No more dumb hex!

Rethinking binary tooling

🦌🇵🇱 Rafał Hirsz

👶🇫🇷 Ange Albertini

2019/03/21

🏰🇩🇪 Troopers, Heidelberg

TL;DR: 🔢🔍=  + 🛠️=🦕+🦖->🤔->💡

Binary analysis is still prehistoric (non executables, that is)

Same old binary tooling. "Cool", but DUMB!

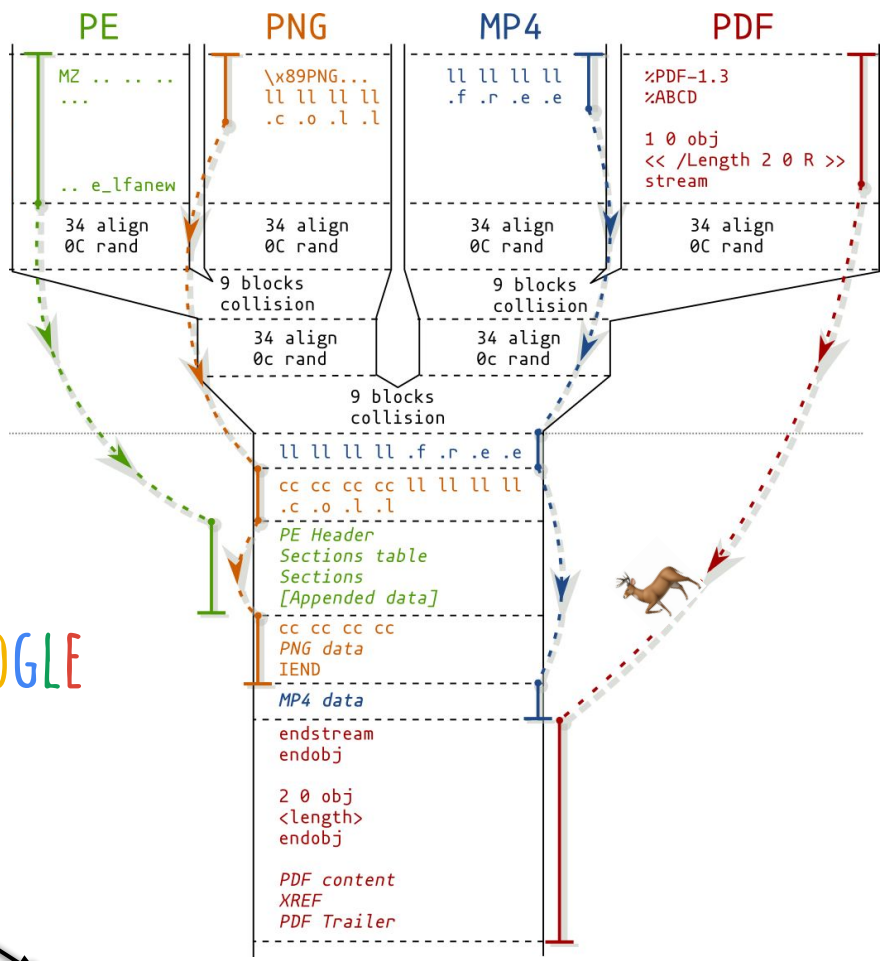
Here are new perspectives and ~~tools~~ *messy prototypes*

to dissect, craft or visualize file formats.

Ange Albertini - 👼

- hex addict since 1989
- Author of Corkami
- malware analyst for 13 years

Information Security Engineer at Google

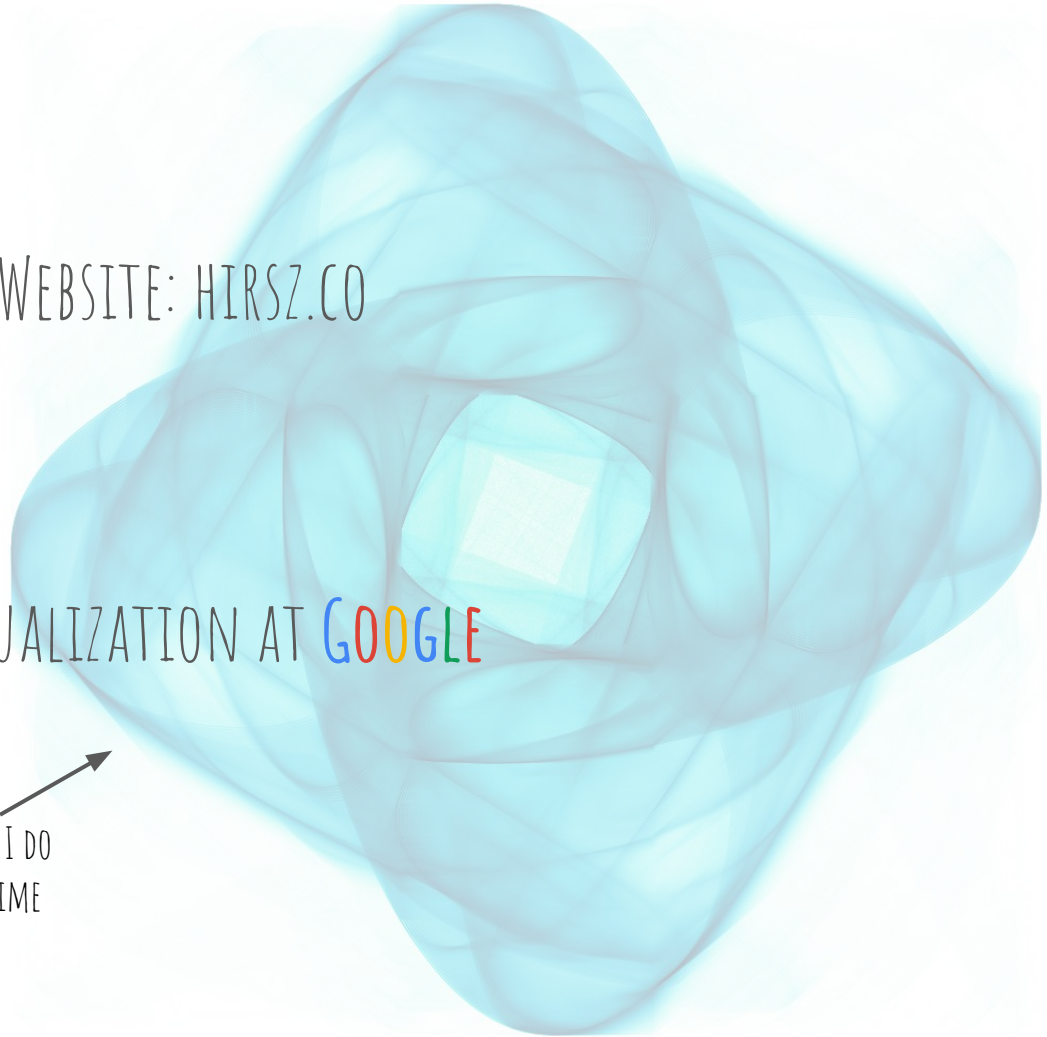


Disclaimer:
These are our own views.
Not from **any** of our employers.

The kind of things I do during my spare time → instant MD5 collision of any PE, PNG, MP4 and PDF quartet.

Rafał Hirsz - 

- Twitter: @hDevo, GitHub: evol, Website: hirsz.co
- Frontend Developer for 10 years
- not really related to infosec
- Software Engineer for Data Visualization at Google



Disclaimer:
These are our own views.
Not from **any** of our employers.

The kind of things I do during my spare time

👼 In 1989

Our PC (10 MHz, 20 Mb HDD)
was infected by a virus.

https://en.wikipedia.org/wiki/Ping-Pong_virus

Thankfully,

a french magazine explained

how to remove it...

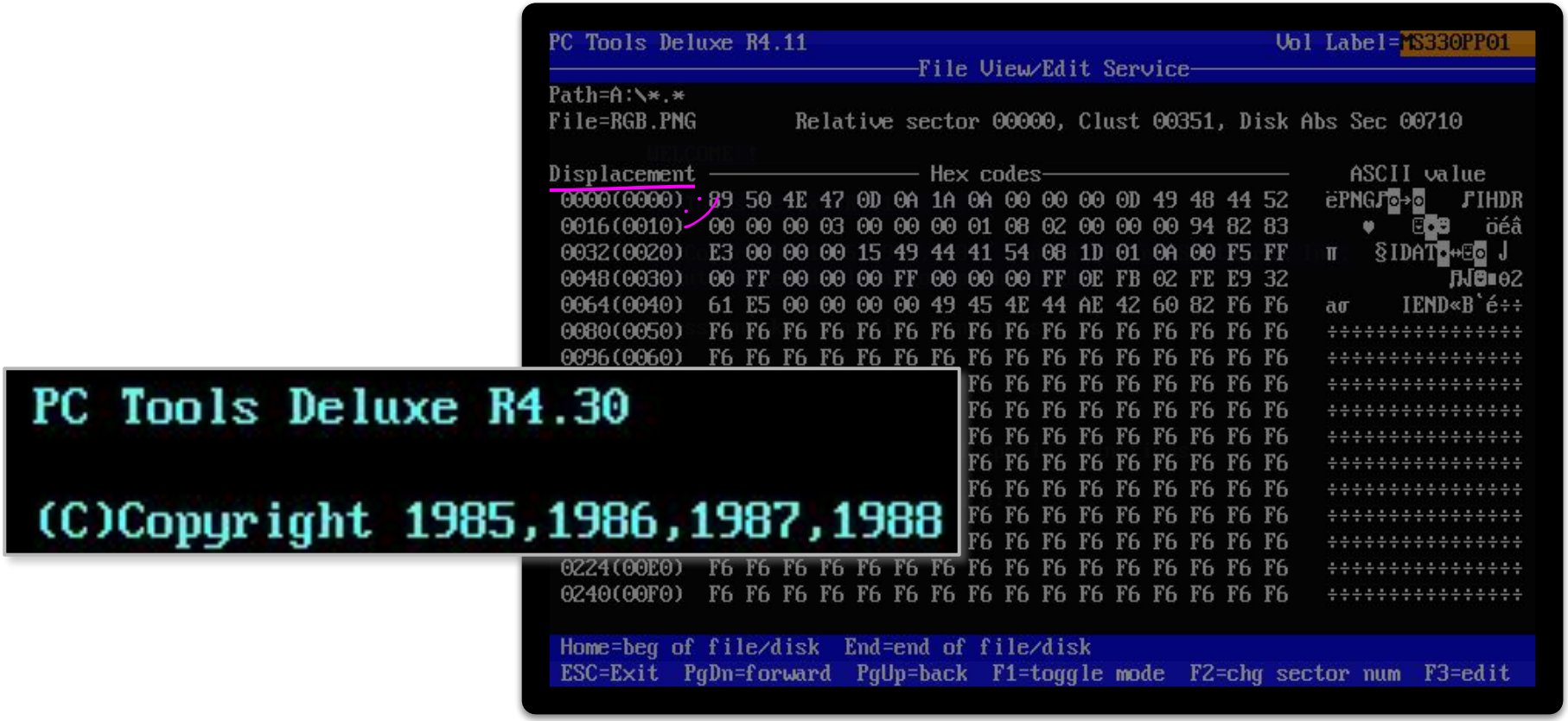
http://fr.1001mags.com/parution/svm/numero-66-novembre-1989/page-146-147-texte-integral

...BY YOURSELF,

WITH A HEX EDITOR!

Dans la série des virus qui sont censés vous sortir de la torpeur inhérente à des heures de travail fastidieux devant un écran, il y a aussi le Ping-pong (ou Italian Bouncing) : avec une lenteur désespérante, une baballe rebondit sur les caractères, puis elle les efface, puis une autre apparaît, rebondit encore, et le phénomène continue de se reproduire jusqu'à ce que l'écran ne soit plus que balles vagabondes. C'est certainement le plus visuel des virus sur compatibles IBM, mais aussi le plus exaspérant et le plus récurrent. Installé sur un secteur des pistes de démarrage, il occupe deux autres secteurs qu'il marque comme endommagés dans la table d'allocation des fichiers. Par chance, il n'attaque que les IBM PC-XT. Pour s'en débarrasser, il faut rétablir les pistes de démarrage dans leur état d'origine. Avec un éditeur d'octets du type PC-Tools, vérifiez la présence des octets 33 C0 dans les zones 30 et 31 du secteur d'amorçage du disque dur ; s'ils sont bien présents, mieux vaut exécuter la commande SYS depuis une disquette Système saine; à la fin de la première table d'allocation des fichiers du disque dur, remplacez les trois derniers octets (FF 7F FF) par FF 0F 00. Puis localisez le code du virus lui-même, qui commence par FF 06 F3 7D 8B 1E, et remplacez-le (ainsi que tous les octets qui suivent, jusqu'à 55 AA) par F6 si le formatage est dû à la commande FORMAT du système, ou par 00 s'il provient de PC-Tools.


comme endommagés dans la table d'alloca-
tion des fichiers. Par chance, il n'attaque que
les IBM PC-XT. Pour s'en débarrasser, il faut
rétablir les pistes de démarrage dans leur état
d'origine. Avec un éditeur d'octets du type PC-
Tools, vérifiez la présence des octets 33 C0
dans les zones 30 et 31 du secteur d'amor-
çage du disque dur ; s'ils sont bien présents,
mieux vaut exécuter la commande SYS de-
puis une disquette Système saine ; à la fin de
la première table d'allocation des fichiers du
disque dur, remplacez les trois derniers oc-
tets (FF 7F FF) par FF 0F 00. Puis localisez le
code du virus lui-même, qui commence par FF
06 F3 7D 8B 1E, et remplacez-le (ainsi que
tous les octets qui suivent, jusqu'à 55 AA) par
F6 si le formatage est dû à la commande FOR-
MAT du système, ou par 00 s'il provient de PC-
Tools. Si l'opération vous semble trop com-



With the typical **O**ffset/**H**ex/**A**SCII view

(My first interaction with a virus...)

The OHA view

It takes 229 characters for Philipp Akesson

https://github.com/pakesson/codegolf/tree/master/hexdump

```
int main(int a, char**v){
  int c, n, t=0;
  FILE *p=fopen(v[1], "r");
  while (c != -1){
    char l[81];
    sprintf(l, "%08X%*c", t, 72, 0);
    for (n=0; n<16 && (c=fgetc(p))!= -1;++n, ++t) {
      sprintf(l + 9 + n*3, "%02X", c);
        l[11 + n*3] = 32;
        l[58 + n] = (c>31 && c<124) ? c : 46;
    }
  puts(l);
  }
}
```

```
$ ./hexdump /bin/sh
00000000 7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010 03 00 3E 00 01 00 00 00 20 4A 00 00 00 00 00 00  ..>..... J......
00000020 40 00 00 00 00 00 00 00 58 D3 01 00 00 00 00 00  @.......X.......
00000030 00 00 00 00 40 00 38 00 09 00 40 00 1C 00 1B 00  ....@.8...@.....
00000040 06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00  ........@.......
[...]
0001DA10 00 00 00 00 00 00 00 00 01 00 00 00 03 00 00 00  ................
0001DA20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001DA30 54 D2 01 00 00 00 00 00 01 01 00 00 00 00 00 00  T...............
0001DA40 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
0001DA50 00 00 00 00 00 00 00 00                          ........
```

Trivial. Reliable. Dumb.

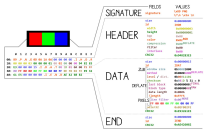
It's great & cool, but...

Can't we do better?

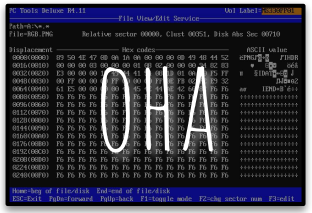
What do we want?

File → ?

- View
- Edit
- Focus
  - hide/highlight/...
  - tell a story



FILE



OHA

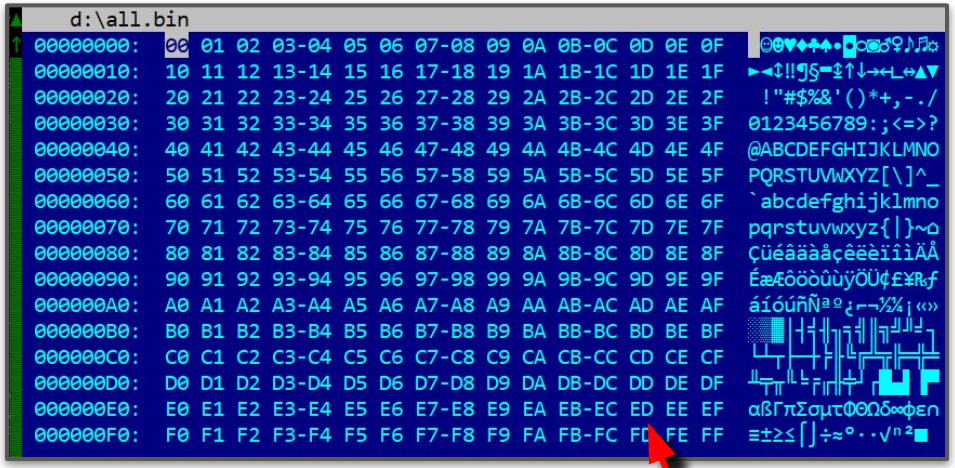
VIEW: GENERIC

EDIT: HARD

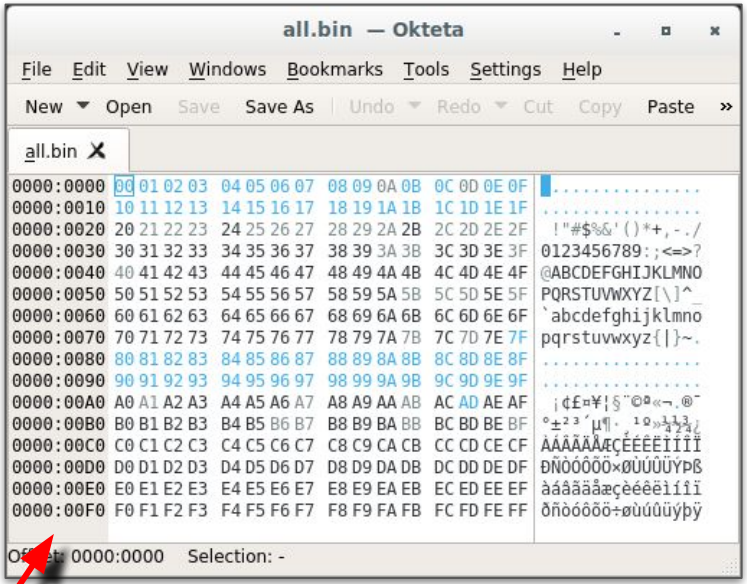
FOCUS: NONE

Limitations

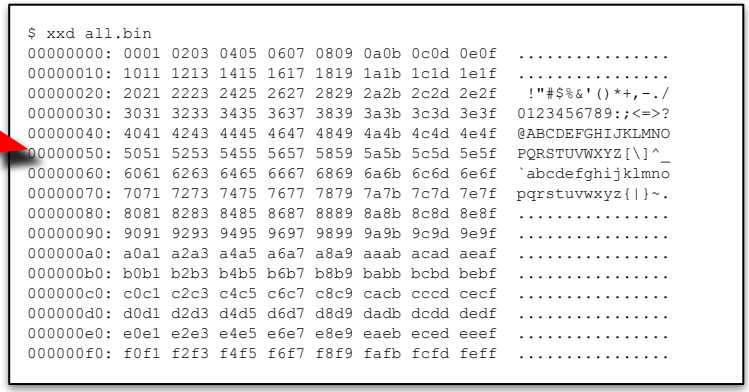
- everything shown equal:
  - compressed data? Huge blocks of `00` ?
  - we don't need Hex & ASCII simultaneously
    Modern fonts made it useless.
- no knowledge of the underneath format: critical stuff? comments?
- wrapping at 16 bytes.



Old school? Useful!



What you see with various hex viewers
(all 256 bytes are present once)



Alphanumeric < ASCII < CodePage

☺☻♥♦♣♠•◘○◙♂♀♪♫☼

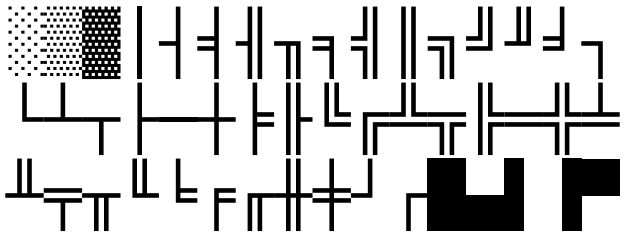
►◄↕‼¶§▬↨↑↓→←∟↔▲▼

⌂

Çüéâäàåçêëèïîì ÄÅ

ÉæÆôöòûùÿÖÜ¢£¥₧ƒ

áíóúñÑªº¿⌐¬½¼¡«»

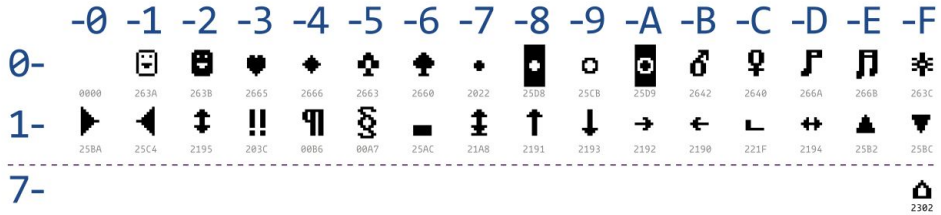


αßΓπΣσµτΦΘΩδ∞φε∩

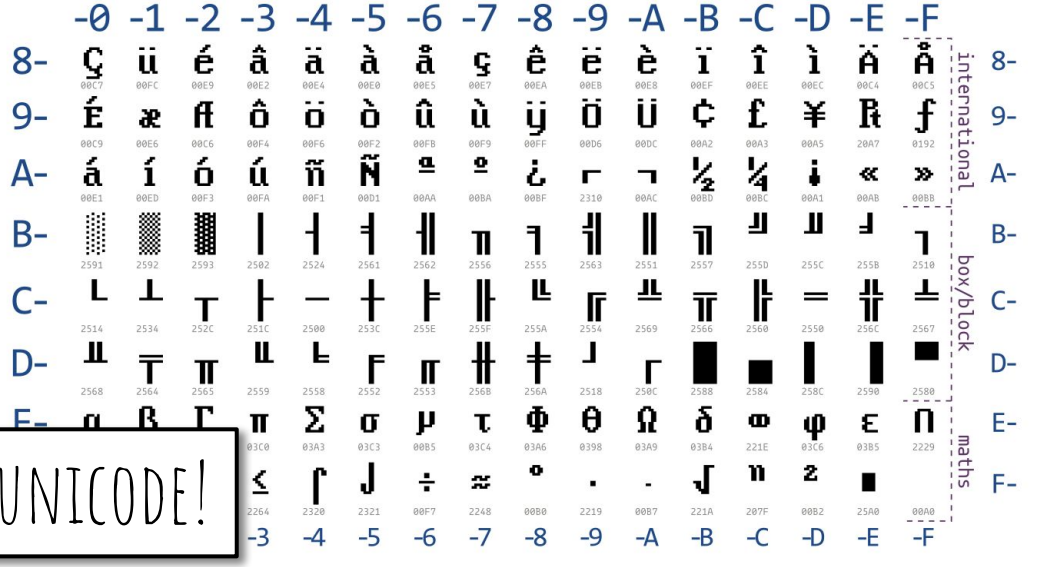
≡±≥≤⌠⌡÷≈°∙·√ⁿ²■

YET THESE CHARACTERS STILL EXIST IN UNICODE!

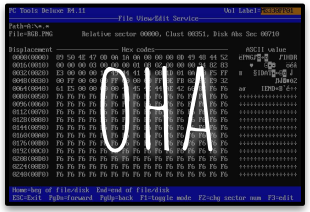
Control characters



Extension: Code Page 437



Select difficulty



YES

KEEP IT?

NO

FIX IT!

KILL IT!

VIEWING++

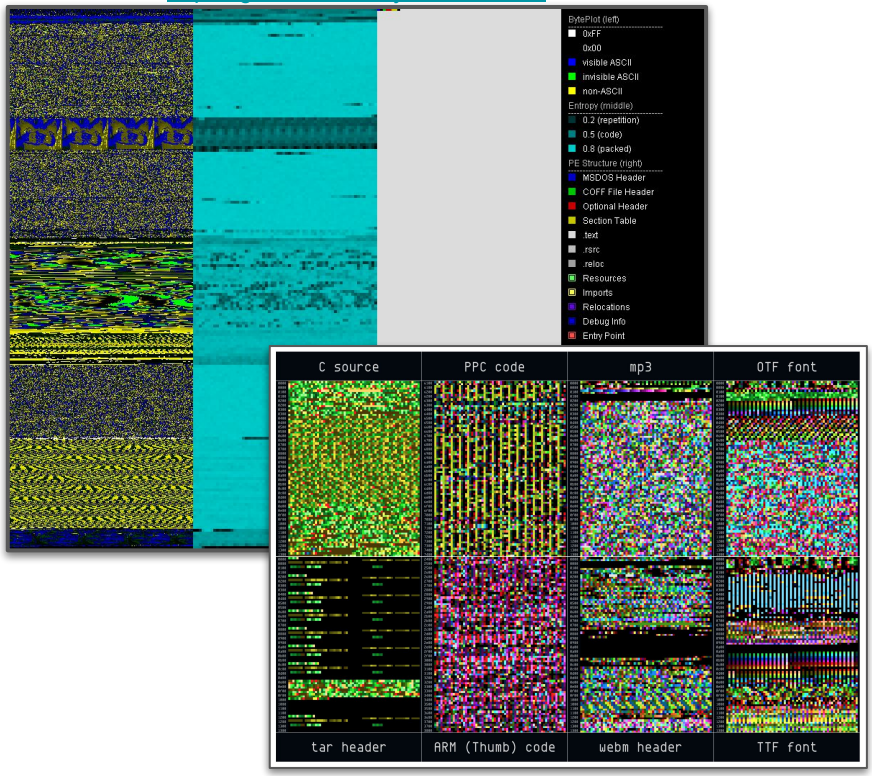
EDITING++

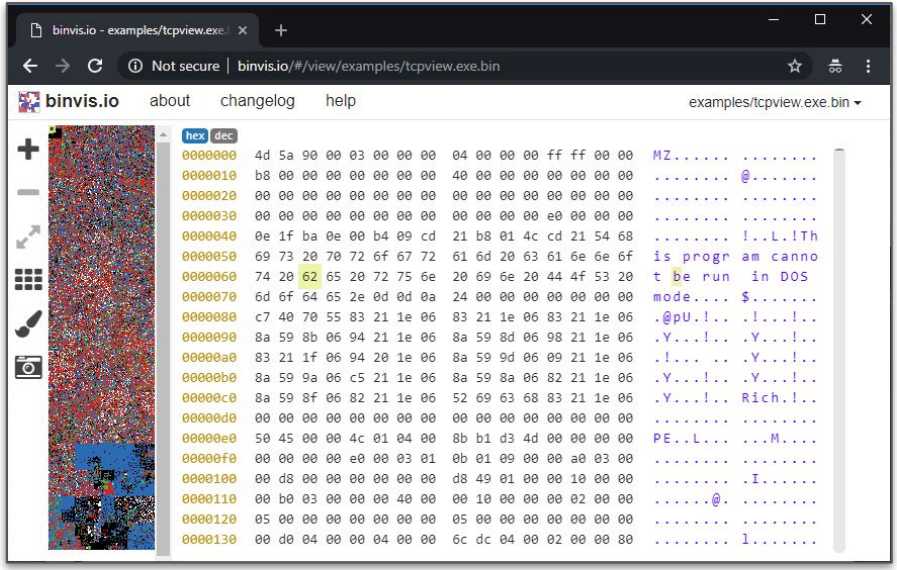
Fixing the OHA view

Current visualisations: navigation map

PortEx, BinVis

https://github.com/katjahahn/PortEx

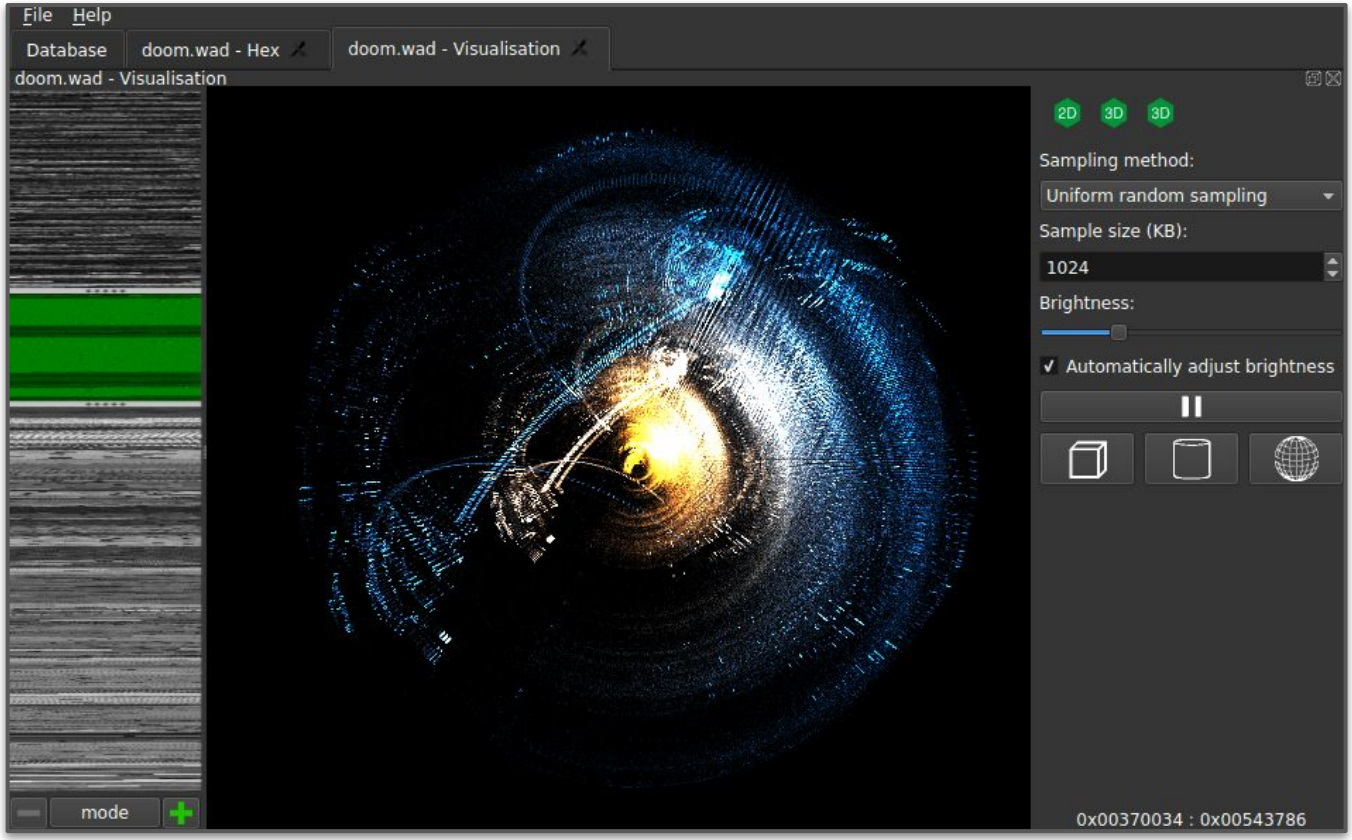




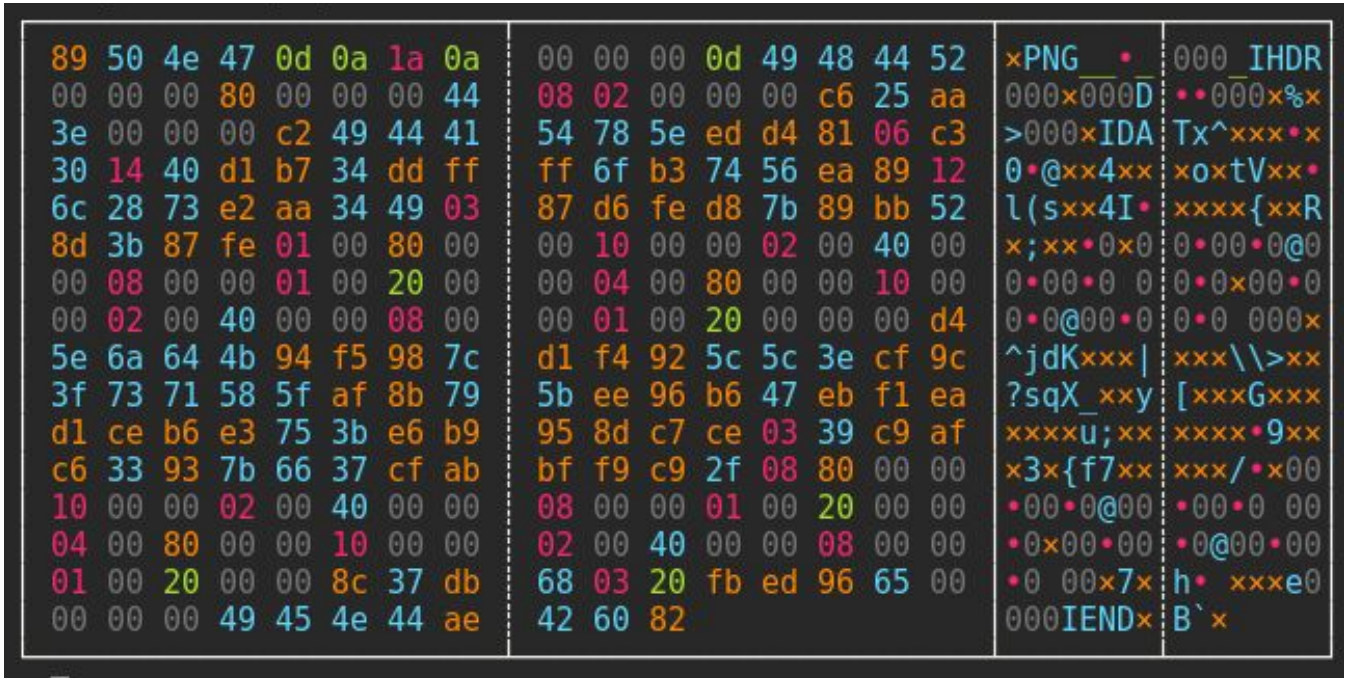
https://github.com/FireyFly/pixd

Or weird stuff....

Veles



Dumb coloring

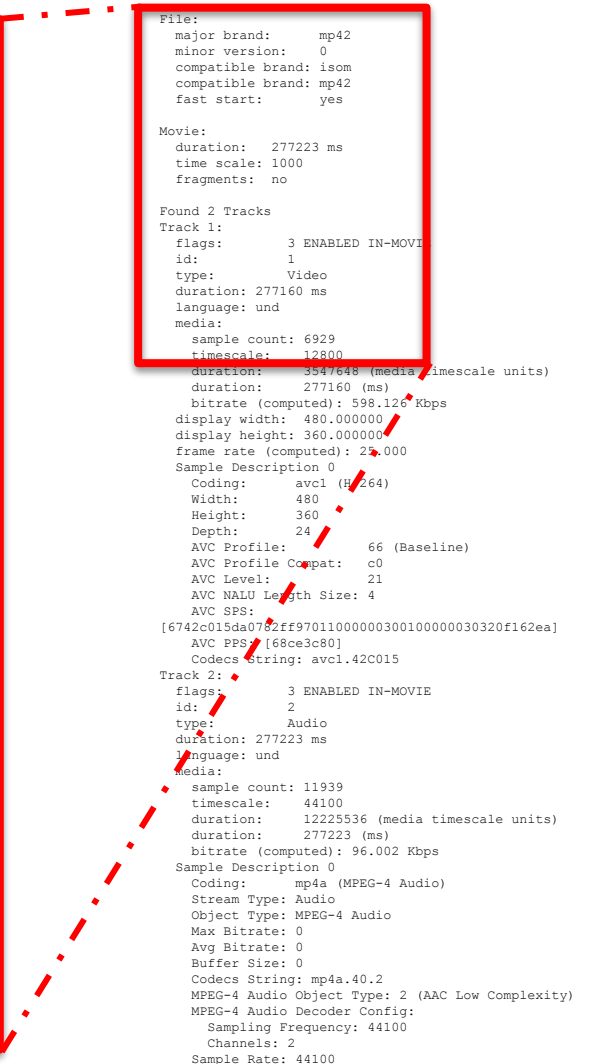


```
89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52  ×PNG__•_  000_IHDR
00 00 00 80 00 00 00 44  08 02 00 00 00 c6 25 aa  000×000D  ••0000×%×
3e 00 00 00 c2 49 44 41  54 78 5e ed d4 81 06 c3  >000×IDA  Tx^×××•×
30 14 40 d1 b7 34 dd ff  ff 6f b3 74 56 ea 89 12  0•@××4××  ×o×tV××•
6c 28 73 e2 aa 34 49 03  87 d6 fe d8 7b 89 bb 52  l(s××4I•  ××××{××R
8d 3b 87 fe 01 00 80 00  00 10 00 00 02 00 40 00  ×;××•0×0  0•00•0@0
00 08 00 00 01 00 20 00  00 04 00 80 00 00 10 00  0•00•0 0  0•0×00•0
00 02 00 40 00 00 08 00  00 01 00 20 00 00 00 d4  0•0@00•0  0•0 000×
5e 6a 64 4b 94 f5 98 7c  d1 f4 92 5c 5c 3e cf 9c  ^jdK×××|  ×××\\>××
3f 73 71 58 5f af 8b 79  5b ee 96 b6 47 eb f1 ea  ?sqX_××y  [×××G×××
d1 ce b6 e3 75 3b e6 b9  95 8d c7 ce 03 39 c9 af  ××××u;××  ××××•9××
c6 33 93 7b 66 37 cf ab  bf f9 c9 2f 08 80 00 00  ×3×{f7××  ×××/•×00
10 00 00 02 00 40 00 00  08 00 00 01 00 20 00 00  •00•0@00  •00•0 00
04 00 80 00 00 10 00 00  02 00 40 00 00 08 00 00  •0×00•00  •0@00•00
01 00 20 00 00 8c 37 db  68 03 20 fb ed 96 65 00  •0 00×7×  h• ×××e0
00 00 00 49 45 4e 44 ae  42 60 82                 000IEND×  B`×
```

Don't we know something about this format...?

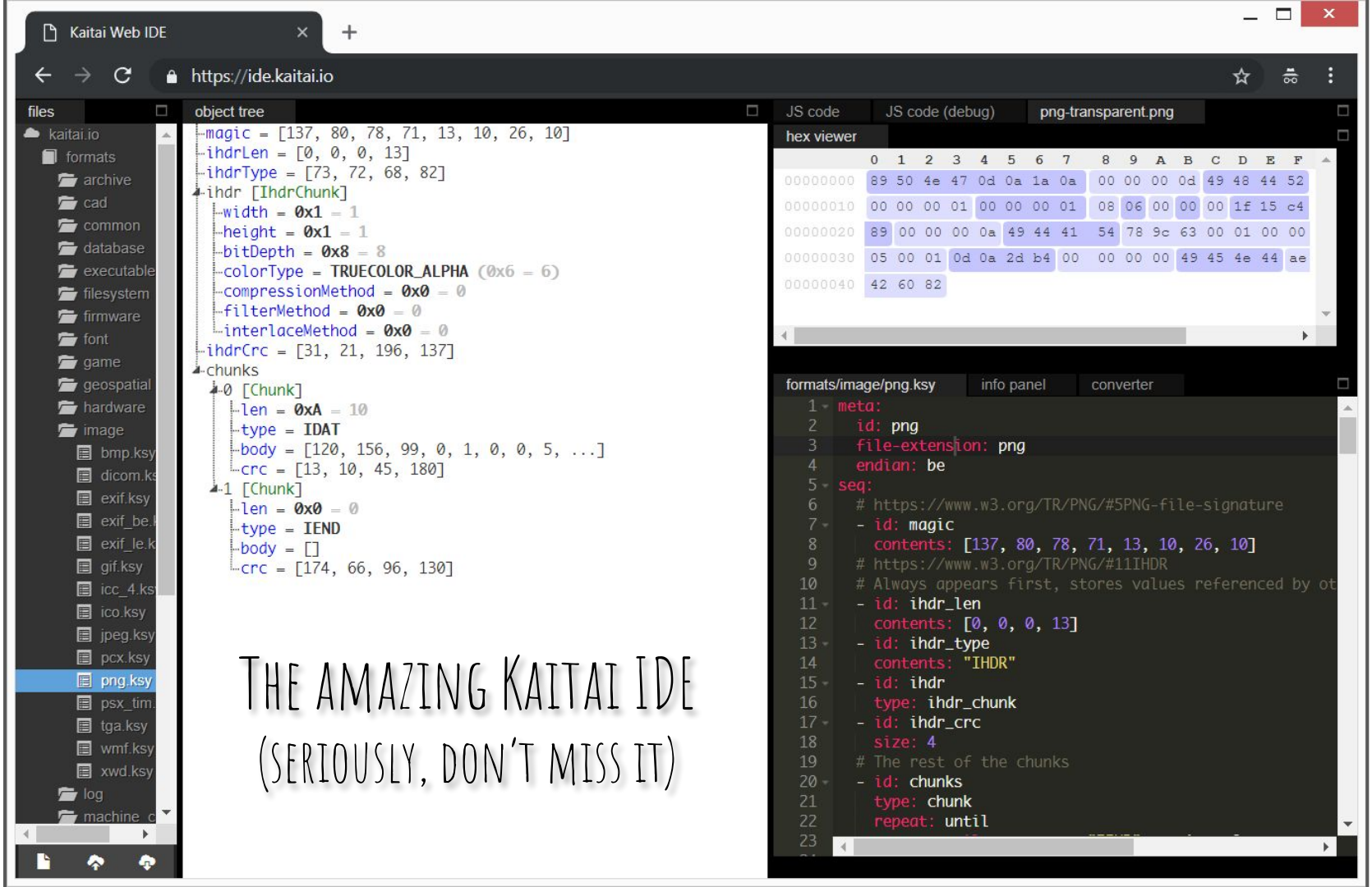
Parsing dissociates the content from its meaning.

```
File:
  major brand:       mp42
  minor version:     0
  compatible brand: isom
  compatible brand: mp42
  fast start:        yes

Movie:
  duration:    277223 ms
  time scale: 1000
  fragments:  no

Found 2 Tracks
Track 1:
  flags:          3
  id:             1
  type:           Video
  duration: 277160 ms
  language: und
  media:
    sample count: 6929
```



Standard idea:
Associate OHA with parsing

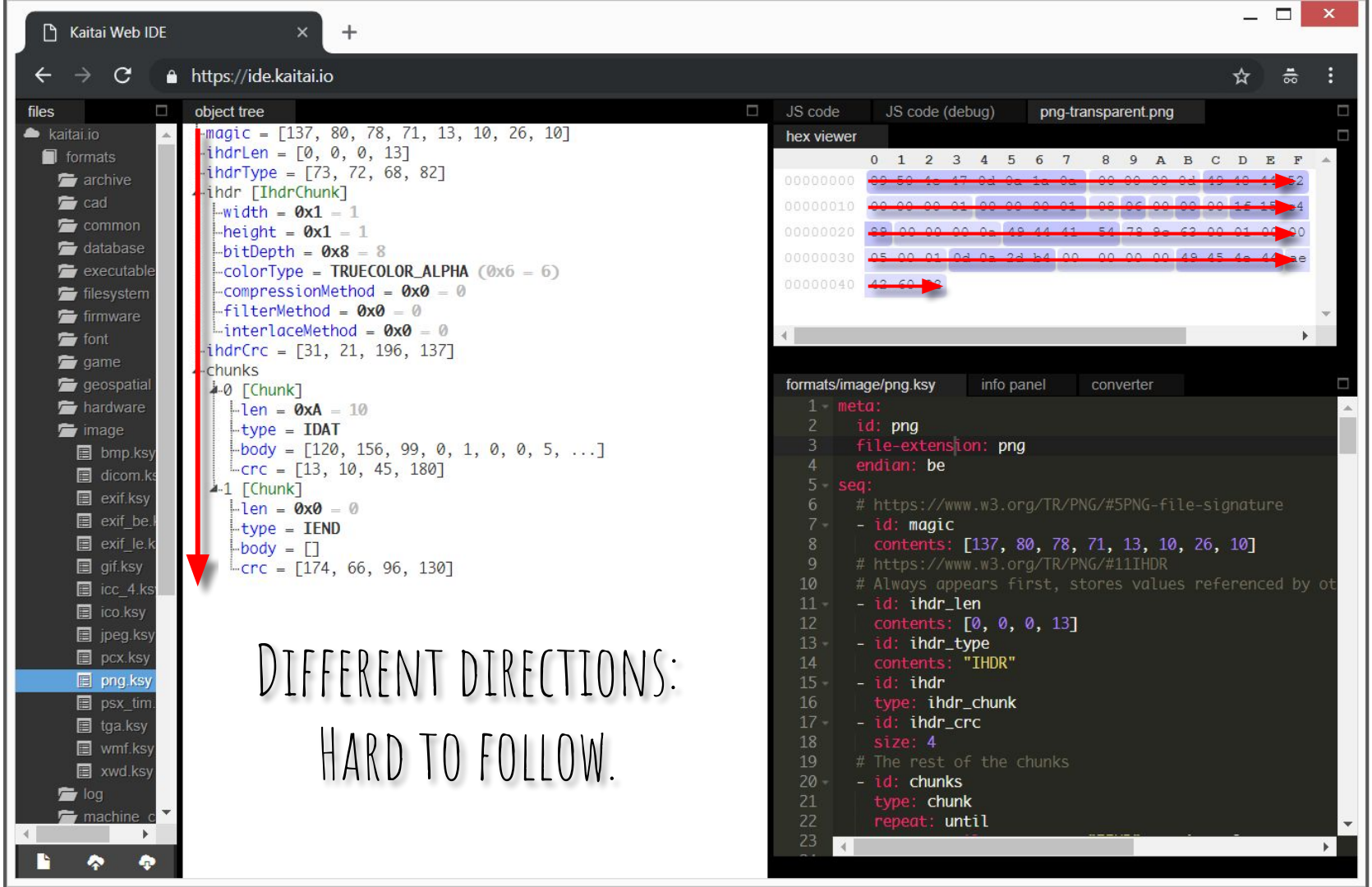
Kaitai - IceBuddha
010 editor - Synalyze


Kaitai Web IDE
https://ide.kaitai.io
files
kaitai.io
formats
archive
cad
common
database
executable
filesystem
firmware
font
game
geospatial
hardware
image
bmp.ksy
dicom.ks
exif.ksy
exif_be.k
exif_le.k
gif.ksy
icc_4.ks
ico.ksy
jpeg.ksy
pcx.ksy
png.ksy
psx_tim.
tga.ksy
wmf.ksy
xwd.ksy
log
machine_c
object tree
magic = [137, 80, 78, 71, 13, 10, 26, 10]
ihdrLen = [0, 0, 0, 13]
ihdrType = [73, 72, 68, 82]
ihdr [IhdrChunk]
width = 0x1 = 1
height = 0x1 = 1
bitDepth = 0x8 = 8
colorType = TRUECOLOR_ALPHA (0x6 = 6)
compressionMethod = 0x0 = 0
filterMethod = 0x0 = 0
interlaceMethod = 0x0 = 0
ihdrCrc = [31, 21, 196, 137]
chunks
0 [Chunk]
len = 0xA = 10
type = IDAT
body = [120, 156, 99, 0, 1, 0, 0, 5, ...]
crc = [13, 10, 45, 180]
1 [Chunk]
len = 0x0 = 0
type = IEND
body = []
crc = [174, 66, 96, 130]
JS code
JS code (debug)
png-transparent.png
hex viewer
0 1 2 3 4 5 6 7 8 9 A B C D E F
00000000 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52
00000010 00 00 00 01 00 00 00 01 08 06 00 00 00 1f 15 c4
00000020 89 00 00 00 0a 49 44 41 54 78 9c 63 00 01 00 00
00000030 05 00 01 0d 0a 2d b4 00 00 00 00 49 45 4e 44 ae
00000040 42 60 82
formats/image/png.ksy
info panel
converter
1 meta:
2 id: png
3 file-extension: png
4 endian: be
5 seq:
6 # https://www.w3.org/TR/PNG/#5PNG-file-signature
7 - id: magic
8 contents: [137, 80, 78, 71, 13, 10, 26, 10]
9 # https://www.w3.org/TR/PNG/#11IHDR
10 # Always appears first, stores values referenced by ot
11 - id: ihdr_len
12 contents: [0, 0, 0, 13]
13 - id: ihdr_type
14 contents: "IHDR"
15 - id: ihdr
16 type: ihdr_chunk
17 - id: ihdr_crc
18 size: 4
19 # The rest of the chunks
20 - id: chunks
21 type: chunk
22 repeat: until
23
THE AMAZING KAITAI IDE
(SERIOUSLY, DON'T MISS IT)

Kaitai

+ awesome TBH. Many grammars.
+ A Kaitai grammar is easier/faster than most specs.

- A grammar might differ from another parser
- Still a generic view.
- nothing for editing.


Kaitai Web IDE
https://ide.kaitai.io
files
kaitai.io
formats
archive
cad
common
database
executable
filesystem
firmware
font
game
geospatial
hardware
image
bmp.ksy
dicom.ks
exif.ksy
exif_be.k
exif_le.k
gif.ksy
icc_4.ks
ico.ksy
jpeg.ksy
pcx.ksy
png.ksy
psx_tim.
tga.ksy
wmf.ksy
xwd.ksy
log
machine_c
object tree
magic = [137, 80, 78, 71, 13, 10, 26, 10]
ihdrLen = [0, 0, 0, 13]
ihdrType = [73, 72, 68, 82]
ihdr [IhdrChunk]
width = 0x1 = 1
height = 0x1 = 1
bitDepth = 0x8 = 8
colorType = TRUECOLOR_ALPHA (0x6 = 6)
compressionMethod = 0x0 = 0
filterMethod = 0x0 = 0
interlaceMethod = 0x0 = 0
ihdrCrc = [31, 21, 196, 137]
chunks
0 [Chunk]
len = 0xA = 10
type = IDAT
body = [120, 156, 99, 0, 1, 0, 0, 5, ...]
crc = [13, 10, 45, 180]
1 [Chunk]
len = 0x0 = 0
type = IEND
body = []
crc = [174, 66, 96, 130]
JS code
JS code (debug)
png-transparent.png
hex viewer
0 1 2 3 4 5 6 7 8 9 A B C D E F
formats/image/png.ksy
info panel
converter
1 meta:
2 id: png
3 file-extension: png
4 endian: be
5 seq:
6 # https://www.w3.org/TR/PNG/#5PNG-file-signature
7 - id: magic
8 contents: [137, 80, 78, 71, 13, 10, 26, 10]
9 # https://www.w3.org/TR/PNG/#11IHDR
10 # Always appears first, stores values referenced by ot
11 - id: ihdr_len
12 contents: [0, 0, 0, 13]
13 - id: ihdr_type
14 contents: "IHDR"
15 - id: ihdr
16 type: ihdr_chunk
17 - id: ihdr_crc
18 size: 4
19 # The rest of the chunks
20 - id: chunks
21 type: chunk
22 repeat: until
23
DIFFERENT DIRECTIONS:
HARD TO FOLLOW.

Hex + ASCII = HexII

```
00: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  ëPNG♪◙→◙    ♪IHDR
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4     ☺    ☺◘♠    ▼§─
20: 89 00 00 00 0A 49 44 41 54 78 9C 63 00 01 00 00  ë    ◙IDATx£c ☺
30: 05 00 01 0D 0A 2D B4 00 00 00 00 49 45 4E 44 AE  ♣ ☺♪◙-┤     IEND«
40: 42 60 82                                         B`é
```

```
00: 89 .P .N .G 0D 0A 1A 0A 00 00 00 0D .I .H .D .R
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4
20: 89 00 00 00 0A .I .D .A .T .x 9C .c 00 01 00 00
30: 05 00 01 0D 0A 2D B4 00 00 00 00 .I .E .N .D AE
40: .B .` 82
```

Insert **ASCII** in the hexadecimal

HexII is still a generic view

```
00: 89 .P .N .G 0D 0A 1A 0A 00 00 00 0D .I .H .D .R
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4
20: 89 00 00 00 0A .I .D .A .T .x 9C .c 00 01 00 00
30: 05 00 01 0D 0A 2D B4 00 00 00 00 .I .E .N .D AE
40: .B .` 82
```

It's more compact, we don't have to switch back and forth,
but it's still stupid. Some ASCII is misleading and also some Hex.

Next step: connect with a parser to indicate things.

```
00: 89 .P .N .G \r \n ^Z \n 00 00 00 0D .I .H .D .R
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4
20: 89 00 00 00 0A .I .D .A .T 78 9C 63 00 01 00 00
30: 05 00 01 0D 0A 2D B4 00 00 00 00 .I .E .N .D AE
40: 42 60 82

ASCII (signature/types)
LENGTH CRC
```

The parser says what's ASCII or not, and indicates boundaries.

Feels like Associated OHA.

Why wrap at 0x10?

```
00: 89 .P .N .G \r \n ^Z \n

08: 00 00 00 0D .I .H .D .R 00 00 00 01 00 00 00 01 08 06 00 00 00
1D: 1F 15 C4 89

21: 00 00 00 0A .I .D .A .T 78 9C 63 00 01 00 00 05 00 01
33: 0D 0A 2D B4

37: 00 00 00 00 .I .E .N .D
3F: AE 42 60 82
```

ASCII (signature/types)
LENGTH CRC

Suddenly, the structure becomes really obvious.

But it lacks continuity between lines.

Let's create a hex tool?

The usual logic is : File -> hextool -> hex view

-> the 'view' restricted to a GUI/CLI :(

-> need new parsers :(

=> File -> toolS -> rendererS-> (reusable) hex viewS

Cf:
https://github.com/kaitai-io/kaitai_struct/issues/143
https://github.com/kaitai-io/kaitai_struct_visualizer/blob/master/bin/ksdump

RENDERINGS

1- TEXT.

2- GRAPHICS.

3- INTERACTION/ANIMATION (NOT YET)

HERE'S WHAT NICE TEXT OUTPUT CAN LOOK LIKE...

```
ab1beefe 10101011000110111110111011111110
ababeeff 10101011101010111110111011111111
aeabeeff 10101110101010111110111011111111
         ======-=-v=vv=^=^XXX=XXX=======^v
ae2b00ff 10101110001010110000000011111111
ab2e00ff 10101011001011100000000011111111
ae2e00fd 10101110001011100000000011111101
         76543210765432107654321076543210
```

https://github.com/samyk/samytools/blob/master/diffbits

👼's SBUD: DISsector & DATa visualiser

DIS: a parser that outputs Dat-compatible Json

DAT: takes Json, renders it

- DatPy -> ANSI output ... -> HTML/RTF/TeX via converters (Ex: AnsiFilter)
- DatJS -> SVG ... -> PDF

(Dis and Dat don't require each other)

https://github.com/corkami/sbud

Warning:

SBuD is

an experimental suggestion :)

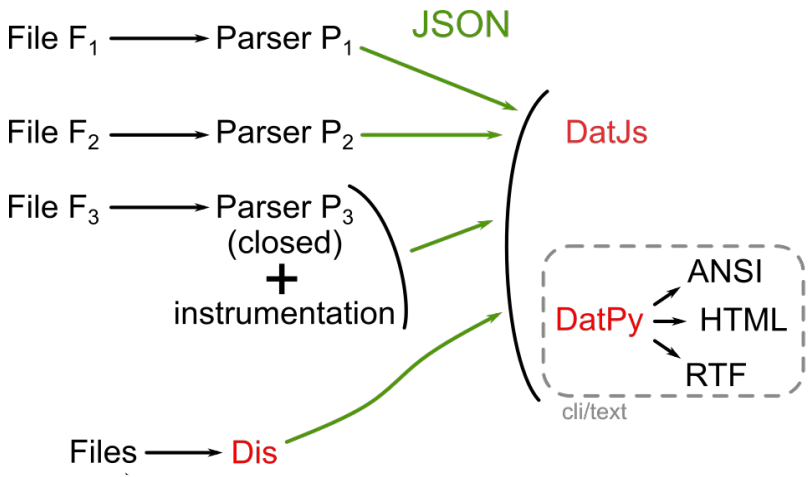
A permanent playground:

"Descriptive" parsers -> Intermediate JSON -> Visualisation

Dis: from File to JSON

Just the basics to slice and describe the hex.

To understand what's needed for Dat.




```
{
 "ASCII": true,
 "name": "signature",
 "offset": 0,
 "size": 8,
 "value": "\\x89PNG\\r\\n\\x1a\\n"
},
{
 "name": "Chunk: Image Header",
 "offset": 8,
 "subEls": [
  {
   "ASCII": false,
   "name": "length",
   "offset": 8,
   "size": 4,
   "value": "13"
  },
  {
   "ASCII": true,
```

```
Type:Png [file]
  000:  89 .P .N .G \r \n 1a \n
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
```

Field	Value
+00 signature	\x89PNG\r\n\x1a\n

```
Chunk: Image Header [chunk]
  000:                          00 00 00 0D .I .H .D .R
  010:  00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
  020:  E3
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
```

Field	Value
+00 length	13
+04 type	IHDR
+15 crc-32	0x948283e3

```
Chunk: Image Data [chunk]
  020:     00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
  030:  00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
  040:  61 E5
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
```

Field	Value
+00 length	21
+04 type	IDAT
+1d crc-32	0xe93261e5

```
Chunk: Image End [chunk]
  040:        00 00 00 00 .I .E .N .D AE 42 60 82
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
```

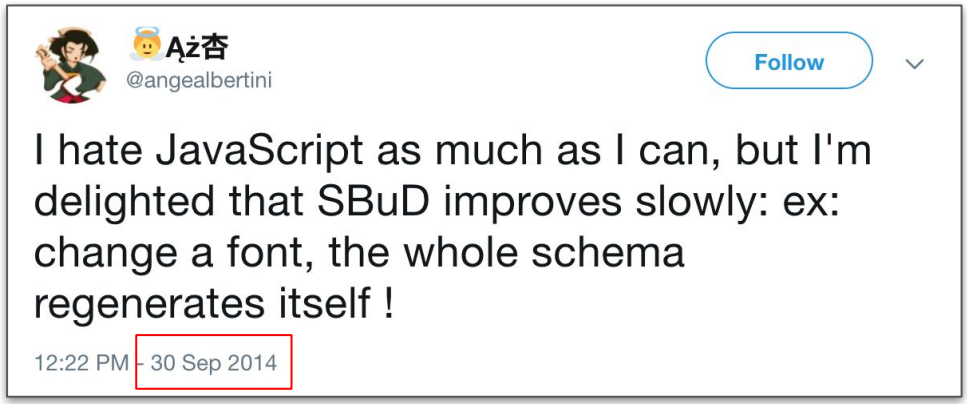
Field	Value
+00 length	0
+04 type	IEND
+08 crc-32	0xae426082

DatPy: from Json to Ansi text

The long quest toward...
(proper)
Visualisation

Tikz? D3 ? Raphael?
Pango+Cairo? Fabric?
(screenshots suck)

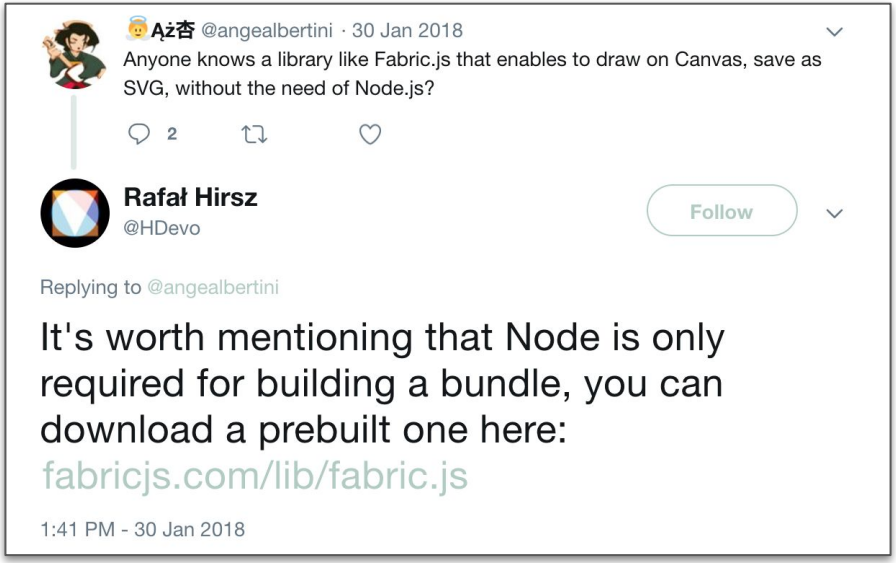
https://twitter.com/angealbertini/status/517031673574477824

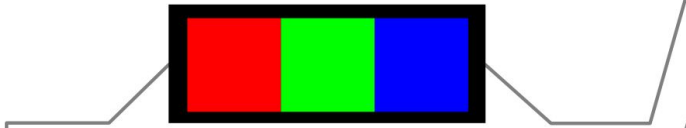


MEETS

A.K.A.

SAVES THE DAY



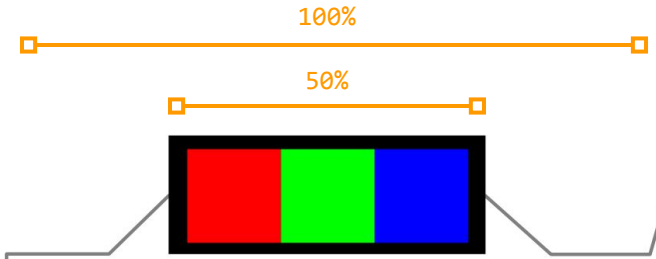


```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 89 .P .N .G 0D 0A 1A 0A 00 00 00 0D .I .H .D .R
10: 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
20: E3 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
30: 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
40: 61 E5 00 00 00 00 .I .E .N .D AE 42 60 82
```

FIELDS	VALUES
SIGNATURE	
signature	\x89 PNG \r\n \x1a \n
HEADER	
size	0x0000000D
id	IHDR
width	0x00000003
height	0x00000001
bpp	0x08
color	0x02 RGB
compression	0x00 DEFLATE
filter	0x00
interlace	0x00
CRC32	0x948283E3
DATA	
size	0x00000015
id	IDAT
ZLIB: window size	0b00001000
ZLIB: method	0b00001000 DEFLATE
ZLIB: level / dict.	0b00011101
ZLIB: checksum	0x081D % 31 = 0
DEFLATE: last block	0b00000001 FINAL
DEFLATE: block type	0b00000001 RAW
DEFLATE: data length	0x000A
DEFLATE: !length	0xFFF5
PIXELS: line filter	0x00 NONE
PIXELS: FF 00 00 00 FF 00 00 00 FF	
ZLIB: adler32	0x0EFB02FE
CRC32	0xE93261E5
END	
size	0x00000000
id	IEND
CRC32	0xAE426082

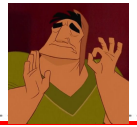
Initial goal

HAS SOME SPECIFIC REQUIREMENTS




```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 89 .P .N .G 0D 0A 1A 0A 00 00 00 0D .I .H .D .R
10: 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
20: E3 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
30: 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
40: 61 E5 00 00 00 00 .I .E .N .D AE 42 60 82
```

FIELDS	VALUES
SIGNATURE = (almost)	\x89 PNG \r\n \x1a \n
HEADER =	
size	0x0000000D
id	IHDR
width	0x00000003
height	0x00000001
bpp	0x08
color	0x02 RGB
compression	0x00 DEFLATE
filter	0x00
interlace	0x00
CRC32	0x948283E3

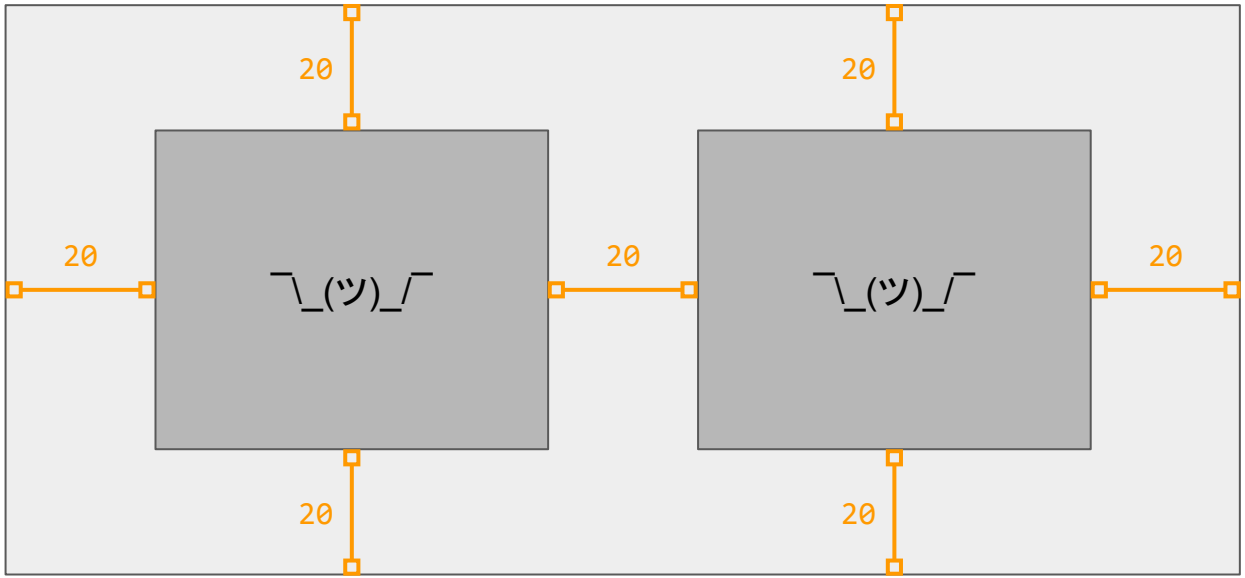


	FIELDS	VALUES
DATA =	size	0x00000015
	id	IDAT
ZLIB	window size	0b00001000
	method	0b00001000 DEFLATE
	level / dict.	0b00011101
	checksum	0x081D % 31 = 0
DEFLATE	last block	0b00000001 FINAL
	block type	0b00000001 RAW
	data length	0x000A
	!length	0xFFF5
PIXELS	line filter	0x00 NONE
	FF 00 00 00 FF 00 00 00 FF	
	adler32	0x0EFB02FE
CRC32		0xE93261E5
END =	size	0x00000000
	id	IEND
CRC32		0xAE426082

The solution: Cassowary

Cassowary is a bird

Cassowary is a ~~bird~~ constraint solving algorithm



More info: https://constraints.cs.washington.edu/cassowary/

Many implementations available

- C, C++, Python, JavaScript, Rust, Nim, Haskell, Swift...
- dedicated community at https://overconstrained.io/

That would settle it, right?

Setting raw constraints is not intuitive

- You manually set (in)equalities between numerical variables
- No visual connection
- It's hard to keep it in your head

Setting raw constraints is not intuitive

```
85      // Add constraints to keep midpoints at line midpoints
86
87      cle = c.Expression.fromConstant(db[0].x).plus(db[1].x).divide(2);
88      cleq = new c.Equation(mp[0].x, cle);
89
90      solver.addConstraint(cleq);
91
92      cle = c.Expression.fromConstant(db[0].y).plus(db[1].y).divide(2);
93      cleq = new c.Equation(mp[0].y, cle);
94
95      solver.addConstraint(cleq);
96
97      cle = c.Expression.fromConstant(db[1].x).plus(db[2].x).divide(2);
98      cleq = new c.Equation(mp[1].x, cle);
99
100     solver.addConstraint(cleq);
101
102     cle = c.Expression.fromConstant(db[1].y).plus(db[2].y).divide(2);
103     cleq = new c.Equation(mp[1].y, cle);
104
105     solver.addConstraint(cleq);
106
107     cle = c.Expression.fromConstant(db[2].x).plus(db[3].x).divide(2);
108     cleq = new c.Equation(mp[2].x, cle);
109
110     solver.addConstraint(cleq);
111
112     cle = c.Expression.fromConstant(db[2].y).plus(db[3].y).divide(2);
113     cleq = new c.Equation(mp[2].y, cle);
114
115     solver.addConstraint(cleq);
```

```
117     cle = c.Expression.fromConstant(db[3].x).plus(db[0].x).divide(2);
118     cleq = new c.Equation(mp[3].x, cle);
119
120     solver.addConstraint(cleq);
121
122     cle = c.Expression.fromConstant(db[3].y).plus(db[0].y).divide(2);
123     cleq = new c.Equation(mp[3].y, cle);
124
125     solver.addConstraint(cleq);
126
127     cle = c.plus(db[0].x, 20);
128
129     solver.addConstraint(new c.Inequality(cle, c.LEQ, db[2].x))
130           .addConstraint(new c.Inequality(cle, c.LEQ, db[3].x));
131
132     cle = c.plus(db[1].x, 20);
133
134     solver.addConstraint(new c.Inequality(cle, c.LEQ, db[2].x))
135           .addConstraint(new c.Inequality(cle, c.LEQ, db[3].x));
136
137     cle = c.plus(db[0].y, 20);
138
139     solver.addConstraint(new c.Inequality(cle, c.LEQ, db[1].y))
140           .addConstraint(new c.Inequality(cle, c.LEQ, db[2].y));
141
142     cle = c.plus(db[3].y, 20);
143
144     solver.addConstraint(new c.Inequality(cle, c.LEQ, db[1].y))
145           .addConstraint(new c.Inequality(cle, c.LEQ, db[2].y));
```

https://github.com/slightlyoff/cassowary.js/blob/master/demos/quad/quaddemo.js

What if we took

Cassowary constraints

and combined them with an

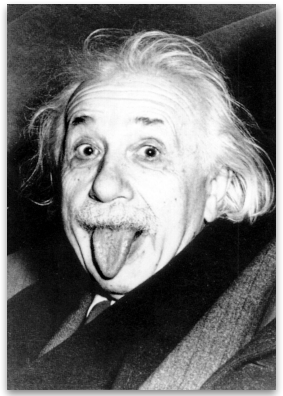
SVG builder?

That's just what 🦌 did!

Meet:

Albert

https://github.com/evoL/albert



Albert

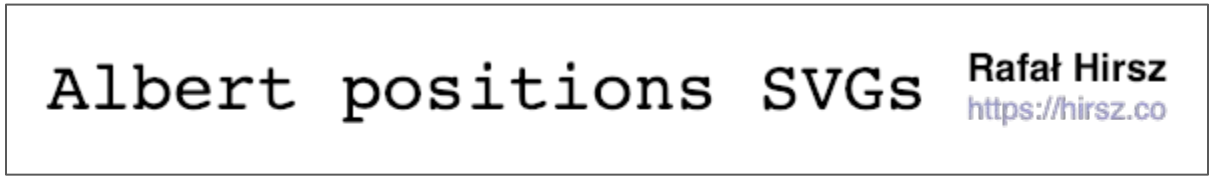
- low-level SVG builder
- Cassowary constraint solving built-in
- JavaScript. Client-side. Standalone.
- One dependency: Cassowary.js

-> browser rendering. -> static SVG.

no CSS/JS for full SVG compatibility -> inkscape and others.

PDF? Just print as PDF in your browser.

How does it work?




```
const rootEl = document.getElementById("svg");
const svg = new albert.Svg(rootEl);
const { align, eq, geq } = albert;

const title = new albert.Text(
  "Albert positions SVGs",
  {"font-family": "monospace"});
const author = new albert.Text("Rafal Hirsz",
  {"font-family": "sans-serif",
   "font-weight": "bold"});
const url = new albert.Text(
  "https://hirsz.co",
  {"font-family": "sans-serif", fill: "#99c"});
const group = new albert.Group([author, url]);

svg.append(title, group);
```

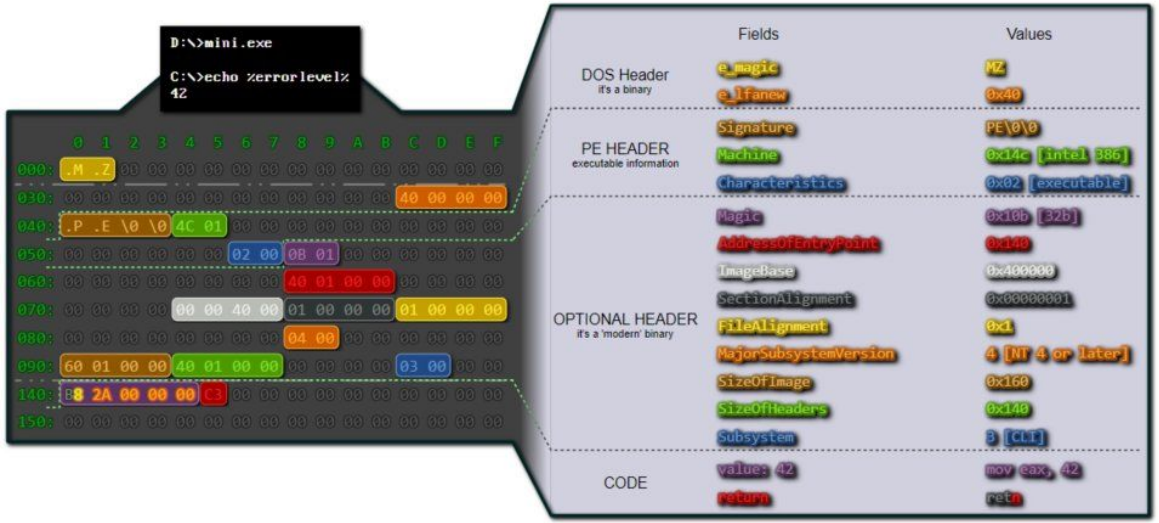
```
svg.constrain(
  // Group constraints
  group
    .spaceVertically()
    .eqAll(child => child.width)
    .forEach(child => geq(child.fontSize, 12))
    .constraints(),
  align(group.rightEdge, svg.rightEdge, -20),
  align(group.topEdge, svg.topEdge, 20),
  // Title constraints
  align(title.leftEdge, svg.leftEdge, 20),
  align(title.topEdge, svg.topEdge, 20),
  align(title.rightEdge, group.leftEdge, -20),
  eq(title.centerY, group.centerY)
);
svg.render();
```

's First Albert tests..

```
D:\>mini.exe

C:\>echo %errorlevel%
42
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000: .M .Z 00 00 00 00 00 00 00 00 00 00 00 00 00 00
030: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00
040: .P .E \0 \0 4C 01 00 00 00 00 00 00 00 00 00 00
050: 00 00 00 00 00 00 02 00 0B 01 00 00 00 00 00 00
060: 00 00 00 00 00 00 00 00 40 01 00 00 00 00 00 00
070: 00 00 00 00 00 00 40 00 01 00 00 00 01 00 00 00
080: 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
090: 60 01 00 00 40 01 00 00 00 00 00 00 03 00 00 00
140: B8 2A 00 00 00 C3 00 00 00 00 00 00 00 00 00 00
150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

	FIELDS	VALUES
DOS HEADER IT'S A BINARY	e_magic	MZ
	e_lfanew	0x40
PE HEADER EXECUTABLE INFORMATION	Signature	PE\0\0
	Machine	0x14c [intel 386]
	Characteristics	0x02 [executable]
OPTIONAL HEADER IT'S A MODERN BINARY	Magic	0x10b [32b]
	AddressOfEntryPoint	0x140
	ImageBase	0x400000
	SectionAlignment	0x00000001
	FileAlignment	0x1
	MajorSubsystemVersion	4 [NT 4 or later]
	SizeOfImage	0x160
	SizeOfHeaders	0x140
	Subsystem	3 [CLI]
CODE	value: 42	mov eax, 42
	return	retn



(They don't scale with file size)

TYPE:PNG

```
 000 89 .P .N .G \r \n 1a \n                                        +00 signature \x89PNG\r\n\x1a\n
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

CHUNK: IMAGE HEADER

```
 000                         00 00 00 0D .I .H .D .R                +00 length 13
 010 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83                +04 type   IHDR
 020 E3                                                             +15 crc-32 0x948283e3
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

CHUNK: IMAGE DATA

```
 020    00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF                +00 length 21
 030 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32                +04 type   IDAT
 040 61 E5                                                          +1d crc-32 0xe93261e5
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

CHUNK: IMAGE END

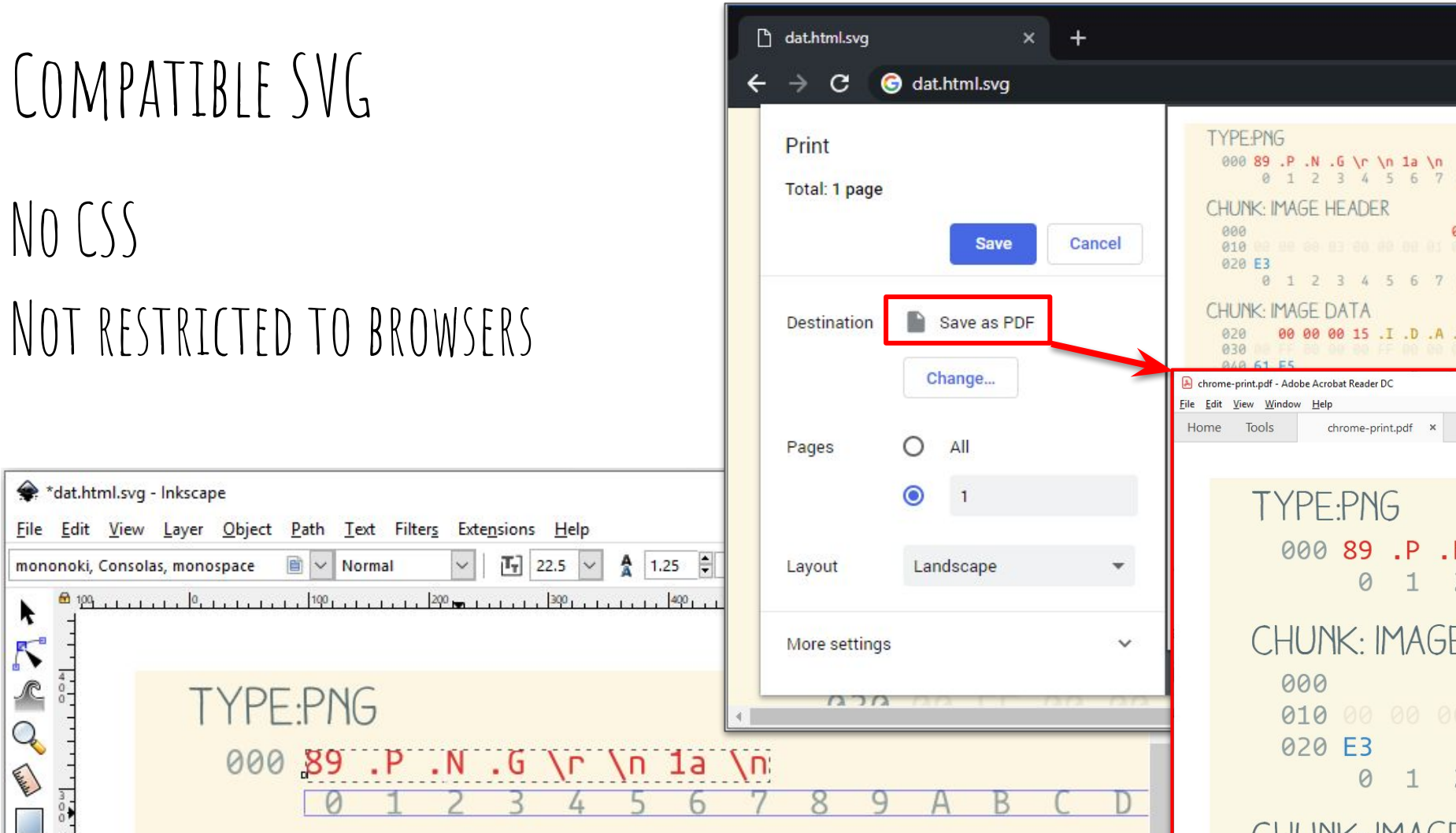
```
 040       00 00 00 00 .I .E .N .D AE 42 60 82                      +00 length 0
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                +04 type   IEND
                                                                    +08 crc-32 0xae426082
```

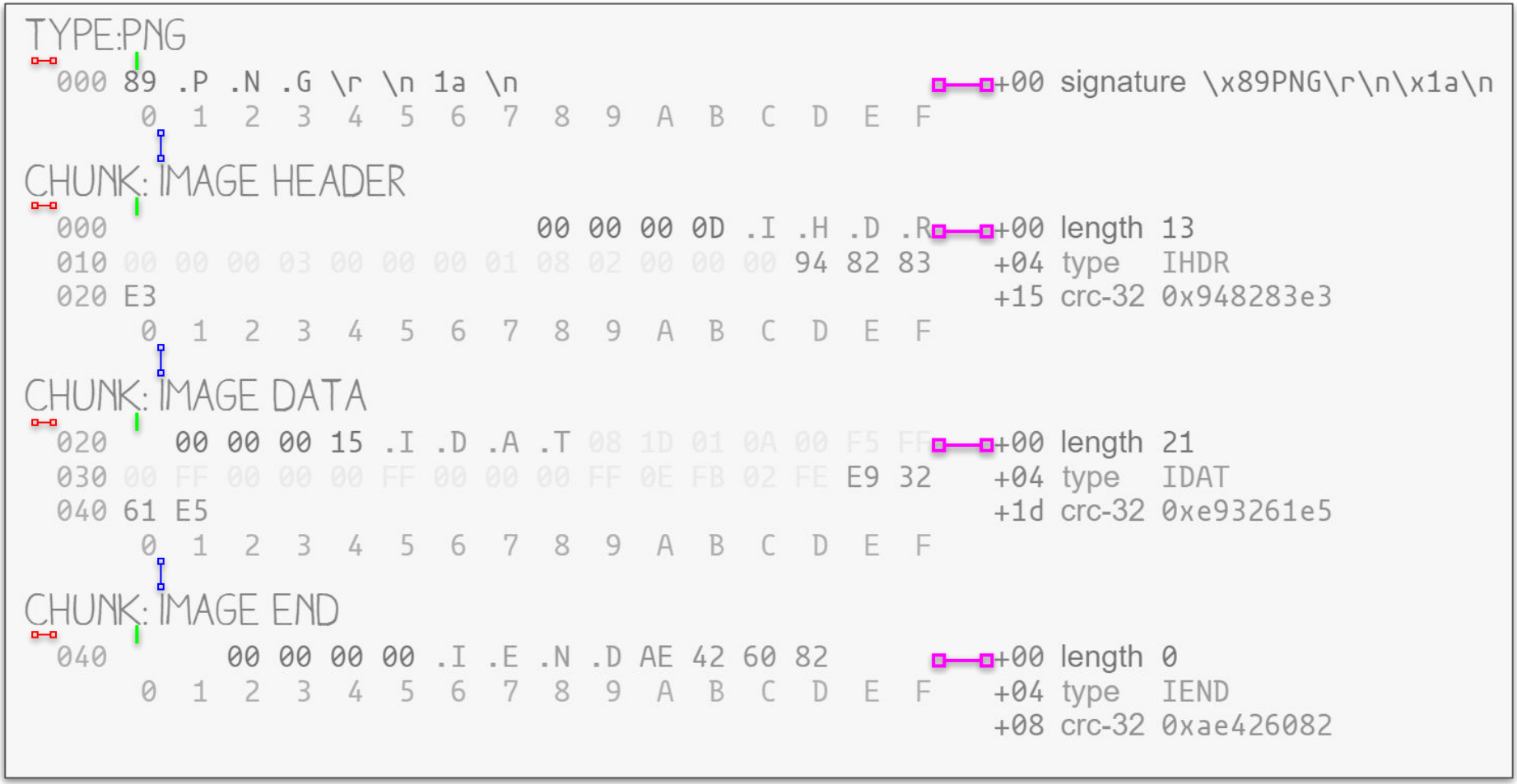
DATJS: FROM JSON TO SVG VIA ALBERT

Compatible SVG

No CSS

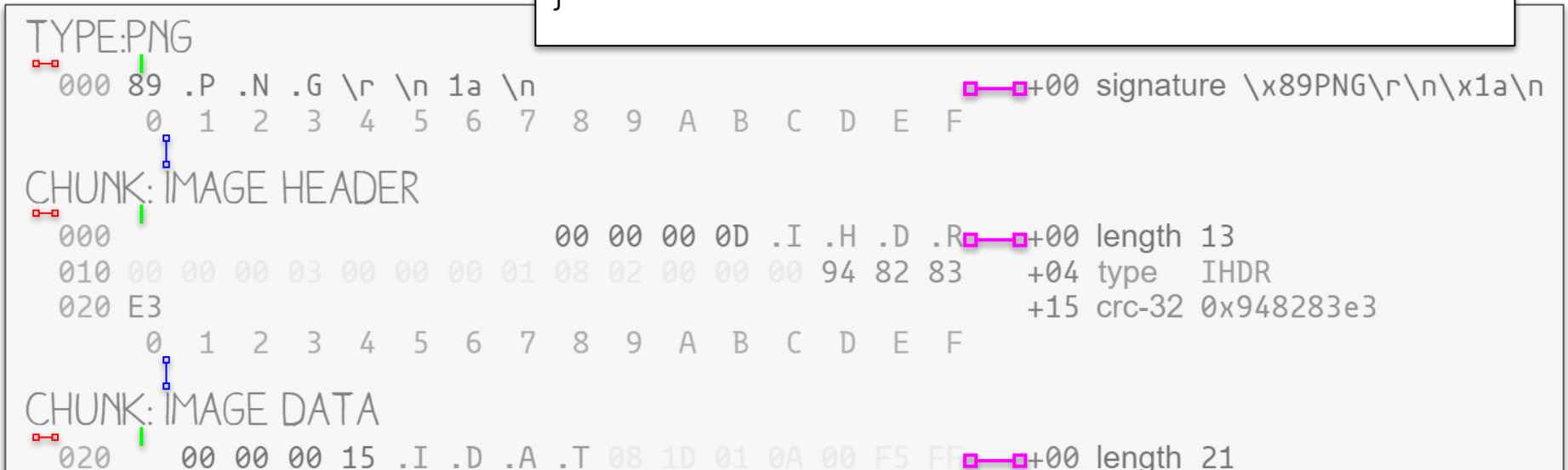
Not restricted to browsers

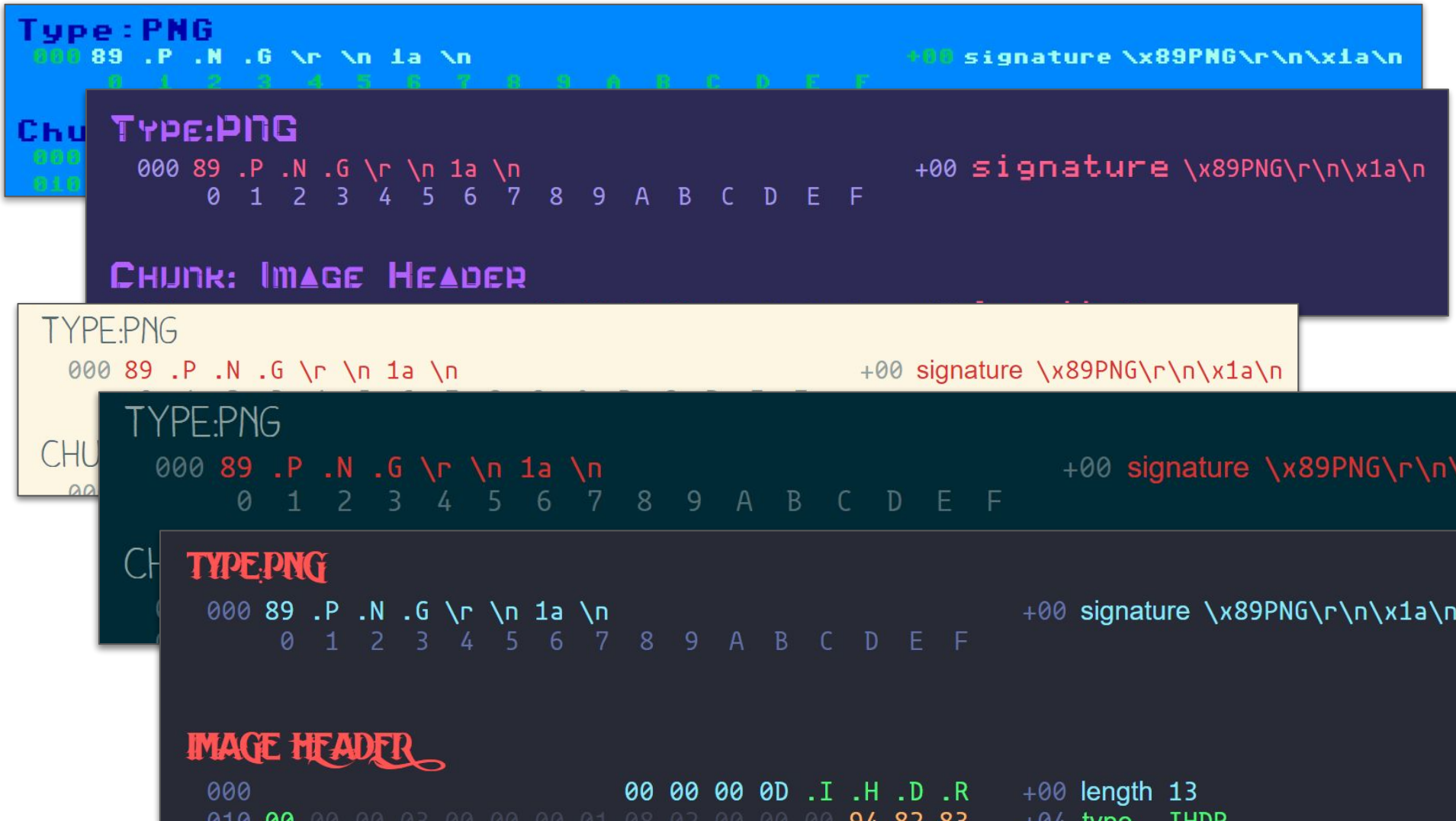



TYPE:PNG
000 89 .P .N .G \r \n 1a \n
+00 signature \x89PNG\r\n\x1a\n
0 1 2 3 4 5 6 7 8 9 A B C D E F
CHUNK: IMAGE HEADER
000 00 00 00 0D .I .H .D .R
010 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
020 E3
+00 length 13
+04 type IHDR
+15 crc-32 0x948283e3
0 1 2 3 4 5 6 7 8 9 A B C D E F
CHUNK: IMAGE DATA
020 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
030 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
040 61 E5
+00 length 21
+04 type IDAT
+1d crc-32 0xe93261e5
0 1 2 3 4 5 6 7 8 9 A B C D E F
CHUNK: IMAGE END
040 00 00 00 00 .I .E .N .D AE 42 60 82
+00 length 0
+04 type IEND
+08 crc-32 0xae426082
0 1 2 3 4 5 6 7 8 9 A B C D E F

Constraints

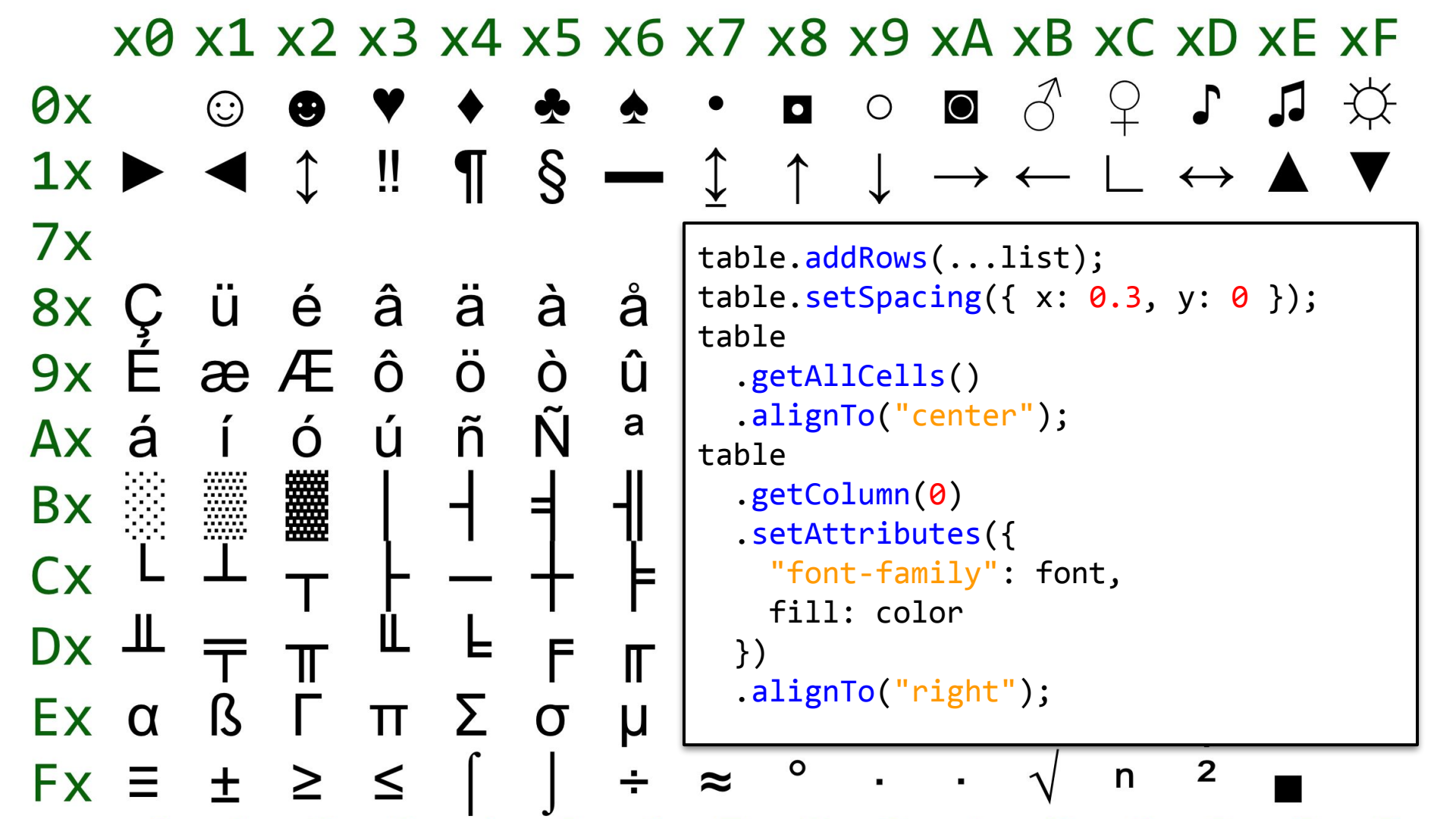
```
for (var i=0; i < fieldvals.length-1; i++) {
  svg.constrain(
    eq(fieldvals[i].topEdge, hexlines[i].topEdge),
    align(hexlines[i].topEdge, headers[i].baseline, .5),
    align(fieldvals[i].leftEdge, hexlines[i].rightEdge, 2),
    align(hexlines[i].leftEdge, headers[i].leftEdge, 2),

    eq(headers[i].leftEdge, headers[i-1].leftEdge),
    align(headers[i].topEdge, hexlines[i-1].bottomEdge, 1),
  );
}
```



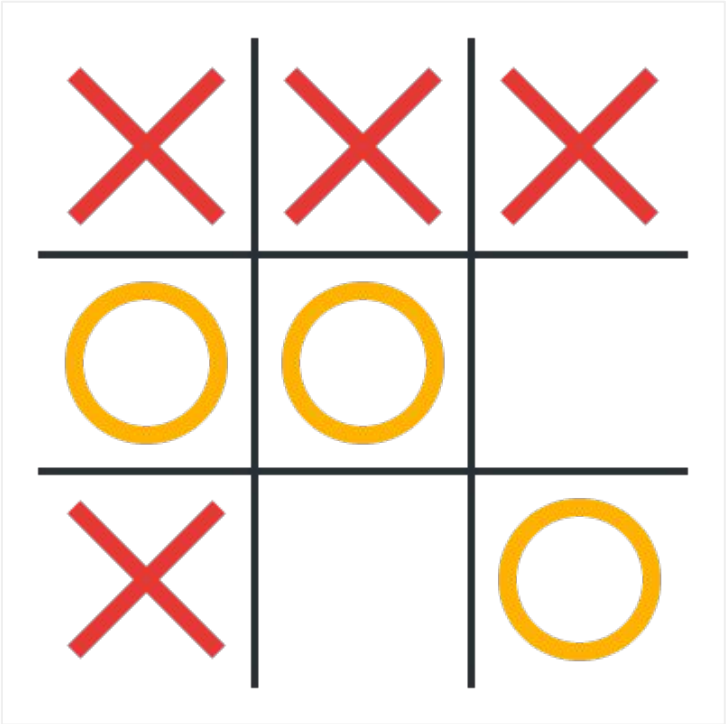

Type:PNG
000 89 .P .N .G \r \n 1a \n
+00 signature \x89PNG\r\n\x1a\n
0 1 2 3 4 5 6 7 8 9 A B C D E F
Chu
000
010
TYPE:PNG
000 89 .P .N .G \r \n 1a \n
+00 signature \x89PNG\r\n\x1a\n
0 1 2 3 4 5 6 7 8 9 A B C D E F
CHUNK: IMAGE HEADER
TYPE:PNG
000 89 .P .N .G \r \n 1a \n
+00 signature \x89PNG\r\n\x1a\n
CHU
TYPE:PNG
000 89 .P .N .G \r \n 1a \n
+00 signature \x89PNG\r\n\
0 1 2 3 4 5 6 7 8 9 A B C D E F
CH
TYPE:PNG
000 89 .P .N .G \r \n 1a \n
+00 signature \x89PNG\r\n\x1a\n
0 1 2 3 4 5 6 7 8 9 A B C D E F
IMAGE HEADER
000 00 00 00 0D .I .H .D .R
+00 length 13

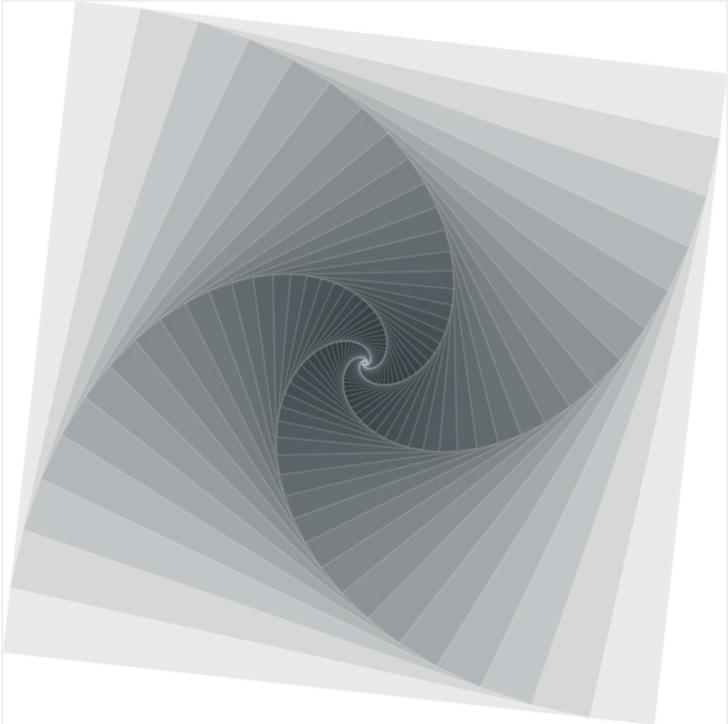
Albert is not just about Hex and hackers

- it was the missing gear in our toolbox.
- "pure" SVG means always compatible.
- Can be applied to many things:
  - diagrams (Syntax, UML...). Diff'ing. Tables.


x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF
0x
1x
7x
8x
9x
Ax
Bx
Cx
Dx
Ex
Fx
table.addRows(...list);
table.setSpacing({ x: 0.3, y: 0 });
table
.getAllCells()
.alignTo("center");
table
.getColumn(0)
.setAttributes({
"font-family": font,
fill: color
})
.alignTo("right");

...OR JUST PRETTY PICTURES





See the examples: https://hirsz.co/albert

Killing the OHA view

👼 In 1999

Emulating arcade video games.

More details @ https://speakerdeck.com/ange/preserving-arcade-games-31c3

Started to patch the best emulator: Callus95

New games

bugfixes

support for controllers

Easter eggs

...

Original:
f8bc1e970a59fab36c50ceb6d52c906da4da98704dcb2dc1ab7026dccab4b3fa cls042w.zip
Final patch:
f4d01d3fdf674d854048164762fa14232bc736b047281d9dd1212f17f35b3e5b cls95p24.zip

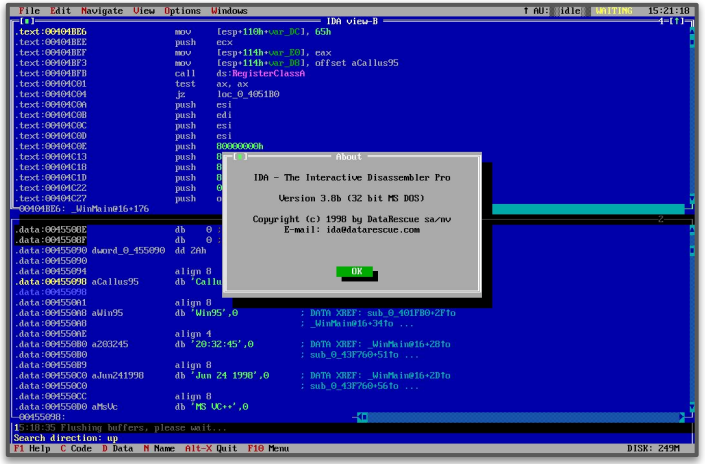
CallusPatch started as a pure hex patch...

The plan worked...

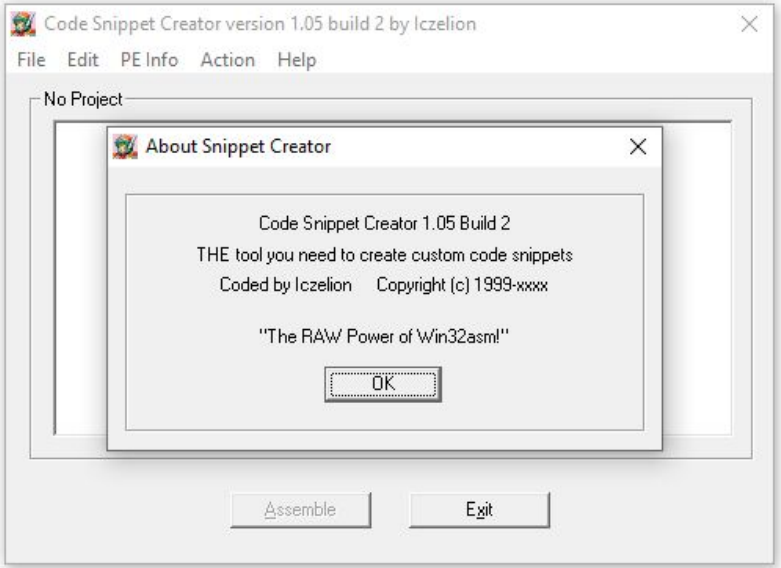
Until an extra element needed to be inserted...

..and re-adjusting too many pointers/lengths/...!

Better solution for a complex project?



Inject a big chunk of ASM



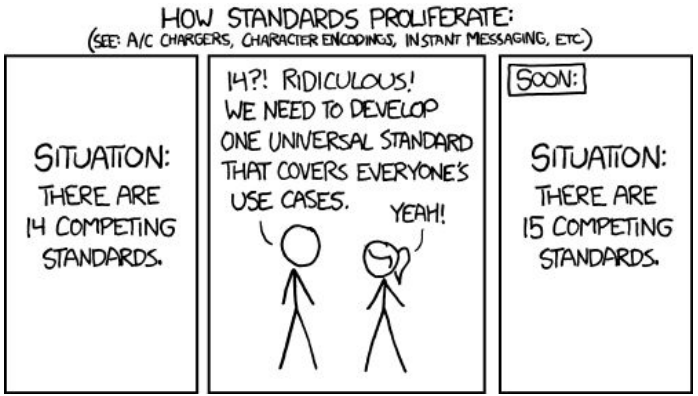

```
section1:00784000                public start
section1:00784000  start         proc near
section1:00784000                mov     ds:off_4394B8, offset aStrider32   ; "strider.32"
section1:0078400A                mov     off_46DBF3, offset dword_78AB38
section1:00784014                mov     off_46DC6B, offset dword_78AB38
section1:0078401E                mov     off_46F9BB, offset sub_78AE3A
section1:00784028                mov     off_46F943, offset sub_78B1A8
section1:00784032                mov     off_4700AB, offset sub_78AFDE
section1:0078403C                mov     ds:byte_402795,  0B8h ; '¸'
section1:00784043                mov     ds:off_402796, offset loc_784DAC
section1:0078404D                mov     ds:dword_40279A,  9090E0FFh
section1:00784057                mov     ds:byte_40279E,  90h
section1:0078405E                mov     ds:off_40277F, offset aExtraSettingsS   ; "\r\n;Extra settings saved by Callus
section1:00784068                mov     ds:byte_438C40,  0B8h ; '¸'
section1:0078406F                mov     ds:off_438C41, offset sub_786DB0
section1:00784079                mov     ds:word_438C45,  0E0FFh
section1:00784082                mov     ds:byte_438D3E,  0B8h ; '¸'
section1:00784089                mov     ds:dword_438D3F, offset sub_786DD3
section1:00784093                mov     ds:dword_438D43,  9090E0FFh
section1:0078409D                mov     ds:byte_438D47,  90h
section1:007840A4                mov     ds:off_43F5D5, offset aParentS   ; "Parent:       %s\r\n"
section1:007840AE                mov     ds:off_438C4D, offset aSsf01   ; "ssf.01"
section1:007840B8                mov     ds:off_438C64, offset aSsfQ01   ; "ssf.q01"
section1:007840C2                mov     ds:off_438C74, offset aSsfQ02   ; "ssf.q02"
section1:007840CC                mov     ds:off_438C9C, offset aSsfQ03   ; "ssf.q03"
section1:007840D6                mov     ds:off_438CB9, offset aSsfQ04   ; "ssf.q04"
section1:007840E0                mov     ds:off_438CC9, offset aSsfQ05   ; "ssf.q05"
section1:007840EA                mov     ds:off_438CF1, offset aSsfQ06   ; "ssf.q06"
section1:007840F4                mov     ds:off_438D0E, offset aSsfQ07   ; "ssf.q07"
section1:007840FE                mov     ds:off_438D1E, offset aSsfQ08   ; "ssf.q08"
section1:00784108                mov     byte_46CCC2,  0BFh ; '¿'
section1:0078410F                mov     byte_46CBDE,  55h ; 'U'
section1:00784116                mov     byte_46CB3C,  5Dh ; ']'
section1:0078411D                mov     byte_46CBBD,  0
section1:00784124                mov     ds:off_43B6AD, offset aKr09Rom   ; "kr_09.rom"
section1:0078412E                mov     ds:off_43B6C4, offset aKr18Rom   ; "kr_18.rom"
section1:00784138                mov     ds:off_43B6D4, offset aKr19Rom   ; "kr_19.rom"
section1:00784142                mov     ds:off_43A994, offset a3wonders18   ; "3wonders.18"
section1:0078414C                mov     ds:off_43A9A4, offset a3wonders19   ; "3wonders.19"

[...]
section1:00785621                mov     dword ptr ds:loc_432E46, offset sub_7866DF
section1:0078562B                mov     dword ptr ds:loc_432E4A,  9090E0FFh
section1:00785635                mov     word ptr ds:loc_432E4A+ 4, 9090h
section1:0078563E                mov     ds:dword_43F6E5,  90909090h
section1:00785648                mov     ds:dword_43F6E9,  90909090h
section1:00785652                mov     ds:dword_43F6ED,  90909090h
section1:0078565C                mov     ds:word_43F6F1,  9090h
section1:00785665                mov     ds:dword_43F6F9,  8B909090h
section1:0078566F                mov     ecx,  56h ; 'V'
section1:00785674                mov     esi, offset off_785687
section1:00785679                mov     edi, offset off_4703E0
section1:0078567E                rep movsd
section1:00785680                mov     eax, offset OEP
section1:00785685                jmp     eax
section1:00785685  start         endp
```

Tool:
002647061c938994c102978523fbe4670c476d96bc1ba2f0f9ed1ad16897e334 sc.zip

Let's be ~~lazy~~ efficient!

Don't reinvent the wheel.

Identify real limitations



We need a language...

- output bytes, data, structures, blocks
- macros, loops...
- standard and available everywhere
- well documented

-> NASM

(no linker, opcodes optional)

https://nasm.us/

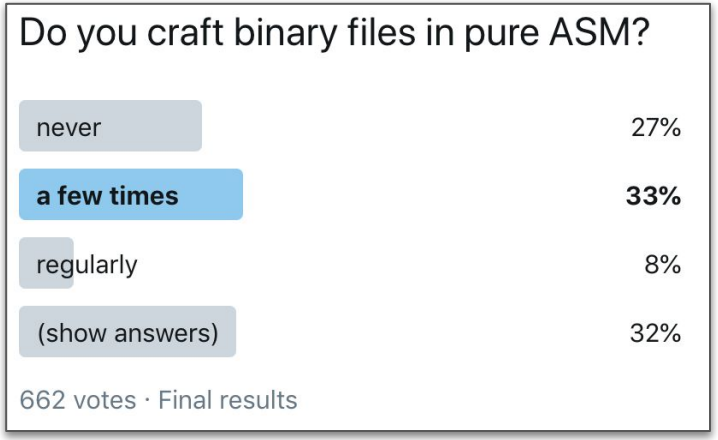
PS: it's not your typical Assembly source.

It's just that its pre-processor is really convenient.

Assembler = crazy?

It's not about talking to a CPU:
Pure ASM is just low-level enough.

http://twitter.com/angealbertini/status/1088866350095835136



Raw ASM == no linker, no code to execute

Empty source -> empty binary.

WebAssembly, for comparison:

```
(module
  (func (result i32) (i32.const 42))
  (export "HelloWorld" (func 0))
)
```

```
$ cat hello.asm
db 'Hello World!'

$ nasm -o hello.bin hello.asm

$ xxd hello.bin
00000000: 4865 6c6c 6f20 576f 726c 6421             Hello World!
```

ASM PRE-PROC 101

```
db 0 → 00
dw 0 → 00 00
dd 0 → 00 00 00 00

label db ‘some string...’, 0Dh
 .LENGTH equ $ - label
```

Defines local `label.LENGTH` as
`<currentaddress> - offset label`

Scope in ASM

```
label:
  .local1 db $
  .local2 db $
  .1.1 db $
  .2.1 db $

label3:
  db $
label.2.end db $
  .1.3 EQU 0x01234
  .1.3.1 db $
label.1.3 EQU 0x05678
```

Source

=>

```
Value     Name
 00001234  label.2.end.1.3
 00005678  label.1.3

Add.  Name
 00    label
 00    label.local1
 01    label.local2
 02    label.1.1
 03    label.2.1
 04    label3
 05    label.2.end
 06    label.2.end.1.3.1
```

Symbols

Binary blobs

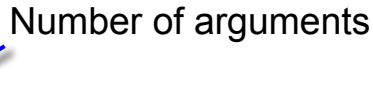
```
incbin “image.png”
```

Includes a binary file → great to ‘collapse’ data

```
incbin <file>
incbin <file>, <start>
incbin <file>, <start>, <size>
```

```
db 89h, “PNG”,0Dh,0Ah,1Ah,0Ah
incbin “image.png”, 8
```

Macros




```
%macro pngsig 0
  db 89h, "PNG",0Dh,0Ah,1Ah,0Ah
%endmacro
```

Big endianness:

```
%macro _dd 1
    db (%1 >> 8 * 3) & 0ffh
    db (%1 >> 8 * 2) & 0ffh
    db (%1 >> 8 * 1) & 0ffh
    db (%1 >> 8 * 0) & 0ffh
%endmacro
```

Add some structure

Blocks of data are **collapsed**.

(no matter their size)

We can **easily** edit, diff...

```
signature db 89h, 'PNG', 0dh, 0ah, 1ah, 0ah

chunk_1:
  .length _dd .crc32 - .data
  .type db 'IHDR'
  .data
    incbin 'png-transparent.png', 010h, 0Dh
  .crc32 _dd 01F15C489h

chunk_2:
  .length _dd .crc32 - .data ; 0ah
  .type db 'IDAT'
  .data
    incbin 'png-transparent.png', 029h, 0Ah
  .crc32 _dd 0D0A2DB4h

chunk_3:
  .length _dd .crc32 - .data
  .type db 'IEND' ; Image End
  .data
  .crc32 _dd 0AE426082h
```

```
00: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  ëPNG♪◙→◙    ♪IHDR
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4     ☺    ☺◘♠    ▼§─
20: 89 00 00 00 0A 49 44 41 54 78 9C 63 00 01 00 00  ë    ◙IDATx£c ☺
30: 05 00 01 0D 0A 2D B4 00 00 00 00 49 45 4E 44 AE  ♣ ☺♪◙-┤     IEND«
40: 42 60 82                                         B`é
```

Annotate with comments

To help with debugging, editing, reminding...

```
;0x0008
chunk_1:
.length _dd .crc32 - .data ; 0dh
.type db 'IHDR' ; Image Header
.data
  incbin 'png-transparent.png', 010h, 0Dh
  ;db '\x00\x00\x00\x01\x00\x00\x00\x01\x08\x06\x00\x00\x00'...
  ;db 0, 0, 0, 1, 0, 0, 0, 1, 8, 6, 0, 0, 0...
.crc32 _dd 01F15C489h

;0x0021
chunk_2:
.length _dd .crc32 - .data ; 0ah
.type db 'IDAT' ; Image Data
[...]
```

Post-processing: Adler, CRCs, Hashes...

Generate symbols: add `[map symbols symbols.map]`

Add specific comments to be parsed

w/ your own post-processor.

```
chunk.1:
.length _dd .crc32 - .data
.type db 'IHDR'
.data
  incbin 'png-transparent.png', 010h, 0Dh
.crc32 _dd 0 ;> chunk.1.crc32=CRC32(chunk.1.type, chunk.1.crc32)
[...]
```

```
- NASM Map file ------------------------------------------------------------

Source file:  structure.asm
Output file:  test.png

-- Symbols -----------------------------------------------------------------

---- Section .text -----------------------------------------------------------

Real              Virtual           Name
               0                 0  signature
               8                 8  chunk.1
               8                 8  chunk.1.length
               C                 C  chunk.1.type
              10                10  chunk.1.data
              1D                1D  chunk.1.crc32
              21                21  chunk.2
              21                21  chunk.2.length
              25                25  chunk.2.type
              29                29  chunk.2.data
              33                33  chunk.2.crc32
              37                37  chunk.3
              37                37  chunk.3.length
              3B                3B  chunk.3.type
              3F                3F  chunk.3.data
              3F                3F  chunk.3.crc32
```

Structures

Fixed size :(

Much clearer - if you need details.

Null values can be skipped.

It just defines a relative offset.

Feel free to abuse.

```
db 0,0,0,1,0,0,0,1,8,6,0,0,0
```

```
; definition

struc IHDR
  .Width        resd 1
  .Height       resd 1
  .Bit_depth    resb 1
  .Color_type   resb 1
  .Compression  resb 1
  .Filter       resb 1
  .Interlace    resb 1
endstruc
```

```
; declaration
istruc IHDR
  at IHDR.Width,          _dd 1
  at IHDR.Height,         _dd 1
  at IHDR.Bit_depth,      _db 8
  at IHDR.Color_type,     _db 6
; at IHDR.Compression,    _db 0
; at IHDR.Filter,         _db 0
; at IHDR.Interlace,      _db 0
iend
```

CONSTANTS

Self explanatory.

```
; constants
compDEFLATE equ 0
[...]
colorRGBA   equ 2
colorALPHA  equ 4
filterNO    equ 0
interlaceNO equ 0
```

```
; declaration
istruc IHDR
  at IHDR.Width,              _dd 1
  at IHDR.Height,             _dd 1
  at IHDR.Bit_depth,          _db 8
  at IHDR.Color_type,         _db colorRGB + colorALPHA
; at IHDR.Compression_method, _db compDEFLATE
; at IHDR.Filter_method,      _db filterNO
; at IHDR.Interlace_method,   _db interlaceNO
iend
```

```
db 0,0,0,1,0,0,0,1,8,6,0,0,0
```

ASM <-> OHA

Compact and meaningful.

Much easier to read, diff, version, tweak...

```
signature db 89h, 'PNG', 0dh, 0ah, 1ah, 0ah

chunk_1:
.length _dd .crc32 - .data
.type db 'IHDR' ; Image Header
.data
  istruc IHDR
    at IHDR.Width,        _dd 1
    at IHDR.Height,       _dd 1
    at IHDR.Bit_depth,    db 8
    at IHDR.Color,        db colorRGBA
    at IHDR.Compression, db compDEFLATE
    at IHDR.Filter,       db filterNO
    at IHDR.Interlace,    db interlaceNO
  iend
.crc32 _dd 01F15C489h ;>.crc32=CRC32(.type,.crc32)

chunk_2:
.length _dd .crc32 - .data
.type db 'IDAT' ; Image Data
.data:
```

```
00: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  ëPNG♪◙→◙    ♪IHDR
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4     ☺    ☺◘♠    ▼§─
20: 89 00 00 00 0A 49 44 41 54 78 9C 63 00 01 00 00  ë    ◙IDATx£c ☺
30: 05 00 01 0D 0A 2D B4 00 00 00 00 49 45 4E 44 AE  ♣ ☺♪◙-┤     IEND«
40: 42 60 82                                         B`é
```

Why not create a new language?

Nasm/Yasm is almost perfect for the task.

Works for most file formats (except text/bit ones)

Some things are missing but it's there to stay.

Weird files in raw ASM (since 2009)

https://github.com/angea/corkami/commit/fd1d9f4c0c171417583d91caf6142905dcabf1ae

```
; a simple PE (a normal PE with many removed elements)

; Ange Albertini, BSD LICENCE 2009-2012

IMAGEBASE equ 400000h
org IMAGEBASE
bits 32

SECTIONALIGN equ 1000h
FILEALIGN equ 200h

%include 'consts.inc'

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; MZ header (start of the file)
istruc IMAGE_DOS_HEADER
    at IMAGE_DOS_HEADER.e_magic, db 'MZ'
    at IMAGE_DOS_HEADER.e_lfanew, dd NT_Signature - IMAGEBASE
iend

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; PE header

NT_Signature:
istruc IMAGE_NT_HEADERS
    at IMAGE_NT_HEADERS.Signature, db 'PE', 0, 0
iend
istruc IMAGE_FILE_HEADER
    at IMAGE_FILE_HEADER.Machine,              dw IMAGE_FILE_MACHINE_I386
    at IMAGE_FILE_HEADER.NumberOfSections,     dw NUMBEROFSECTIONS
    at IMAGE_FILE_HEADER.SizeOfOptionalHeader, dw SIZEOFOPTIONALHEADER
    at IMAGE_FILE_HEADER.Characteristics,      dw IMAGE_FILE_EXECUTABLE_IMAGE | IMAGE_FILE_32BIT_MACHINE
iend
```

```
BITS 32

; Symantec/Norton Antivirus ASPack Remote Heap/Pool memory corruption Vulnerability.
;
; Tavis Ormandy <taviso@google.com>, April 2016
;
; When parsing executables packed by an early version of aspack, a buffer
; overflow can occur in the core Symantec Antivirus Engine used in most Symantec
; and Norton branded Antivirus products. The problem occurs when section data is
; truncated, that is, when SizeOfRawData is greater than SizeOfImage.
;

VirtualAddress      equ 0x10000-0x08    ; VirtualAddress of section data, offset where
SizeOfImage         equ 0x12000-0x0C    ; Size you want to allocate.
SectionPadding      equ 0x2000          ; SizeOfImage-VirtualAddress (but ignoring overh



mzhdr:
        dw "ZM"                         ; e_magic, "MZ" also works.
        dw 0                            ; e_cblp
        dw 0                            ; e_cp
        dw 0                            ; e_crlc
        dw 0                            ; e_cparhdr
        dw 0                            ; e_minalloc
        dw 0                            ; e_maxalloc
        dw 0                            ; e_ss
        dw 0                            ; e_sp
        dw 0                            ; e_csum
        dw 0                            ; e_ip
        dw 0                            ; e_cs
        dw 0                            ; e_lsarlc
        dw 0                            ; e_ovno
        times  4 dw 0                   ; e_res
        dw 0                            ; e_oemid
        dw 0                            ; e_oeminfo
        times 10 dw 0                   ; e_res2
        dd pesig                        ; e_lfanew
pesig:
        dd "PE"
```

PE PoC by Tavis Ormandy

https://googleprojectzero.blogspot.com/2016/06/how-to-compromise-enterprise-endpoint.html

```
; The ultimate abstract ZIP
; by gynvael.coldwind//vx

[bits 32]

; Let's make a ZIP! :)

; Note: how to calculate crc-32? easy! just try to unpack the file
;       with commandline unzip, it shows the good crc ;p

; ---------------------------------------------------------------------

; Loose file FILE HEADER
; This file has no entry in the Central Directory.
; It will be seen only by stream readers, and never by readers that
; interpret ZIP by the book.

file_loose:
dd 0x04034b50               ; local file header signature     4 bytes  (0x04034b50)
dw 0x000a                   ; version needed to extract       2 bytes  (1.0)
dw 0                        ; general purpose bit flag        2 bytes
dw 0                        ; compression method              2 bytes  (0 - store)
dw 0                        ; last mod file time              2 bytes
dw 0                        ; last mod file date              2 bytes
dd 0xe82330fb               ; crc-32                          4 bytes
dd file_loose_data_e - file_loose_data_s ; compressed size                  4 bytes
dd file_loose_data_e - file_loose_data_s ; uncompressed size                4 bytes
dw 17                       ; file name length                2 bytes
dw 0                        ; extra field length              2 bytes
```

ZIP PoC by Gynvael Coldwind

https://gynvael.coldwind.pl/?id=523

POLYGLOTS

CorkaMIX:

1- Concat sources

2- Move elements around (Hide one's into another's)

```
; a binary that is a valid JAR, PE, ZIP, HTML
; mixed version

;Ange Albertini, BSD Licence, 2012

[...]
db 'MZ'
[...]
db '%PDF-1.', 0ah
db 'obj<<>>stream', 0ah

db '<html>'
[...]
db " <script type='text/javascript'>"
db "alert('CorkaMIX [HTML+JavaScript]');</script><!--"
[...]
header:
    db 'PK', 3, 4
    dw 0ah ; version_needed
[...]
_dd 0CAFEBABEh ; signature
_dw 3          ; major version
_dw 2dh        ; minor version
_dw 23         ;constant pool count
[...]
[...]
        db 'endstream', 0ah
        db 'endobj', 0ah
[...]
            _dd 9 ; length of bytecode
                GETSTATIC 8
                LDC 14
                INVOKEVIRTUAL 16
                RETURN
            _dw 0 ; exceptions_count
            _dw 0 ; attributes_count
[...]
```

https://github.com/corkami/pocs/blob/master/poly/CorkaMIX/corkamix.asm

Add a payload in a PNG?

Just copy/paste a chunk structure.

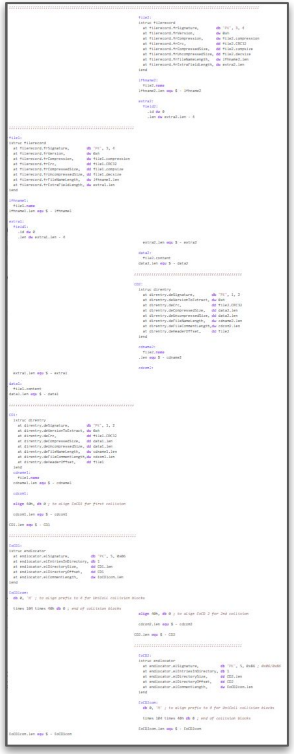
```
chunk:
.length _dd .crc32 - .data
.type db 'fAKE'
.data
  db '"><script>alert(1)</script>'
.crc32 _dd 0 ;> chunk_2.crc32=CRC32(chunk_2.type,chunk_2.crc32)
```

HASH COLLISIONS

http://github.com/corkami/collisions

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

file2:
istruc filerecord
  at filerecord.frSignature,        db "PK", 3, 4
  at filerecord.frVersion,          dw 0ah
  at filerecord.frCompression,      dw file2.compression
  at filerecord.frCrc,              dd file2.CRC32
  at filerecord.frCompressedSize,   dd file2.compsize
  at filerecord.frUncompressedSize, dd file2.decsize
  at filerecord.frFileNameLength,   dw lfhname2.len
  at filerecord.frExtraFieldLength, dw extra2.len
iend

lfhname2:
  file2.name
lfhname2.len equ $ - lfhname2

extra2:
  field2:
    .id dw 0
    .len dw extra2.len - 4
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

file1:
istruc filerecord
  at filerecord.frSignature,        db "PK", 3, 4
  at filerecord.frVersion,          dw 0ah
  at filerecord.frCompression,      dw file1.compression
  at filerecord.frCrc,              dd file1.CRC32
  at filerecord.frCompressedSize,   dd file1.compsize
  at filerecord.frUncompressedSize, dd file1.decsize
  at filerecord.frFileNameLength,   dw lfhname1.len
  at filerecord.frExtraFieldLength, dw extra1.len
iend

lfhname1:
  file1.name
lfhname1.len equ $ - lfhname1
```

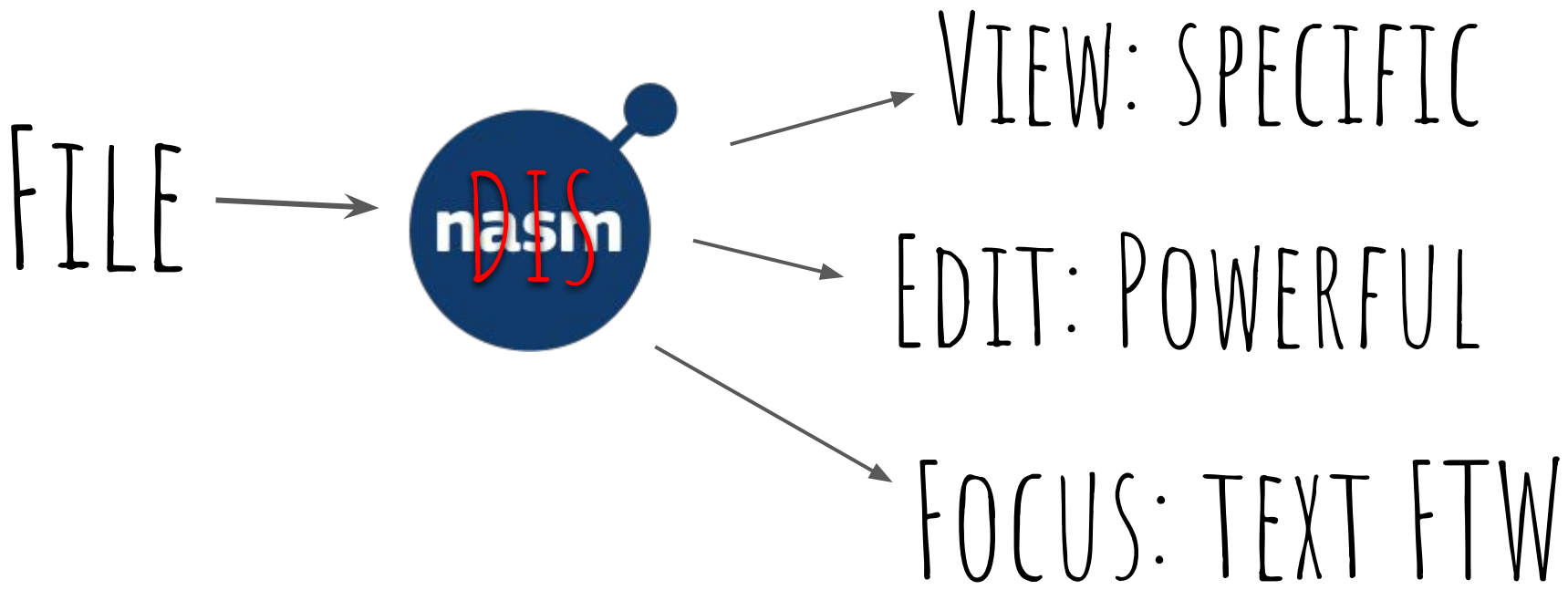


2 STRUCTURES IN PARALLEL:

INDENTATION FTW

```
db `\x89PNG\r\n\x1a\n`                          ; signature                    ;0000: 89 50 4e 47 0d 0a 1a 0a (+8)

chunk1:                                         ; chunk1 { //Image Header
 ddbe 13                                        ;  length                      ;0008: 00 00 00 0d (+4)
;ddbe (chunk1.crc32 - chunk1.data)
.type db `IHDR`                                 ;  type                        ;000c: 49 48 44 52 (+4)
.data:                                          ; Data {
  incbin 'rgb.png', 0x10, 0xd                                                  ;0010: 00 00 00 03 00 00 00 01 08 02 00 00 00 (+
  ;}                                            ;   } //Data
.crc32 ddbe 0x948283e3                          ;  crc-32                      ;001d: 94 82 83 e3 (+4)
 ;> chunk1.crc32=CRC32(chunk1.type,chunk1.crc32)
;}                                              ; } //chunk

chunk2:                                         ; chunk2 { //Image Data
 ddbe 21                                        ;  length                      ;0021: 00 00 00 15 (+4)
;ddbe (chunk2.crc32 - chunk2.data)
.type db `IDAT`                                 ;  type                        ;0025: 49 44 41 54 (+4)
.data:                                          ; Data {
  incbin 'rgb.png', 0x29, 0x15                                                 ;0029: 08 1d 01 0a ..... 00 ff 0e fb 02 fe (+21)
  ;}                                            ;   } //Data
.crc32 ddbe 0xe93261e5                          ;  crc-32                      ;003e: e9 32 61 e5 (+4)
 ;> chunk2.crc32=CRC32(chunk2.type,chunk2.crc32)
;}                                              ; } //chunk

chunk3:                                         ; chunk3 { //Image End
 ddbe 0                                         ;  length                      ;0042: 00 00 00 00 (+4)
;ddbe (chunk3.crc32 - chunk3.data)
.type db `IEND`                                 ;  type                        ;0046: 49 45 4e 44 (+4)
.data:
.crc32 ddbe 0xae426082                          ;  crc-32
 ;> chunk3.crc32=CRC32(chunk3.type,chunk3.crc32)
;}                                              ; } //chunk
```

DIRECT ASM OUTPUT FROM DIS


File
nasm
DIS
View: specific
Edit: Powerful
Focus: text FTW

Conclusion

```
Type:Png [file]                                          Field       Value
  000:  89 .P .N .G \r \n 1a \n                          +00 signature  \x89PNG\r\n\x1a\n
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f

Chunk: Image Header [chunk]                              Field     Value
  000:                        00 00 00 0D .I .H .D .R    +00 length  13
  010:  00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83  +04 type    IHDR
  020:  E3                                               +15 crc-32  0x948283e3
```

```
TYPE:PNG
  000 89 .P .N .G \r \n 1a \n                            +00 signature \x89PNG\r\n\x1a\n
       0  1  2  3  4  5  6  7                  C  D  E  F

CHUNK: IMAGE HEADER
  000                                          .H .D .R  +00 length 13
  010 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83    +04 type   IHDR
  020 E3                                                 +15 crc-32 0x948283e3
       0  1  2  3  4  5  6  7                  C  D  E  F

CHUNK: IMAGE DATA
  020    00 00 00 15 .I .D .A                  00 F5 FF  +00 length 21
  030 00 FF 00 00 00 FF 00 00 00 FF 00 00 00 FE E9 32    +04 type   IDAT
  040 61 E5                                              +1d crc-32 0xe93261e5
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

CHUNK: IMAGE END
  040       00 00 00 00 .I .E .N .D AE 42 60 82          +00 length 0
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    +04 type   IEND
                                                         +08 crc-32 0xae426082
```

```
db `\x89PNG\r\n\x1a\n`                  ; signature                 ;0000: 89 50 4e 47 0d 0a 1a 0a (+8)

chunk1:                                 ; chunk1 { //Image Header
 ddbe 13                                ;  Length                  ;0008: 00 00 00 0d (+4)
;ddbe (chunk1.crc32 - chunk1.data)
.type db `IHDR`                         ;  type                    ;000c: 49 48 44 52 (+4)
.data:                                  ; Data {
  incbin 'rgb.png', 0x10, 0xd           ;                          ;0010: 00 00 00 03 00 00 00 01 08 02 00 00 00 (+13)
  ;}                                    ;   } //Data
.crc32 ddbe 0x948283e3                  ;  crc-32                  ;001d: 94 82 83 e3 (+4)
 ;> chunk1.crc32=CRC32(chunk1.type,chunk1.crc32)
;}                                      ; } //

chunk2:                                 ; ch
 ddbe 21                                ;                          ;0021: 00 00 00 15 (+4)
;ddbe (chunk2.crc32 - chunk2.data)
.type db `IDAT`                         ;                          ;0025: 49 44 41 54 (+4)
.data:                                  ;
  incbin 'rgb.png', 0x29, 0x15          ;                          ;0029: 08 1d 01 0a ..... 00 ff 0e fb 02 fe (+21)
  ;}                                    ;
.crc32 ddbe 0xe93261e5                  ;                          ;003e: e9 32 61 e5 (+4)
 ;> chunk2.crc32=CRC32(chunk2.type,chunk2.crc32)
;}                                      ; }

chunk3:                                 ; chu
 ddbe 0                                 ;  Le                      ;0042: 00 00 00 00 (+4)
;ddbe (chunk3.crc32 - chunk3.data)
.type db `IEND`                         ;  typ                     ;0046: 49 45 4e 44 (+4)
.data:
.crc32 ddbe 0xae426082                  ;  crc-32                  ;004a: ae 42 60 82 (+4)
 ;> chunk3.crc32=CRC32(chunk3.type,chunk3.crc32)
;}                                      ; } //chunk
```

The Offset/Hex/Ascii view is useful

Enhance it with information from parsers.

Improve renderings:

- Text or Graphics
- compatibility
- reusability

```
00: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  ëPNG♪◙→◙    ♪IHDR
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4     ☺    ☺◘♠    ▼§─
20: 89 00 00 00 0A 49 44 41 54 78 9C 63 00 01 00 00  ë...◙IDATx£c ☺
30: 05 00 01 0D 0A 2D B4 00 00 00 00 49 45 4E 44 AE  ♣ ☺♪◙-┤....IEND«
40: 42 60 82                                         B`é
```

```
Type:Png [file]                                                  Field
  000:  89 .P .N .G \r \n 1a \n                                  +00 signatur
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f

Chunk: Image Header [chunk]                                      Field    V
  000:                          00 00 00 0D .I .H .D .R          +00 length
  010:  00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83          +04 type
  020:  E3                                                       +15 crc-32
         0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f

Chunk:
  020:
  030:
  040:

Chunk:
  040:
```

```
TYPE:PNG
  000 89 .P .N .G \r \n 1a \n                                    +00 sign
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

CHUNK: IMAGE HEADER
  000                        00 00 00 0D .I .H .D .R            +00 leng
  010 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83           +04 type
  020 E3                                                         +15 crc-
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

CHUNK: IMAGE DATA
  020    00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF           +00 leng
         00 00 FF 00 00 00 FF 0E FB 02 FE E9 32                 +04 type
                                                                +1d crc-
  2  3  4  5  6  7  8  9  A  B  C  D  E  F

E END
  00 00 00 00 .I .E .N .D AE 42 60 82                           +00 leng
  2  3  4  5  6  7  8  9  A  B  C  D  E  F                     +04 type
                                                                +08 crc-
```

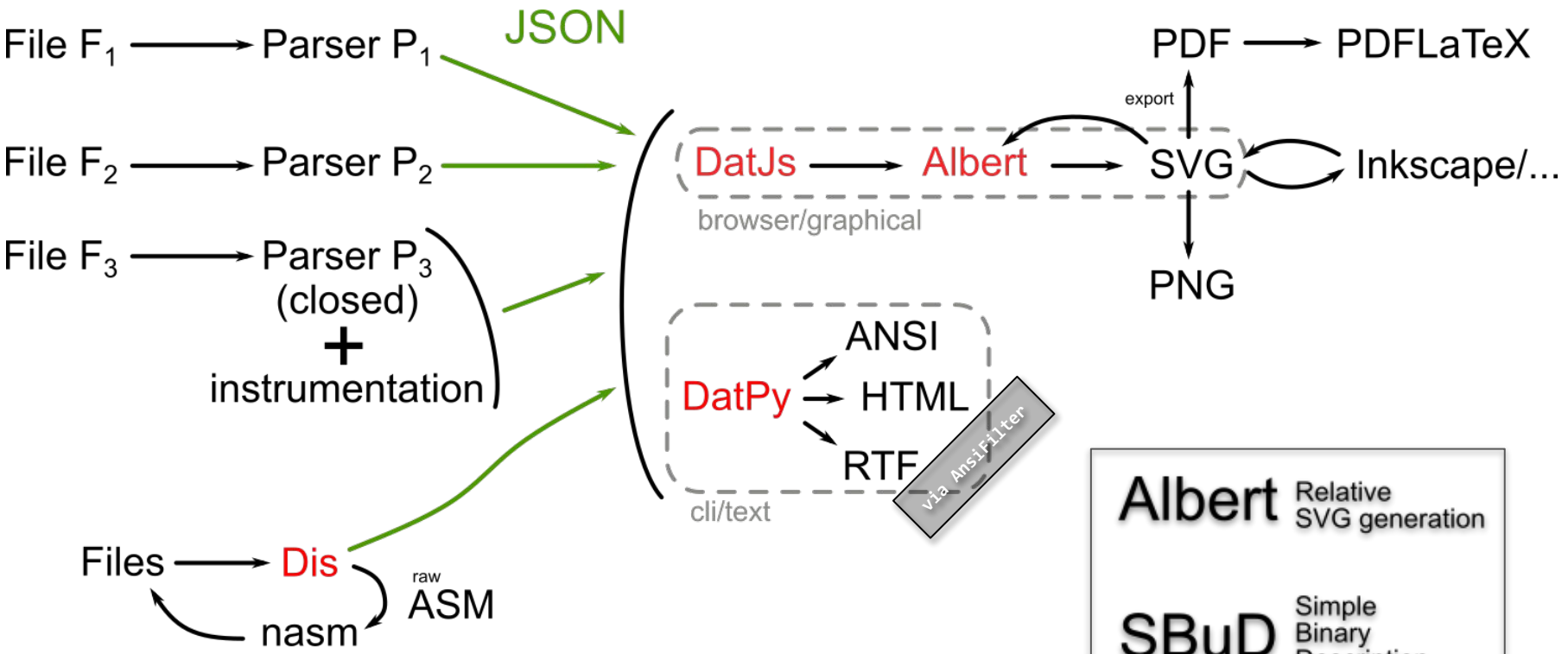
The Offset/Hex/Ascii view is dumb

Dissect binary files as raw assembly:

easy to read, edit or diff.

```
00: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52  ëPNG♪◙→◙    ♪IHDR
10: 00 00 00 01 00 00 00 01 08 06 00 00 00 1F 15 C4     ☺     ☺◘♠    ▼§─
20: 89 00 00 00 0A 49 44 41 54 78 9C 63 00 01 00 00  ë...◙IDATx£c ☺
30: 05 00 01 0D 0A 2D B4 00 00 00 00 49 45 4E 44 AE  ♣ ☺♪◙-┤....IEND«
40: 42 60 82                                         B`é
```

```
signature db 89h, 'PNG', 0dh, 0ah, 1ah, 0ah

chunk_1:
.length _dd .crc32 - .data
.type db 'IHDR' ; Image Header
.data
  istruc IHDR
    at IHDR.Width,       _dd 1
    at IHDR.Height,      _dd 1
    at IHDR.Bit_depth,   db 8
    at IHDR.Color,       db colorRGBA
    at IHDR.Compression, db compDEFLATE
    at IHDR.Filter,      db filterNO
    at IHDR.Interlace,   db interlaceNO
  iend
.crc32 _dd 01F15C489h ;>.crc32=CRC32(.type,.crc32)

chunk_2:
.length _dd .crc32 - .data
.type db 'IDAT' ; Image Data
.data:
                'png-transparent.png', 029h, 0Ah
                dd 0D0A2DB4h ;>.crc32=CRC32(.type,.crc32)


                _dd .crc32 - .data
                'IEND' ; Image End
.data:
.crc32 _dd 0AE426082h ;>.crc32=CRC32(.type,.crc32)
```



Available today: Albert, SBuD [Dis & Dat]

Albert

An SVG library with a twist, tailored for hackers.

Lightweight, low level, client-side, no dependencies.
Constraint-based drawing.

Not limited to hexadecimal representation.

https://github.com/evoL/albert

Personal conclusions - #1

Leaving your comfort zone is **hard**.
But brings exceptional results.

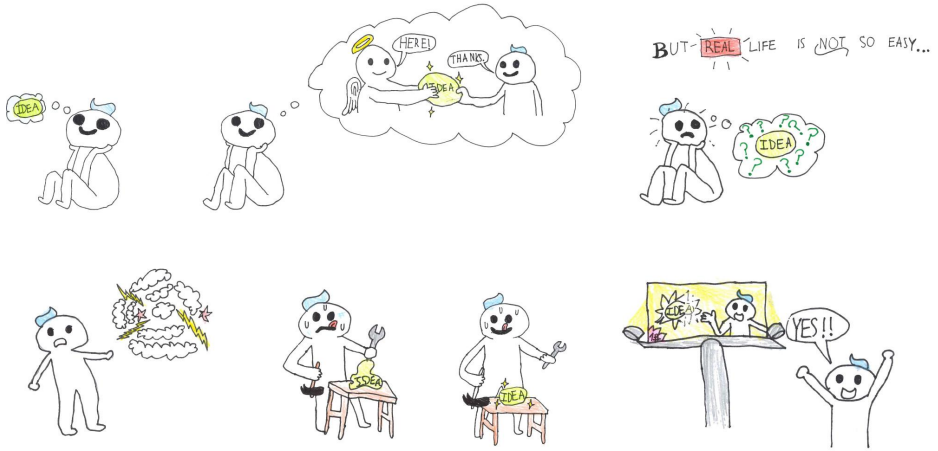
Faster alone, further together.

#2 Keep trying

More tries means more fails, but also more opportunities to explore.

Yes, you're totally hopeless!
(that's what 👼 thought, at least)

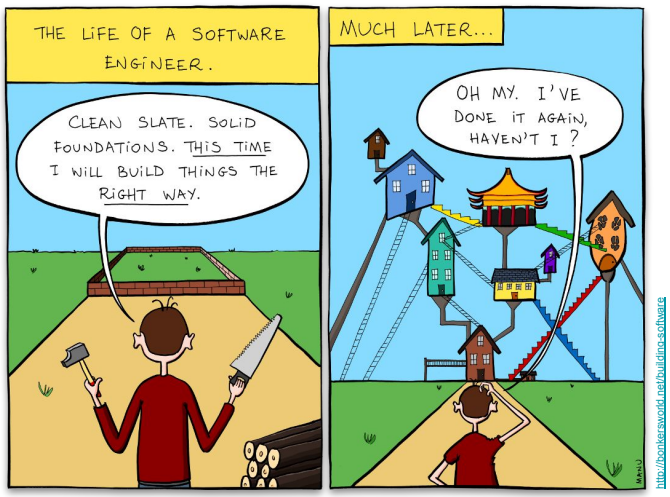
-> **keep** having **fun**,
now that you have nothing to lose!



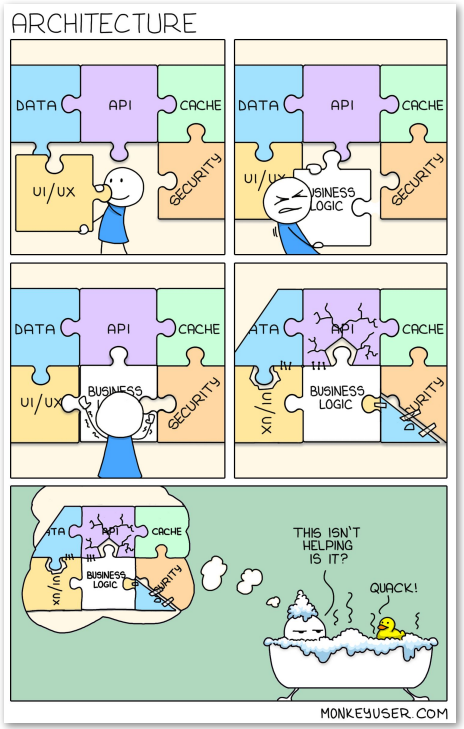
Too small, then too ambitious, then...

Success -> utopia
Standards -> ambitions

3rd attempt is the right one?



http://monkeyswork.net/building-software



https://www.monkeyuser.com/2018/architecture/

https://imgur.com/RaRXx1a.jpg
https://web.archive.org/web/20161121205329/http://owlturd.com/post/154247077094/vertical-square

https://web.archive.org/web/20161220111115/http://owlturd.com/post/154549097274

What's (likely) next?

- Corkami: good file format specs, file corpus.
- SBuD:
  - web interface (interactive dissection w/ rendering = Dis+Dat)
  - Dis: parsers that describe or explain files.
  - Dat: structure maps, dedicated format views, interactivity
- Albert:
  - used in different projects (tables, syntax diagrams, protocols)
  - debuggability and tooling

Thanks!

Feedback?

Rafał Hirsz

ANGE ALBERTINI
reverse engineering
VISUAL DOCUMENTATIONS
@angealbertini
ange@corkami.com
http://www.corkami.com

A RAHAN Production

Acknowledgements: Ero, Luca, 4am, Hector, Daniel, Samy, Miau, Barbie, Phil

Example of a PDF PoC: CVE-2018-5158

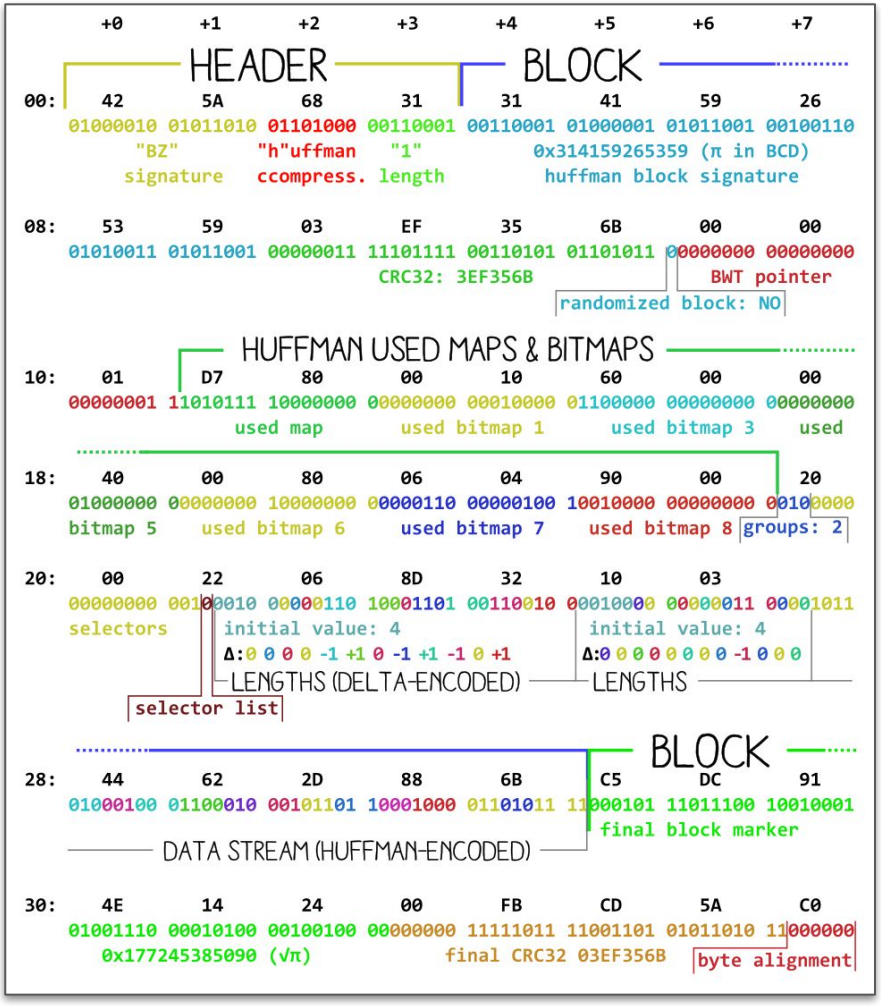
https://bugzilla.mozilla.org/show_bug.cgi?id=1452075

```
[...]
1 0 obj
<<
  /FunctionType 4
  /Domain [(console.log("Hello, this is code running in " + location.href)) 0]
  /Range [0 0]
  /Length 12
>>
stream
{
  0 add
}
endstream
endobj

4 0 obj
[ /Indexed
  [ /DeviceN
    [/Cyan /Black]
    /DeviceCMYK
    1 0 R
  ]
  1(123)
]
endobj

[...]
  /ColorSpace 4 0 R
[...]
```

<= ALL TEXT!

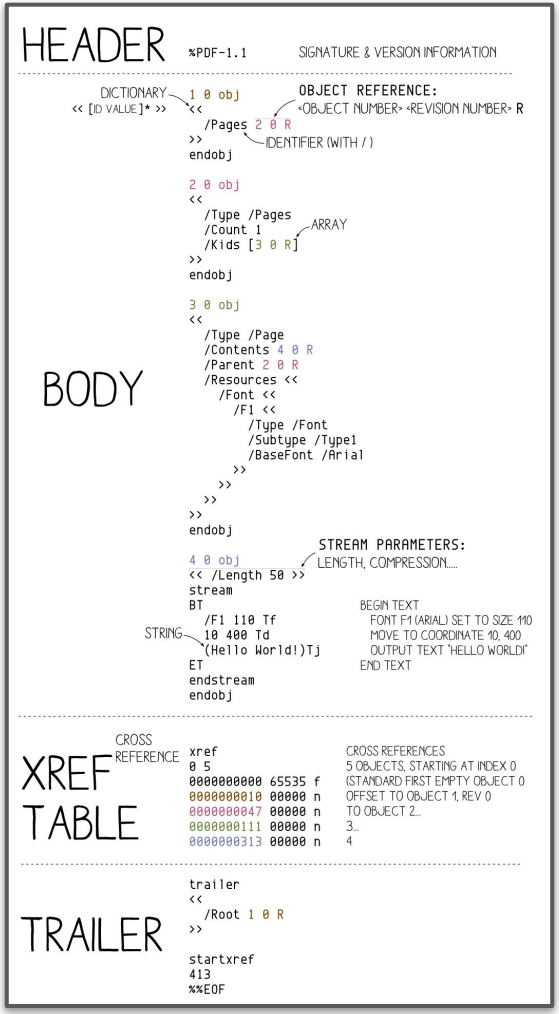
Formats can be *very* different

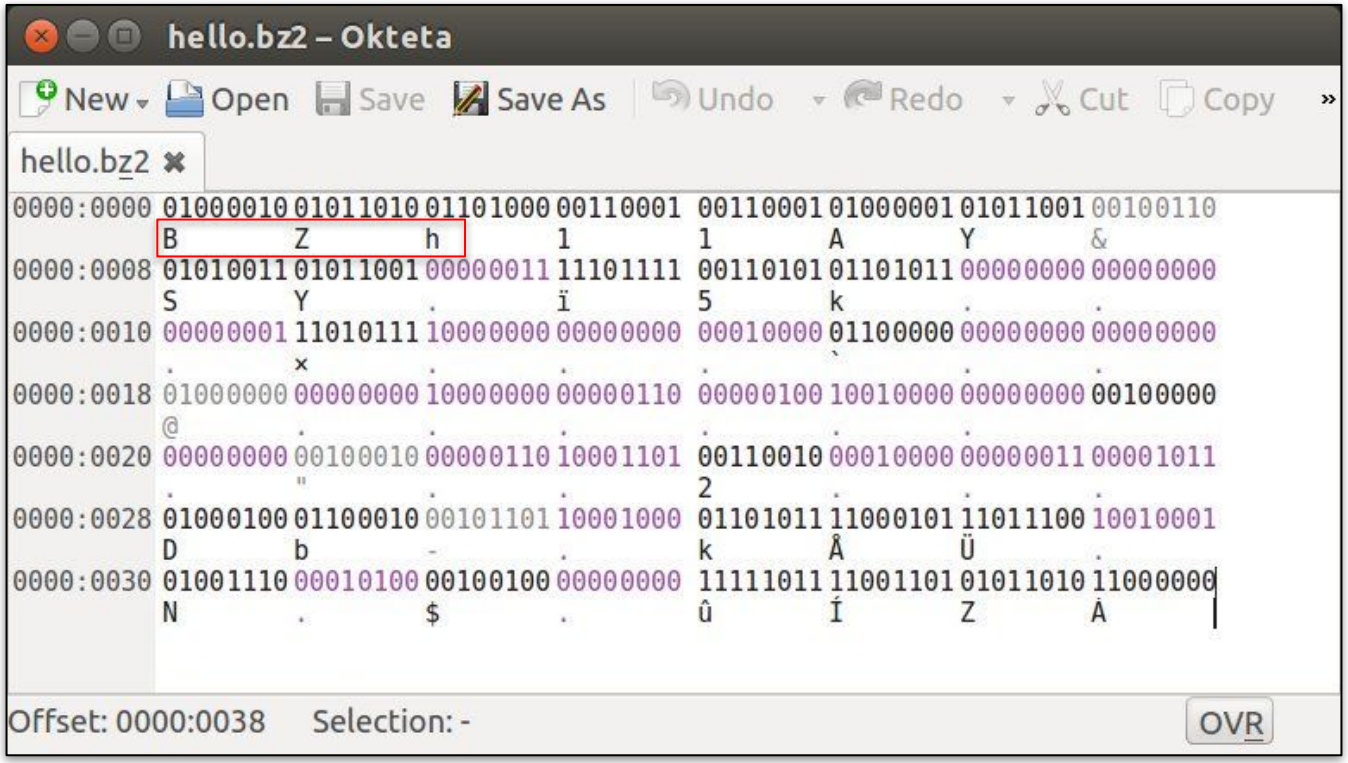
A generic view *can't* be optimal



<- BZIP2
(BIT-BASED)

PDF ->
(TEXT SKELETON)



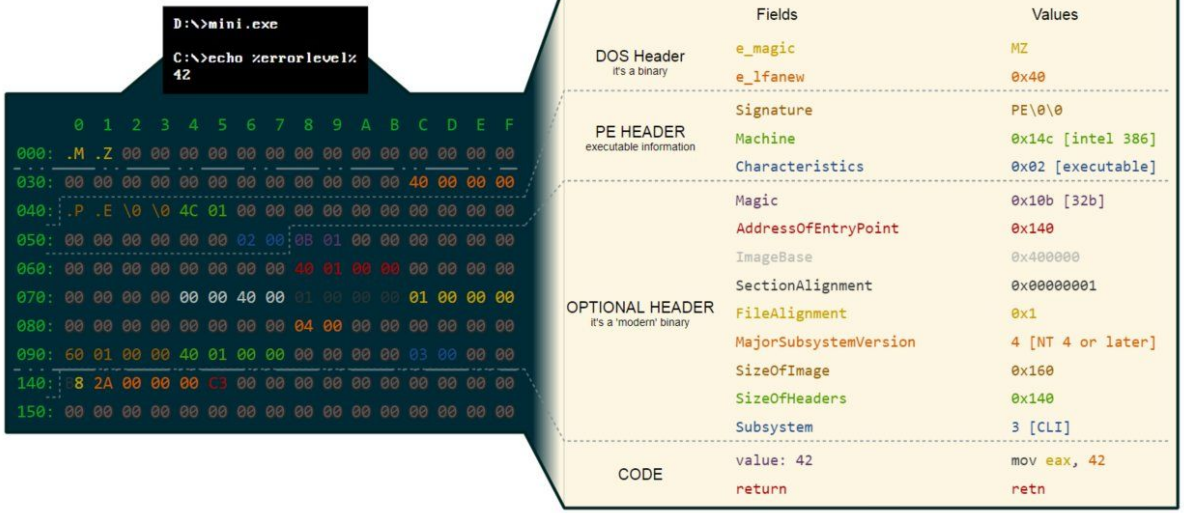


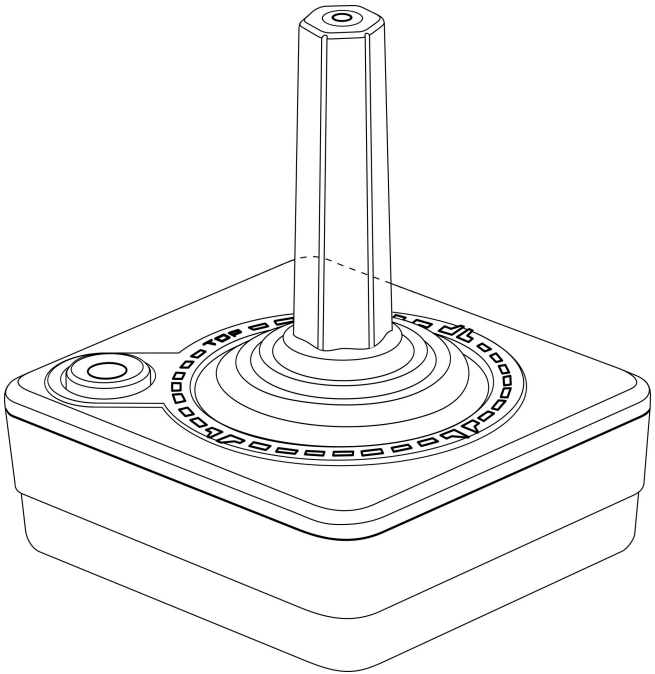
< = THE SAME BZIP2

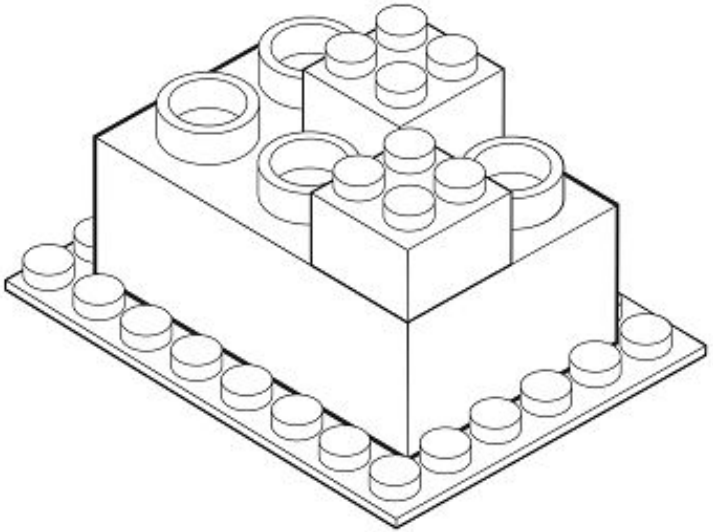
< AS BEFORE

OKTETA HAS A BIT-LEVEL VIEW! BUT IT'S STILL HARD TO EDIT (INSERT, DELETE...)

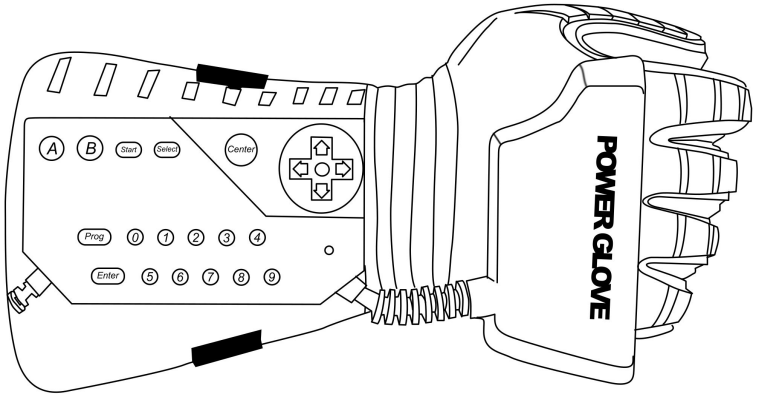
Style drawing is addictive...

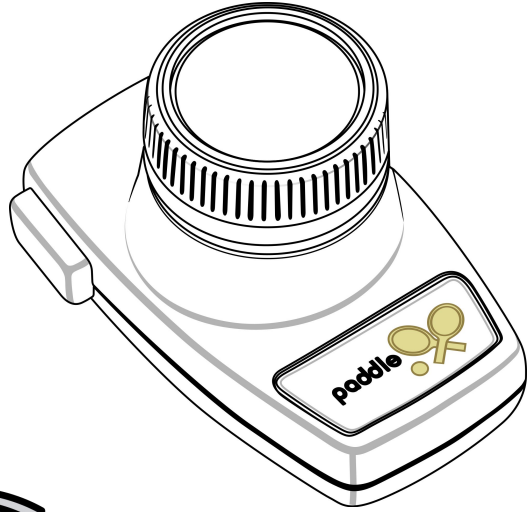


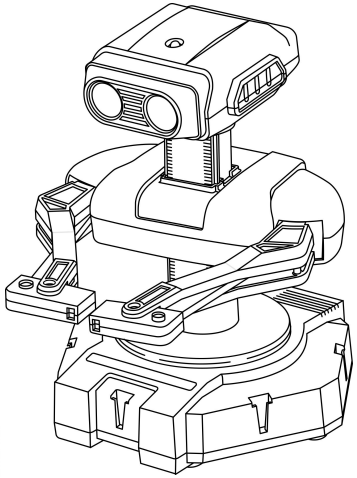


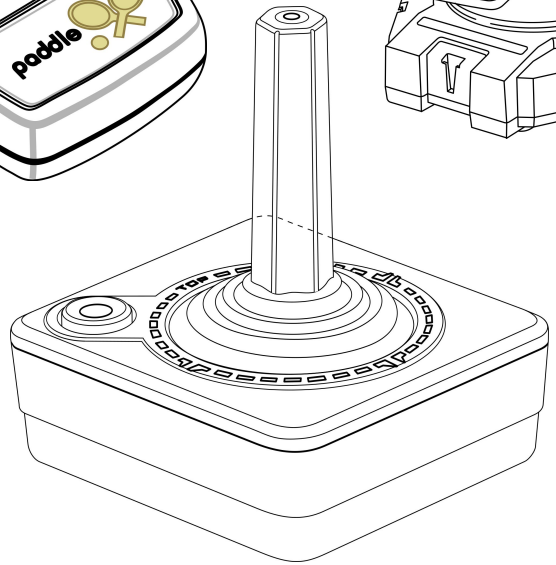


Which ones enables more creativity?









Each libraries comes with controls and restrictions.

What complex file manipulation feels like.

GIF bounty @ facebook

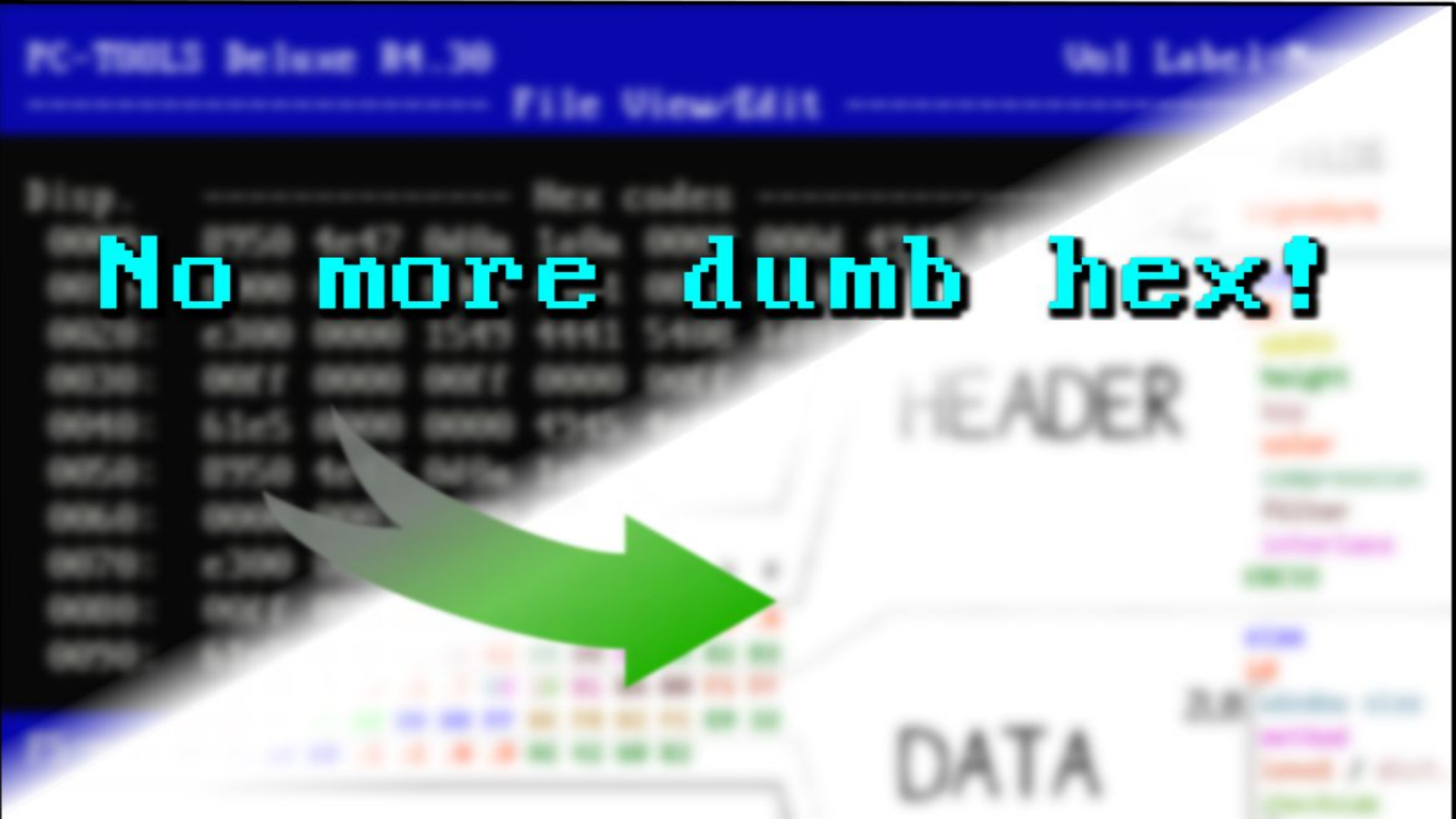
Similar in principles,

but no dissection.

-> ASM is a common ground.

https://www.vulnano.com/2019/03/facebook-messenger-server-random-memory.html

```
import struct

screenWidth = 640
screenHeight = 480

f = open('test.gif', 'wb')

# Offset   Length   Contents
#   0      3 bytes  "GIF"
#   3      3 bytes  "87a" or "89a"
f.write(b"GIF89a")

#   6      2 bytes  <Logical Screen Width>
f.write(struct.pack('<h', screenWidth))

#   8      2 bytes  <Logical Screen Height>
f.write(struct.pack('<h', screenHeight))

#  10      1 byte   bit 0:    Global Color Table Flag (GCTF)
#                   bit 1..3: Color Resolution
#                   bit 4:    Sort Flag to Global Color Table
#                   bit 5..7: Size of Global Color Table: 2^(
bits = int('00000010', 2)
f.write(struct.pack('<b', bits))

#  11      1 byte   <Background Color Index>
f.write(struct.pack('<b', 0))

#  12      1 byte   <Pixel Aspect Ratio>
f.write(struct.pack('<b', 1))
```


No more dumb hex!
HEADER
DATA