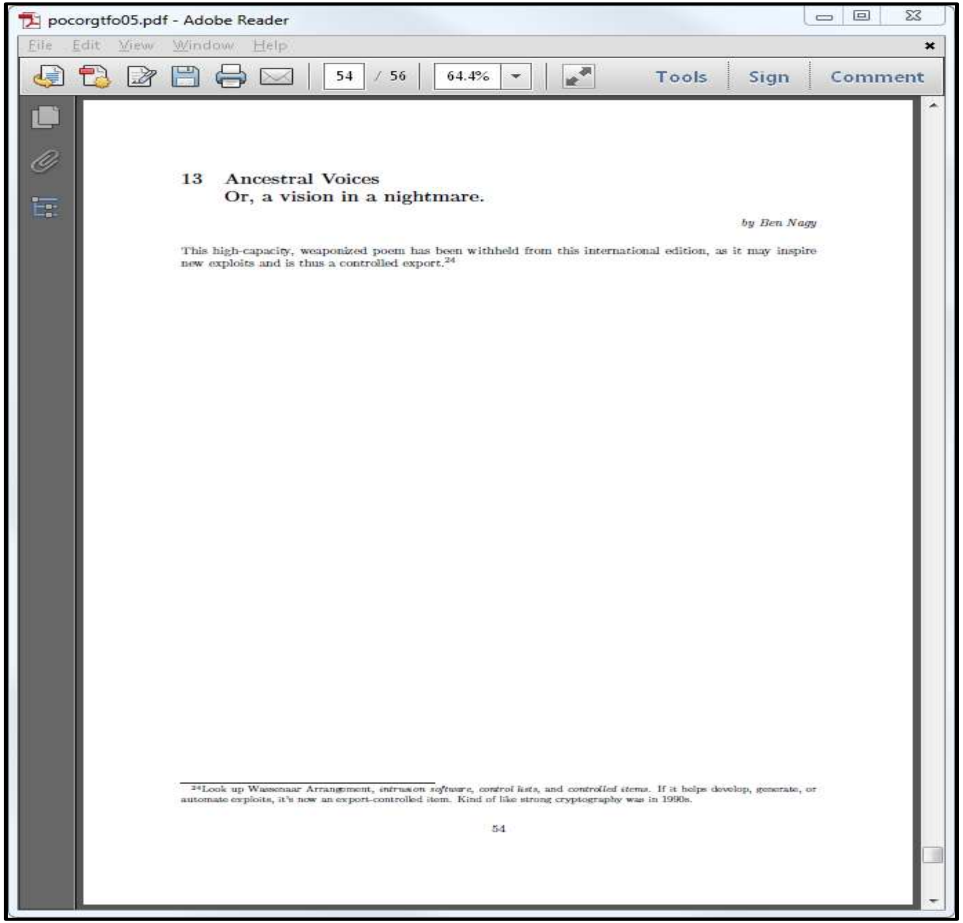
12

UNCLASSIFIED

pocorgtfo05.pdf

PoC||GTFO 0x05

1. restore structure
2. decompress
3. locate *
4. modify operator



13 Ancestral Voices
Or, a vision in a nightmare.

by Ben Nagy

This high-capacity, weaponized poem has been withheld from this international edition, as it may inspire new exploits and is thus a controlled export.[24]

[24] Look up Wassenaar Arrangement, *intrusion software*, *control lists*, and *controlled items*. If it helps develop, generate, or automate exploits, it's now an export-controlled item. Kind of like strong cryptography was in 1990s.

54

Conclusion

Conclusion

- the PDF file format is awkward & complex
  - different logics together
  - a format still evolving
    - 2.0 is in final draft at ISO, due in 2016
- accidental leaks of information can be easy
- text can still be modified
  - adding/removing watermarks and other contents

This was just an overview - have fun!

ACK

@Daeinar @veorq @_Quack1 @MunrekFR @dominicgs @mwgamera @kevinallix @munin @kristamonster @ClaudioAlbertin @push_pnx @JHeguia @doegox @gynvael @nst021 @iamreddave @chrisnklein

Bonus

Prepare a PDF for the text editor

(check out these tools + make your pick)

```
qpdf --qdf --object-streams=disable in.pdf  out.pdf
pdftk  in.pdf  cat output        out.pdf uncompress
mutool clean -d   in.pdf         out.pdf
podofouncompress  in.pdf         out.pdf
```

Check for errors after editing

```
qpdf --check                      edited.pdf
qpdf --show-xref                  edited.pdf
gs -o /dev/null -sDEVICE=pdfwrite edited.pdf
pdfinfo   -box  -f 1 -l 1000      edited.pdf
pdfimages -list -f 1 -l 1000      edited.pdf
pdffonts        -f 1 -l 1000      edited.pdf
```

Prepare a PDF for the text editor

- Be sure to check with *different* PDF viewers: Ghostscript/gv, MuPDF, SumatraPDF, FoxitReader, Adobe Reader, Adobe Acrobat, Chrome's builtin PDF viewer, PDF.js in Firefox, Evince, Preview.app (on OSX), Zathura...
- Scroll through all PDF pages (some errors only materialize when page must be rendered)
- If Acrobat/Adobe Reader open PDF with no warning or error, but upon closing ask if you want to *"save the changes"*... it's not ***your*** changes it wants to save, but some errors it found!

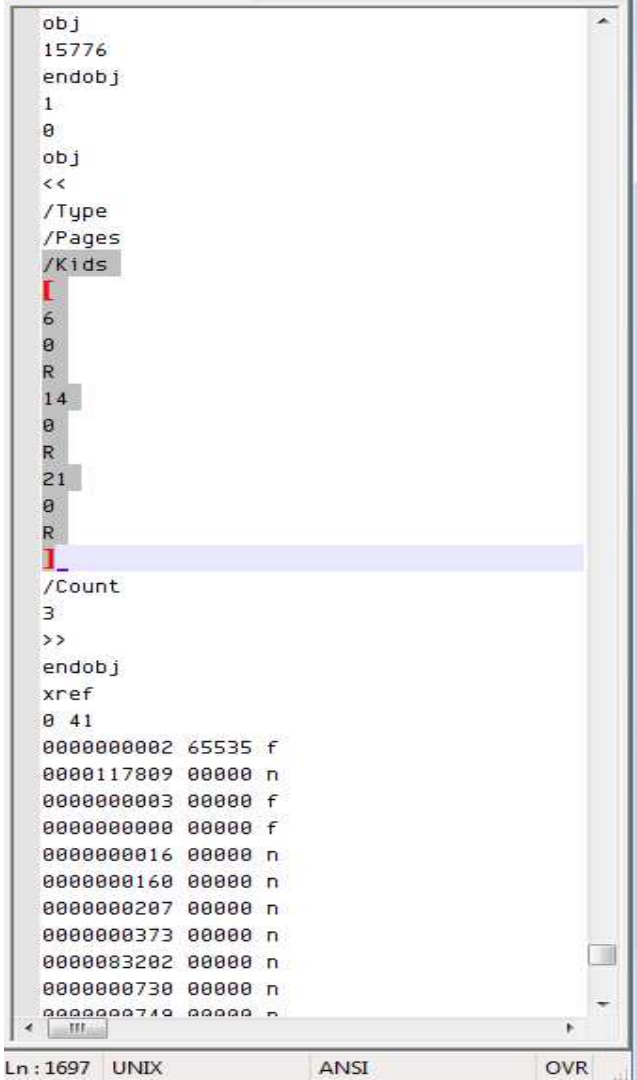
Fixing errors after editing

```
qpdf edited.pdf fixed.pdf
gs -o fixed.pdf -sDEVICE=pdfwrite edited.pdf
mutool clean edited.pdf fixed.pdf
```

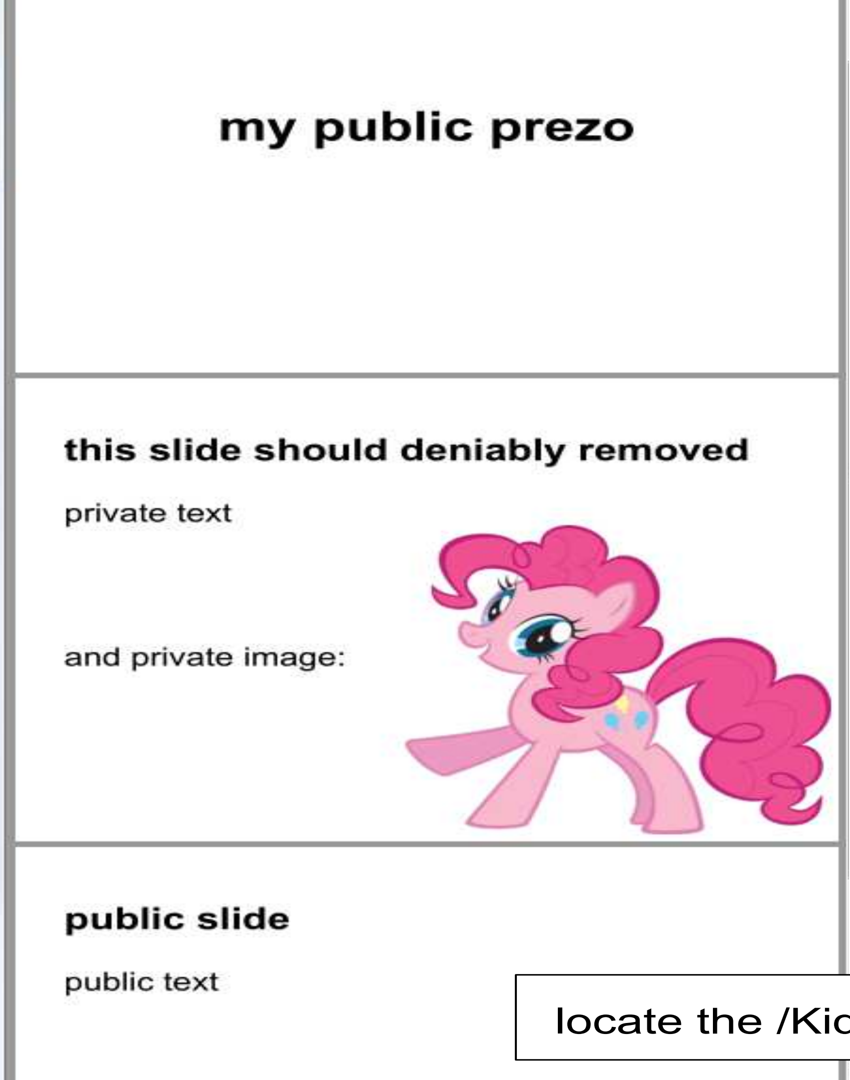
Remove a page ?

easy hiding

1. remove reference from `/Kids` (commenting out is sufficient)
2. write it back later



```
obj
15776
endobj
1
0
obj
<<
/Type
/Pages
/Kids
[
6
0
R
14
0
R
21
0
R
]
/Count
3
>>
endobj
xref
0 41
0000000002 65535 f
0000117809 00000 n
0000000003 00000 f
0000000000 00000 f
0000000016 00000 n
0000000160 00000 n
0000000207 00000 n
0000000373 00000 n
0000083202 00000 n
0000000730 00000 n
0000000749 00000 n
```



my public prezo

this slide should deniably removed

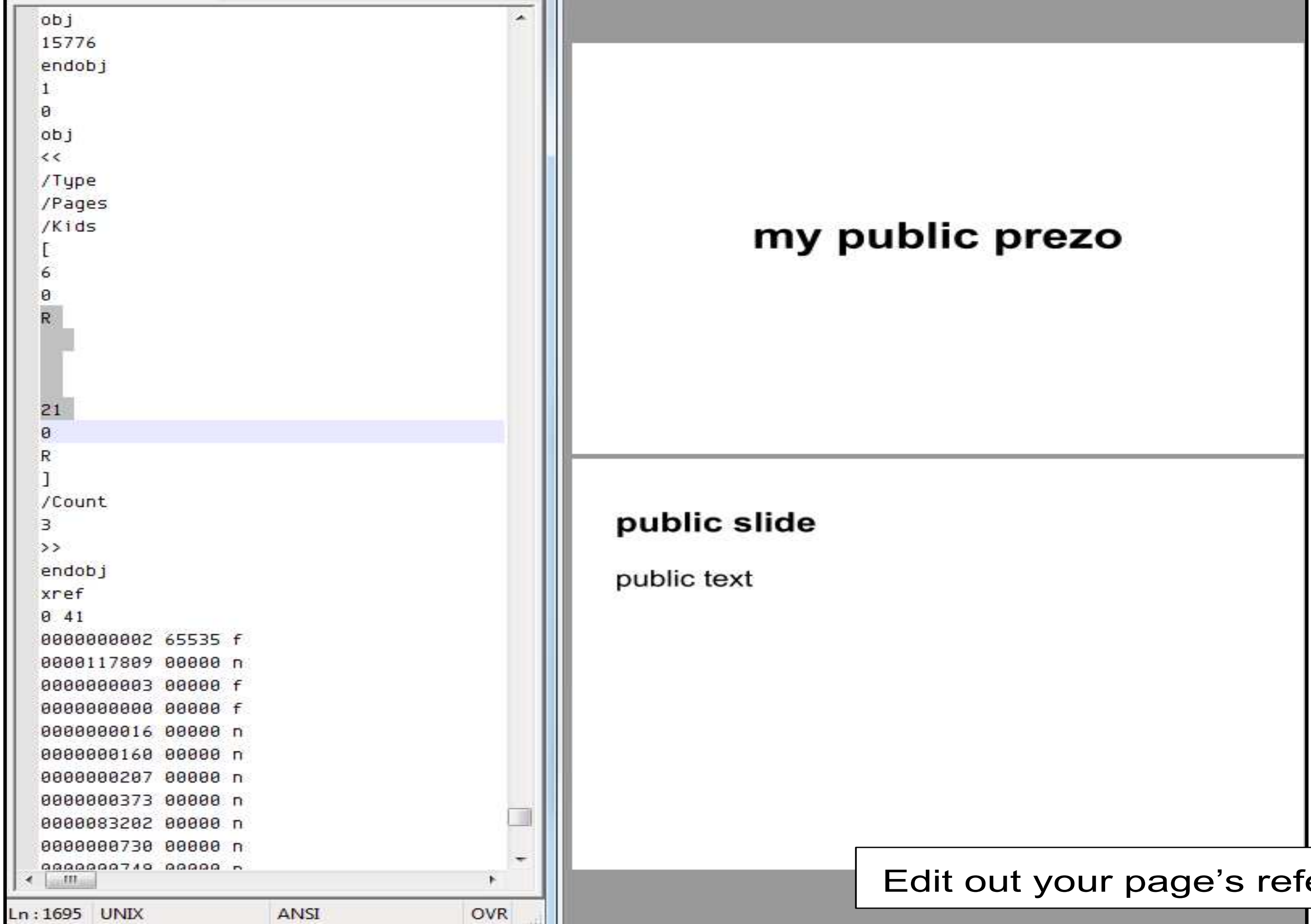
private text

and private image:

public slide

public text

locate the /Kids array



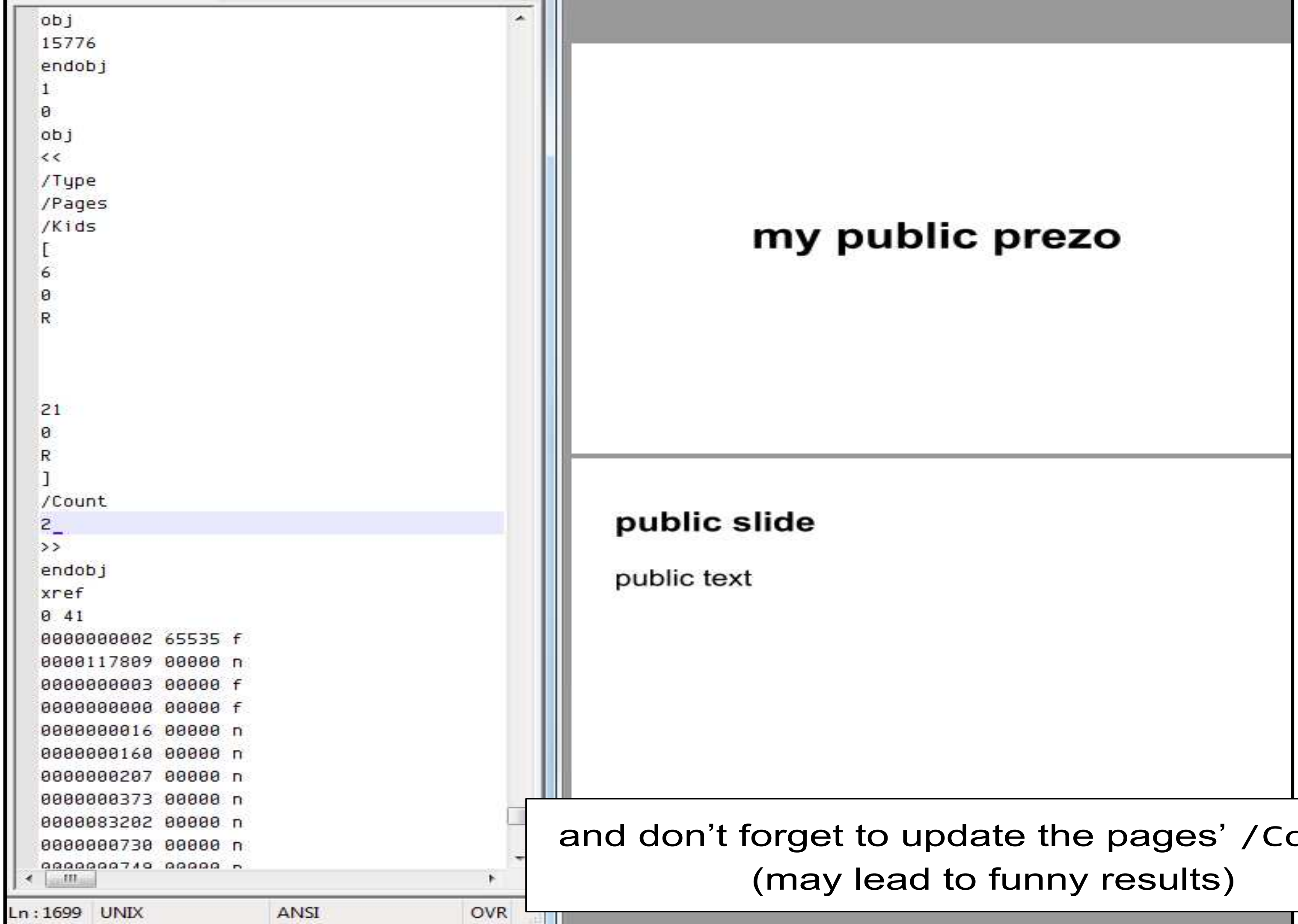
```
obj
15776
endobj
1
0
obj
<<
/Type
/Pages
/Kids
[
6
0
R



21
0
R
]
/Count
3
>>
endobj
xref
0 41
0000000002 65535 f
0000117809 00000 n
0000000003 00000 f
0000000000 00000 f
0000000016 00000 n
0000000160 00000 n
0000000207 00000 n
0000000373 00000 n
0000083202 00000 n
0000000730 00000 n
0000000749 00000 n
```

my public prezo

public slide

public text

Edit out your page’s reference



```
obj
15776
endobj
1
0
obj
<<
/Type
/Pages
/Kids
[
6
0
R




21
0
R
]
/Count
2_
>>
endobj
xref
0 41
0000000002 65535 f
0000117809 00000 n
0000000003 00000 f
0000000000 00000 f
0000000016 00000 n
0000000160 00000 n
0000000207 00000 n
0000000373 00000 n
0000083202 00000 n
0000000730 00000 n
0000000749 00000 n
```

my public prezo

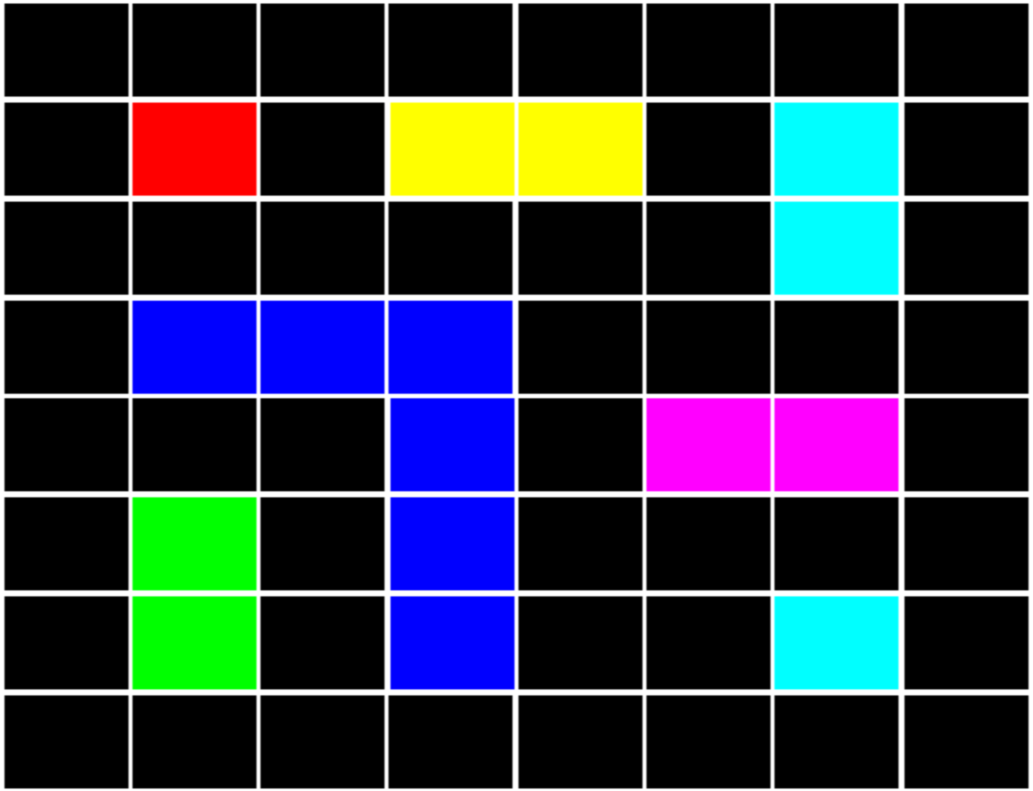
public slide

public text

and don’t forget to update the pages’ `/Count` ☺
(may lead to funny results)

A little riddle to solve...

Which hidden message is in this PDF?



this slide intentionally left blank

PDF + PostScript: Myths and Facts

- PDF is ***not*** an "extension of PostScript" (PS is a Turing-complete programming language -- PDF is not!)
- Yes, PDF inherited its basic *graphics model* from PS (and extended it with many new features)
- But PDF got everything removed what made PS a programming language: conditions, loops, arithmetics,... *precisely because it did more bad than good for PS as: (1) a universal "electronic document format"; (2) a "reliable print job format"*
  (however, its retrofitted JavaScript support since PDF-1.3 makes up for this ;-)

fractals-sierpinski-quadrat-depth-[1-8].ps
fractals-sierpinski-quadrat-depth-[1-8].pdf

PDF + PostScript: Guesses and Facts

- Which format does typically requires smaller file sizes for identical screen or page content? *PDF or PS?* (Make an un-educated guess, if you dare. Then look at the *fractals-sierpinski-*.ps* files. Open them in a text editor. Convert them to PDF with Ghostscript -- a working command is in the PS comments. Check their respective file sizes.)

- Which graphics type does typically produce smaller files for the same screen or page content? *Raster images or vector drawings?* (Again, make an un-educated guess, if you dare. Then look at the same files as above. Open them in a text editor. Convert them to PDF with Ghostscript -- a working command is in the PS comments. Open them, one after another in your favorite PDF viewer and watch them render... Would the PS version render faster?)

(Warning! Beware of the files with depth 8 or higher!)

@angealbertini
@pdfkunfoo