Reverse-Engineering of (binary) File-Formats

From seemingly arbitrary zeros and ones to a PCB file

Thomas Pointhuber

Open Source Computer Aided Modeling and Design devroom - FOSDEM'21

My Background

Aug. 2015	my first KiCad contribution
since Jan. 2016	KiCad Library Maintainer Team
since Oct. 2020	KiCad Lead Development Team

Nov 2016	my first security competition
since then	part of the university team[1]

“I’m a Software Engineer with focus on Security”

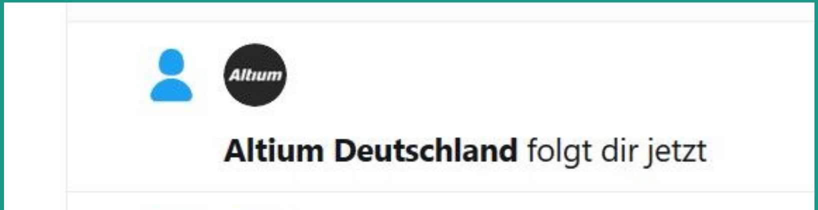
Find a project where I can combine those two worlds:

Reverse-Engineering the ~~Allegro~~ Altium file format

and write a KiCad importer!

[1] https://www.sigflag.at

General Background



Altium Deutschland folgt dir jetzt

they unfollowed, perhaps too many KiCad tweets :D

@Chaos_Robotic

Step 0: Legal Bases

We want to figure out how a proprietary file formats works.

Companies may have something against that work.

Better be safe than sorry.

**Law differs by country and change over time.
For reliable statements contact a local lawyer.**

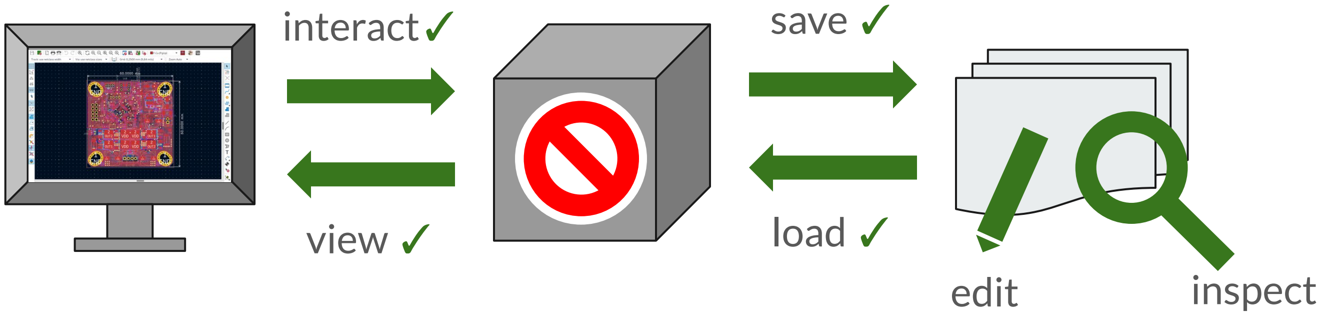
Use those informations at your own risk!

Step 0: Legal Bases [Reverse-Engineering]

Black-Box

Reverse-Engineering

"*__usually__, you are allowed to observe what a program does*"



interact ✓

view ✓

save ✓

load ✓

edit

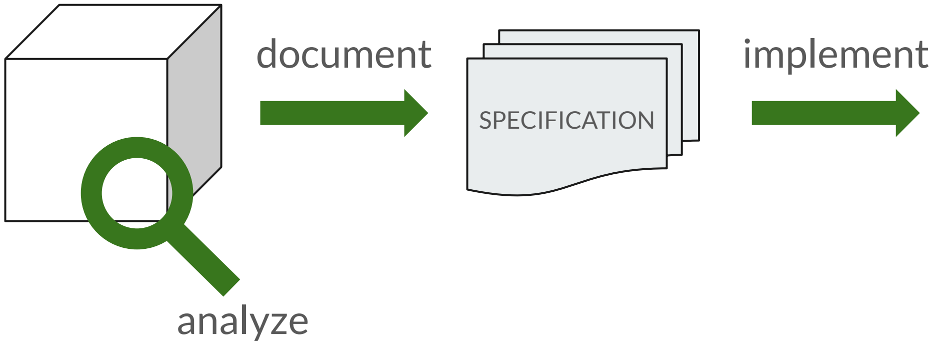
inspect

White-Box

Reverse-Engineering (Clean-Room Design)

"*__usually__, only allowed for interoperability reasons*"

TALK WITH YOUR LAWYER!



document

SPECIFICATION

implement

analyze

Step 1: Get a Legal Copy of the Program

“If you don’t own the program, it is hard to reverse-engineer it”

Simple

- Direct access (yourself, friend, company, remote)
- Freeware, Demo-Version, Educational License
- Use different tool with shared codebase

Hard Mode

- Indirect access (files are created by other person)
- Free viewer

Step 2: Collect Files for Analysis

“Diversity matters, everyone uses the tool differently!”

- If there exists an ASCII and a Binary format, collect both!
- Search by file extension

Google: `filetype:PcbDoc` **Gitlab:** `extension:PcbDoc`

- Different program, shared codebase (and file format)?

`.PcbDoc` *same as?* `.CSPcbDoc` *same as?* `.CMPcbDoc`

Altium Designer **Altium Circuit Studio** **Altium Circuit Maker**

Step 3: Existing Work and Documentation

https://github.com/thesourcerer8/altium2kicad	**The "standard" converter at that time**
https://github.com/matthiasbock/python-altium	Correctly handled Altium records
https://github.com/pcjc2/openaltium	The only C++ implementation I found
https://github.com/issus/AltiumSharp	Extensive, but published after I started
https://gitlab.cern.ch/msuminsk/altium_converter/	Runs inside Altium, creates KiCad footprints
https://github.com/vadmium/python-altium	**Contains a schematic file documentation!**
https://github.com/a3ng7n/Altium-Schematic-Parser	Altium schematic → JSON converter

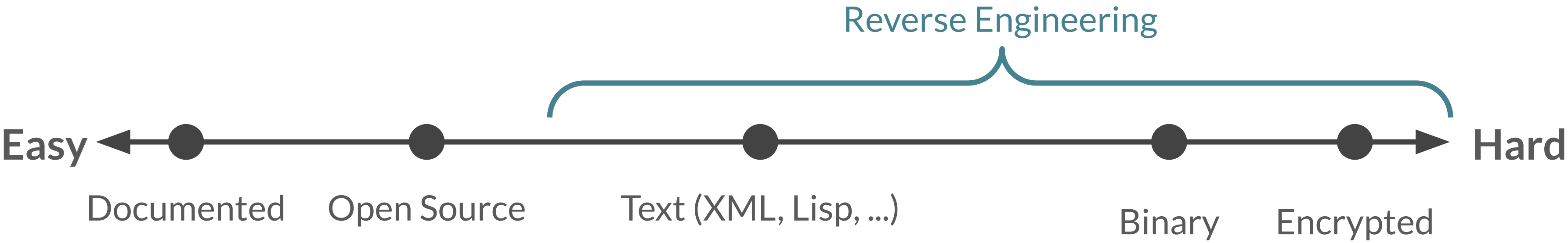
Binary File Analysis

Additional Resources

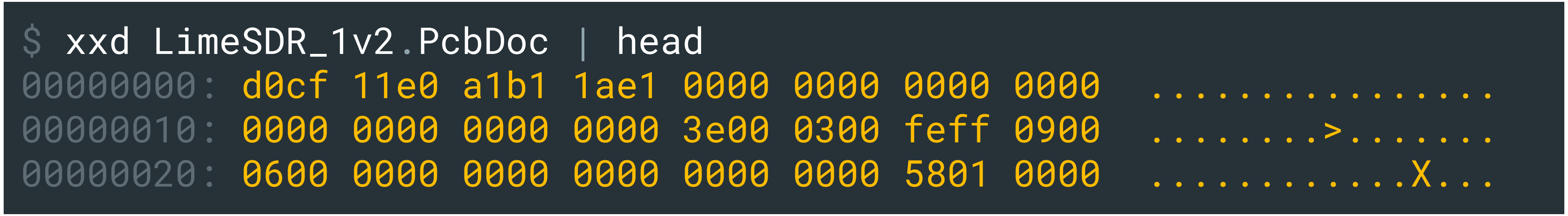
KiCad Importer Basics: Importing into KiCad from CADSTAR by Roberto Fernandez Bautista

Introduction Into File Reverse-Engineering: https://wiki.xentax.com/index.php/DGTEFF

Step 4: Text or Binary?



Null-bytes and other non-printable characters are a good hint toward binary files.



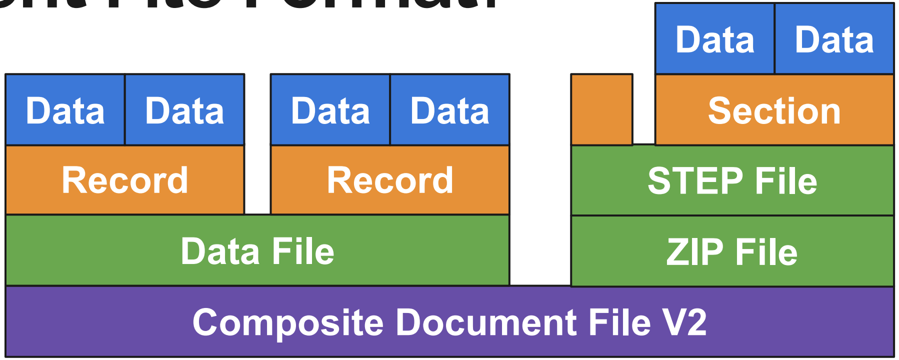
```
$ xxd LimeSDR_1v2.PcbDoc | head
00000000: d0cf 11e0 a1b1 1ae1 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 3e00 0300 feff 0900  ........>.......
00000020: 0600 0000 0000 0000 0000 0000 5801 0000  ............X...
```

The more blue you see, the easier reversing will be :D https://binvis.io

Step 5: Known Document File Format?

1. Known **magic bytes**?
2. Is it a **compound document**?
3. **Custom** file format?

What we see →



*If you have luck, the "**file**" command is sufficient. To identify embedded files, use "**binwalk**".*

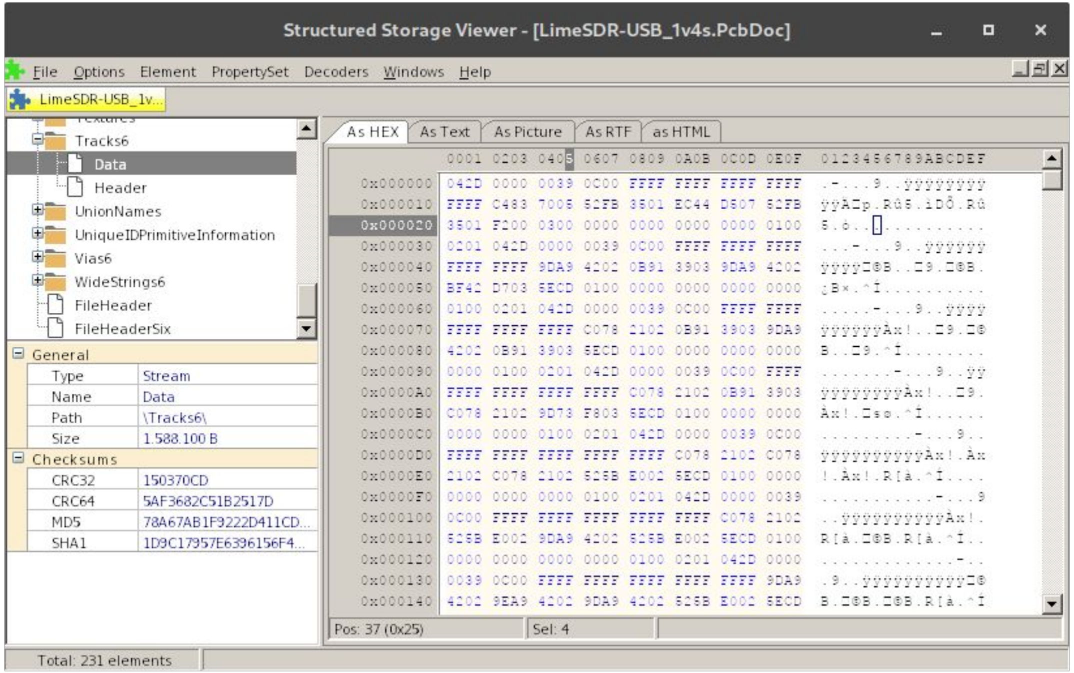
```
$ file LimeSDR_1v2.PcbDoc
LimeSDR_1v2.PcbDoc: Composite Document File V2 Document

$ binwalk -b LimeSDR_1v2.PcbDoc
```

Step 5: Known Document File Format? [Altium]

For my case (Altium PCB)

- Known file format
    - used in Windows
- Existing Viewer[1] ✓
- Existing Library[2] ✓



General	
Type	Stream
Name	Data
Path	\Tracks6\
Size	1.588.100 B
Checksums	
CRC32	150370CD
CRC64	5AF3682C51B2517D
MD5	78A67AB1F9222D411CD...
SHA1	1D9C17957E6396156F4...

```
          0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
0x000000  042D 0000 0039 0C00 FFFF FFFF FFFF FFFF  .-...9..ÿÿÿÿÿÿÿÿ
0x000010  FFFF C483 7005 52FB 3501 EC44 D507 52FB  ÿÿÄ□p.Rû5.ìDÕ.Rû
0x000020  3501 F200 0300 0000 0000 0000 0000 0100  5.ò.............
0x000030  0201 042D 0000 0039 0C00 FFFF FFFF FFFF  ...-...9..ÿÿÿÿÿÿ
0x000040  FFFF FFFF 9DA9 4202 0B91 3903 9DA9 4202  ÿÿÿÿ□©B..□9.□©B.
0x000050  BF42 D703 5ECD 0100 0000 0000 0000 0000  ¿B×.^Í..........
0x000060  0100 0201 042D 0000 0039 0C00 FFFF FFFF  .....-...9..ÿÿÿÿ
0x000070  FFFF FFFF FFFF C078 2102 0B91 3903 9DA9  ÿÿÿÿÿÿÀx!..□9.□©
0x000080  4202 0B91 3903 5ECD 0100 0000 0000 0000  B..□9.^Í........
0x000090  0000 0100 0201 042D 0000 0039 0C00 FFFF  .......-...9..ÿÿ
0x0000A0  FFFF FFFF FFFF FFFF C078 2102 0B91 3903  ÿÿÿÿÿÿÿÿÀx!..□9.
0x0000B0  C078 2102 9D73 F803 5ECD 0100 0000 0000  Àx!.□sø.^Í......
0x0000C0  0000 0000 0100 0201 042D 0000 0039 0C00  .........-...9..
0x0000D0  FFFF FFFF FFFF FFFF FFFF C078 2102 C078  ÿÿÿÿÿÿÿÿÿÿÀx!.Àx
0x0000E0  2102 C078 2102 525B E002 5ECD 0100 0000  !.Àx!.R[à.^Í....
0x0000F0  0000 0000 0000 0100 0201 042D 0000 0039  .............-...9
0x000100  0C00 FFFF FFFF FFFF FFFF FFFF C078 2102  ..ÿÿÿÿÿÿÿÿÿÿÀx!.
0x000110  525B E002 9DA9 4202 525B E002 5ECD 0100  R[à.□©B.R[à.^Í..
0x000120  0000 0000 0000 0000 0100 0201 042D 0000  .............-..
0x000130  0039 0C00 FFFF FFFF FFFF FFFF FFFF 9DA9  .9..ÿÿÿÿÿÿÿÿÿÿ□©
0x000140  4202 9EA9 4202 9DA9 4202 525B E002 5ECD  B.□©B.□©B.R[à.^Í
```

Pos: 37 (0x25) | Sel: 4

Total: 231 elements

[1] https://www.mitec.cz/ssv.html

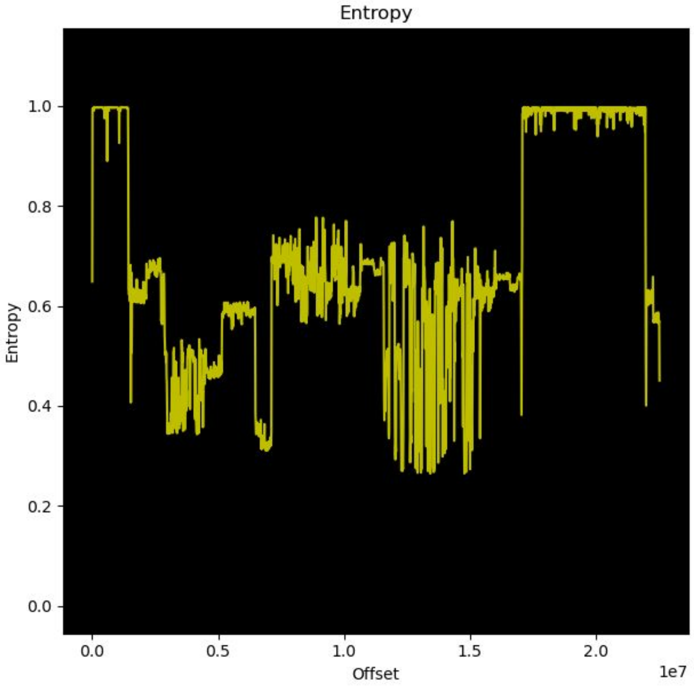
[2] https://github.com/microsoft/compoundfilereader

Step 6: Compression or Encryption Involved?

- **Entropy** is the measurement of **randomness.**
- **Encryption** results in **pseudo randomness**.

Can also be used to **detect file sections**.

```
$ binwalk -E LimeSDR_1v2.PcbDoc
```



Step 7: Tooling

HexEd.it[1]

- Web based hex editor
- Nice search utility for data types

Kaitai Struct[2]

- Describe the semantics of a file
- Useful hex view for parsed data (web based)

CyberChef[3]

- The Swiss Army Knife for data decoding

https://hex-works.com	*- simple hex viewer with diff functionality*
https://github.com/Mahlet-Inc/hobbits	*- bit based analysis with Kaitai support*
https://github.com/WerWolv/ImHex	*- hex editor for reverse engineers*
https://www.sweetscape.com/010editor/	*- propertiary hex editor*

1. https://hexed.it/
2. https://kaitai.io/
3. https://github.com/gchq/cyberchef

Step 8: Is the File-Format Canonical?

“How much does a file change on save (with and without editing)”

- A program which saves the file without moving stuff around simplifies our work

Binary diff of multiple binary files:

```
$ binwalk -WiU before_change.PcbDoc after_change.PcbDoc
```

If you want numbers (slow!):

```
$ radiff2 -sV before_change.PcbDoc after_change.PcbDoc
File size differs 127488 vs 129024
similarity: 0.952
distance: 6162
```

Step 9: Endianness

1. Insert an unique integer into the document using a numeric field (e.g. **305419896**)
   a. do NOT use a field which could be converted before save (e.g. dimension)
   b. ensure that the value is correctly saved (data type is big enough, no integer overflow)
2. Search for this value

Little Endian good old x86	**305419896** →	`78 56 34 12`	(most files are little endian)
Big Endian PowerPC, SPARC	**305419896** →	`12 34 56 78`	

Step 10: Integers

What we need to find out:

● Bit Width	Usually, 1, 2, 4 or 8 bytes long
● Signed/Unsigned	
● “Encoding”	two complement or some variable length integer?

Fixed-Length Integer
two complement

8192 → `00 20 00 00`

-8192 → `00 E0 FF FF`

Variable-Length Integer
VLQ, LEB128,...

8192 → `00 C0`

(e.g. used by Protobuf)

Step 11: Floating-Point Numbers

What we need to find out:

- Bit Width
- Encoding

Usually, 2, 4 or 8 bytes long

“Search for 90, -90, 180, -180, 270, -270, 900, ... using your hex viewer.”

IEEE 754 Sign, Exponent, Mantissa	**90.** →	`00 00 B4 42`	beware of Inf and NaN
Fake Floats no rounding errors	**90. = 900** →	`84 03 00 00`	(e.g. save angle in 0.1°)

Step 12: Internal Units

Find out the dependency between the stored value and the displayed value.

- Usually, a multiple of the metric or imperial/US unit
- integer types allow a homogeneous representation of the coordinate system

“To avoid rounding-errors, use the same unit in the program as you test for!”

Metric unit mm, µm, nm	1mil = 0.0254 mm	nm resolution allows storage of imperial units without rounding issues
Imperial/US unit inch, mil, µin	1mm = 39.37007874015748 mil	

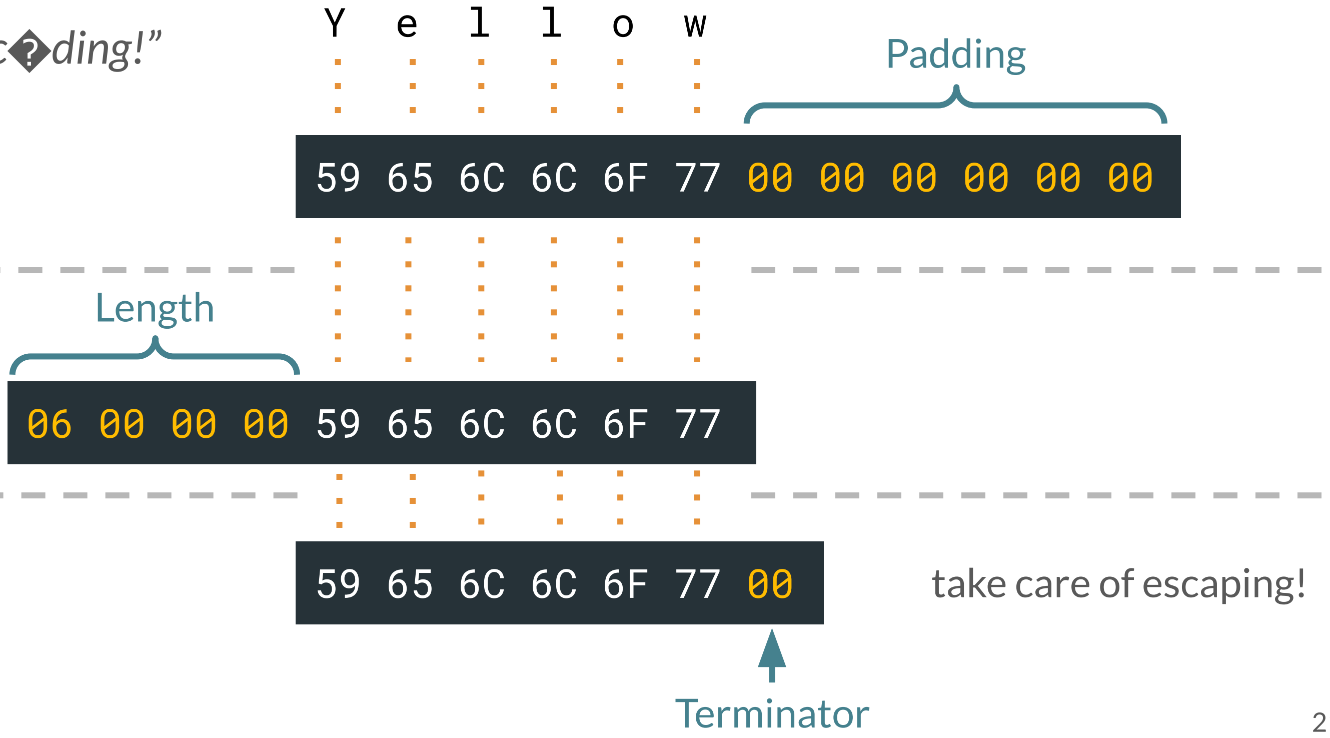
Step 13: Find Strings Inside the Binary

“Just looking at the strings allows us to see what data is presumably in the file”

```
$ strings LimeSDR_1v2.PcbDoc
PCB 6.0 Binary File
ZThis is a version 6.0 file and cannot be read correctly into this
version of tH
he software.
+Close this file immediately without saving.
-Saving this file will result in loss of data.
|RECORD=AdvancedPlacerOptions|PLACELARGECLEAR=50mil|PLACESMALLCLEAR=2
0mil|PLACEUSEROTATION=TRUE|PLACEUSELAYERSWAP=FALSE|PLACEBYPASSNET1=|P
LACEBYPASSNET2=|PLACEUSEADVANCEDPLACE=TRUE|PLACEUSD
```

Step 14: Strings

“Don’t forget about enc�ding!”



Fixed Length

simple and inflexible

```
Y  e  l  l  o  w
59 65 6C 6C 6F 77 00 00 00 00 00 00
```

Length Prefixed

```
06 00 00 00 59 65 6C 6C 6F 77
```

Terminator Based

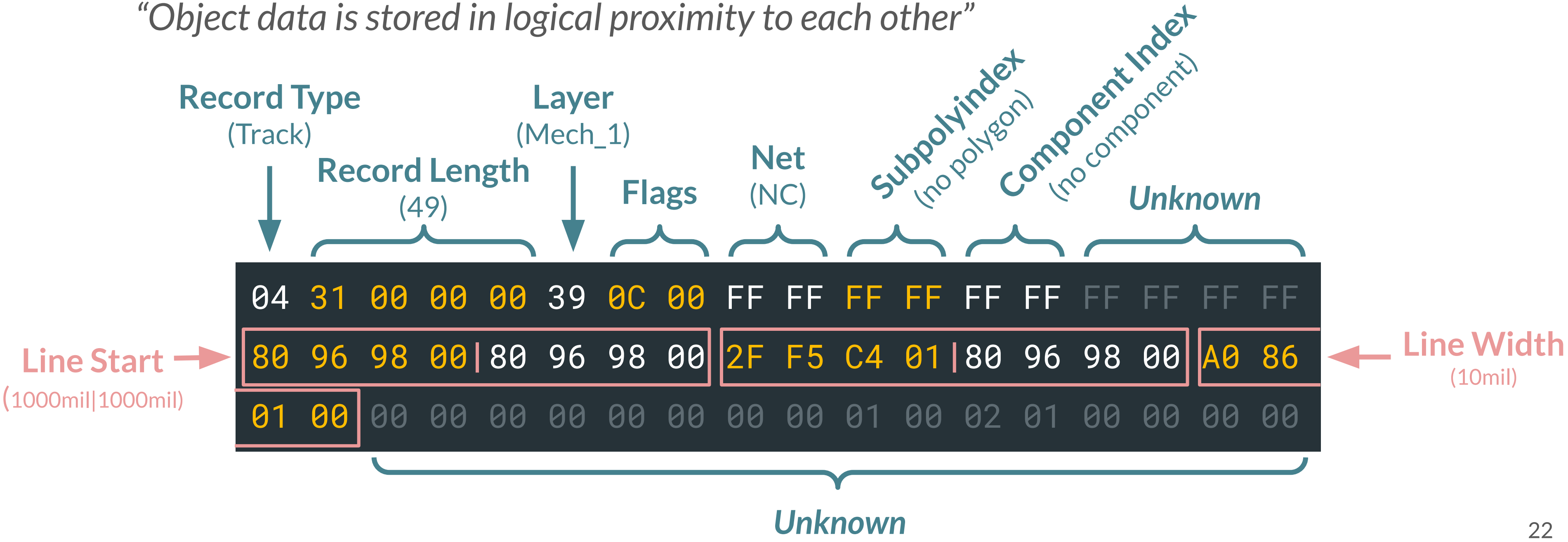
e.g. zero byte

```
59 65 6C 6C 6F 77 00
```

take care of escaping!

Step 15: Identify Records

“Object data is stored in logical proximity to each other”



```
04 31 00 00 00 39 0C 00 FF FF FF FF FF FF FF FF FF FF
80 96 98 00|80 96 98 00 2F F5 C4 01|80 96 98 00 A0 86
01 00 00 00 00 00 00 00 00 00 01 00 02 01 00 00 00 00
```

Record Type (Track)

Record Length (49)

Layer (Mech_1)

Flags

Net (NC)

Subpolyindex (no polygon)

Component Index (no component)

Unknown

Line Start (1000mil|1000mil)

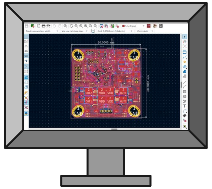
Line Width (10mil)

Unknown

Step 16: Analyzing the Record Structure

File Comparison

save modified file and run diff



save file

```
V1:  04 31 00 00 00 39 0C 00 FF FF
V2:  04 31 00 00 00 3B 0C 00 FF FF
```

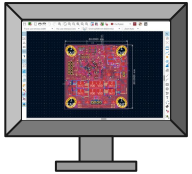
Manipulate File

modify data and view change

Mutate Data

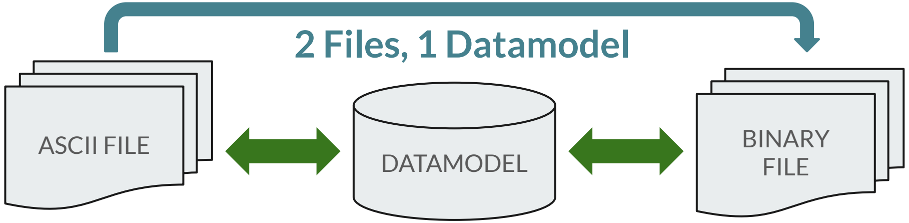
```
04 31 00 00 00 3B 0C 00 FF FF
```

load file



Documentation

ASCII <-> binary similarity



assuming a similar data-structure!

Reverse -> Code -> Test -> Repeat

“The simplest explanation is usually the correct one” [1]

Tipps

- **Start** with **visual objects**. They are easier to validate.
- Write a **parser**. Do not just **document** your findings.[2]
- Use an **intermediate data-model** for parsing.[3]
- **Check assumptions** in your code! Perhaps they are incorrect.
- Don’t be afraid of **magic constants**. Over time you will find the correct solution.
- Strive for **simplicity**. Programmers are lazy![1]
- **Testing**, Testing, Testing!

[1] Also known as Occam's razor.

[2] Use Kaitai Struct. Machine readable documentation is both!

[3] From this intermediate date-model you can then do the semantic transformation into your internal data-model.