Tools for analyzing binary formats

Petr Pučil

admin and developer of **Kaitai Struct** (a tool for working with binary formats)

Contents

1. Binary formats

Text vs. binary files
Structure of binary formats

2. Binary analysis

Parsing and serialization
Getting a parser

3. Kaitai Struct

History
Features

4. Other tools

Kaitai collaboration with other tools

1

Binary formats

1 / Text vs. binary file

Text file

- the basic unit is a **character**
- divided into **lines** - sequence of printable and whitespace characters + end of line marker (LF, CR, CR LF)
- **Readable** and easy to edit in plain text editors (Notepad, VS Code, Notepad++, Vim)
- .txt, .rtf, .csv, .json, .xml
- Characters are encoded into bytes according to the **character set** (e.g. ASCII, UTF-8)

Binary file

- the basic unit is a **byte** (1 byte = 8 bits)
- a sequence of bytes from start to end
- **Unreadable** in a plain text editor
- Image (.jpg), video (.mp4), font (.ttf), database (.sqlite), archive (.zip), executable code (ELF, Windows .exe)
- Integer values are in the **same format** as in the computer's memory

1 / Structure of binary formats

1 x 1 pixel GIF

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
00    4749 4638 3961 0100 0100 8000 00FF 8000  GIF89a....€..˙€.
10    FFFF FF2C 0000 0000 0100 0100 0002 0244  ˙˙˙,..........D
20    0100 3B                                  ..;
```

Displaying the binary representation of a GIF file in the PSpad editor

Parsing a binary file

		Offset (dec)	Byte sequence (hex)	Byte sequence meaning
Header	GIF signature + version	00	47 49 46	ASCII string ‘GIF’ – "magic" file number (GIF signature)
		03	38 39 61	ASCII string ‘89a’ – GIF version
	Logical Screen Descriptor	06	01 00	Logical screen width: one pixel
		08	01 00	Logical screen height: one pixel
		10	80	Bit field: enable global color palette
		11	00	Background color index
		12	00	Pixel width to height ratio: 1:1
	Global Color Table	13	FF 80 00	First color in the palette: orange (#ff8000)
		16	FF FF FF	Second color in the palette: white (#ffffff)

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
Image data	19	2C	Image descriptor start marker
	20	00 00	x-position of the left edge of the image: column zero
	22	00 00	y-position of the top edge of the image: line zero
	24	01 00	Image width: one pixel
	26	01 00	Image height: one pixel
	28	00	Bit field: no color palette and no interlacing
	29	02	Initial LZW code size in bits minus 1
	30	02	LZW-encoded block size in bytes
	31	44 01	Encoded image data (one pixel with zero index)
	33	00	Terminator of the LZW-encoded block
	34	3B	GIF file terminator: the ‘;’ character

1 / Structure of binary formats

1 x 1 pixel GIF

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
00    4749 4638 3961 0100 0100 8000 00FF 8000  GIF89a....€..ÿ€.
10    FFFF FF2C 0000 0000 0100 0100 0002 0244  ÿÿÿ,...........D
20    0100 3B                                  ..;
```

Displaying the binary representation of a GIF file in the PSpad editor

Parsing a binary file

Header

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
GIF signature + version	00	47 49 46	ASCII string ‘GIF’ – "magic" file number (GIF signature)
	03	38 39 61	ASCII string ‘89a’ – GIF version
Logical Screen Descriptor	06	01 00	Logical screen width: one pixel
	08	01 00	Logical screen height: one pixel
	10	80	Bit field: enable global color palette
	11	00	Background color index
	12	00	Pixel width to height ratio: 1:1
Global Color Table	13	FF 80 00	First color in the palette: orange (#ff8000)
	16	FF FF FF	Second color in the palette: white (#ffffff)

Image data

Offset (dec)	Byte sequence (hex)	Byte sequence meaning
19	2C	Image descriptor start marker
20	00 00	x-position of the left edge of the image: column zero
22	00 00	y-position of the top edge of the image: line zero
24	01 00	Image width: one pixel
26	01 00	Image height: one pixel
28	00	Bit field: no color palette and no interlacing
29	02	Initial LZW code size in bits minus 1
30	02	LZW-encoded block size in bytes
31	44 01	Encoded image data (one pixel with zero index)
33	00	Terminator of the LZW-encoded block
34	3B	GIF file terminator: the ‘;’ character

1 / Structure of binary formats

1 x 1 pixel GIF

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
00    4749 4638 3961 0100 0100 8000 00FF 8000  GIF89a....€..ÿ€.
10    FFFF FF2C 0000 0000 0100 0100 0002 0244  ÿÿÿ,...........D
20    0100 3B                                  ..;
```

Displaying the binary representation of a GIF file in the PSpad editor

Parsing a binary file

Header

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
GIF signature + version	00	47 49 46	ASCII string ‘GIF’ – "magic" file number (GIF signature)
	03	38 39 61	ASCII string ‘89a’ – GIF version
Logical Screen Descriptor	06	01 00	Logical screen width: one pixel
	08	01 00	Logical screen height: one pixel
	10	80	Bit field: enable global color palette
	11	00	Background color index
	12	00	Pixel width to height ratio: 1:1
Global Color Table	13	FF 80 00	First color in the palette: orange (#ff8000)
	16	FF FF FF	Second color in the palette: white (#ffffff)

Image data

Offset (dec)	Byte sequence (hex)	Byte sequence meaning
19	2C	Image descriptor start marker
20	00 00	x-position of the left edge of the image: column zero
22	00 00	y-position of the top edge of the image: line zero
24	01 00	Image width: one pixel
26	01 00	Image height: one pixel
28	00	Bit field: no color palette and no interlacing
29	02	Initial LZW code size in bits minus 1
30	02	LZW-encoded block size in bytes
31	44 01	Encoded image data (one pixel with zero index)
33	00	Terminator of the LZW-encoded block
34	3B	GIF file terminator: the ‘;’ character

1 / Structure of binary formats

1 x 1 pixel GIF

Parsing a binary file

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
00    4749 4638 3961 0100 0100 8000 00FF 8000  GIF89a....€..ˇ€.
10    FFFF FF2C 0000 0000 0100 0100 0002 0244  ˙˙˙,..........D
20    0100 3B                                  ..;
```

Displaying the binary representation of a GIF file in the PSpad editor

Header

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
GIF signature + version	00	47 49 46	ASCII string ‘GIF’ – "magic" file number (GIF signature)
	03	38 39 61	ASCII string ‘89a’ – GIF version
Logical Screen Descriptor	06	01 00	Logical screen width: one pixel
	08	01 00	Logical screen height: one pixel
	10	80	Bit field: enable global color palette
	11	00	Background color index
	12	00	Pixel width to height ratio: 1:1
Global Color Table	13	FF 80 00	First color in the palette: orange (#ff8000)
	16	FF FF FF	Second color in the palette: white (#ffffff)

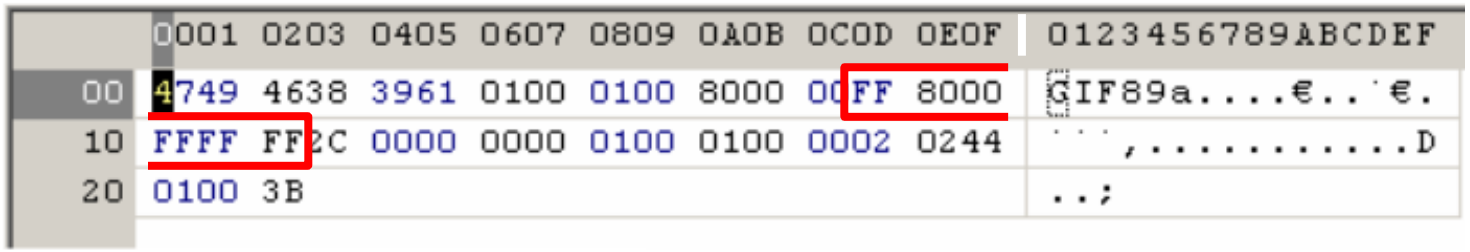
Image data

Offset (dec)	Byte sequence (hex)	Byte sequence meaning
19	2C	Image descriptor start marker
20	00 00	x-position of the left edge of the image: column zero
22	00 00	y-position of the top edge of the image: line zero
24	01 00	Image width: one pixel
26	01 00	Image height: one pixel
28	00	Bit field: no color palette and no interlacing
29	02	Initial LZW code size in bits minus 1
30	02	LZW-encoded block size in bytes
31	44 01	Encoded image data (one pixel with zero index)
33	00	Terminator of the LZW-encoded block
34	3B	GIF file terminator: the ‘;’ character

1 / Structure of binary formats

1 x 1 pixel GIF

Parsing a binary file



	0001 0203 0405 0607 0809 0A0B 0C0D 0E0F	0123456789ABCDEF
00	4749 4638 3961 0100 0100 8000 00FF 8000	GIF89a....€..ÿ€.
10	FFFF FF2C 0000 0000 0100 0100 0002 0244	ÿÿÿ,..........D
20	0100 3B	..;

Displaying the binary representation of a GIF file in the PSpad editor

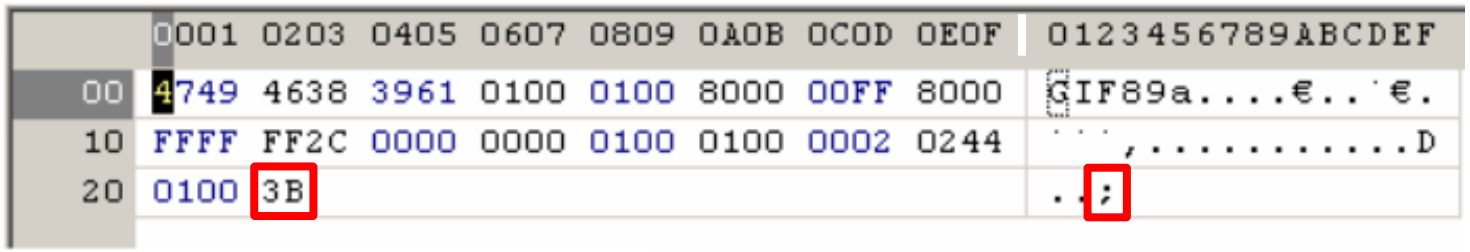
	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
Header – GIF signature + version	00	47 49 46	ASCII string ‘GIF’ – "magic" file number (GIF signature)
	03	38 39 61	ASCII string ‘89a’ – GIF version
Header – Logical Screen Descriptor	06	01 00	Logical screen width: one pixel
	08	01 00	Logical screen height: one pixel
	10	80	Bit field: enable global color palette
	11	00	Background color index
	12	00	Pixel width to height ratio: 1:1
Header – Global Color Table	13	FF 80 00	First color in the palette: orange (#ff8000)
	16	FF FF FF	Second color in the palette: white (#ffffff)

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
Image data	19	2C	Image descriptor start marker
	20	00 00	x-position of the left edge of the image: column zero
	22	00 00	y-position of the top edge of the image: line zero
	24	01 00	Image width: one pixel
	26	01 00	Image height: one pixel
	28	00	Bit field: no color palette and no interlacing
	29	02	Initial LZW code size in bits minus 1
	30	02	LZW-encoded block size in bytes
	31	44 01	Encoded image data (one pixel with zero index)
	33	00	Terminator of the LZW-encoded block
	34	3B	GIF file terminator: the ‘;’ character

1 / Structure of binary formats

1 x 1 pixel GIF

Parsing a binary file



	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
00	4749	4638	3961	0100	0100	8000	00FF	8000	GIF89a....€..ÿ€.
10	FFFF	FF2C	0000	0000	0100	0100	0002	0244	ÿÿÿ,..........D
20	0100	3B							..;

Displaying the binary representation of a GIF file in the PSpad editor

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
GIF signature + version	00	47 49 46	ASCII string ‘GIF’ – "magic" file number (GIF signature)
	03	38 39 61	ASCII string ‘89a’ – GIF version
Logical Screen Descriptor	06	01 00	Logical screen width: one pixel
	08	01 00	Logical screen height: one pixel
	10	80	Bit field: enable global color palette
	11	00	Background color index
	12	00	Pixel width to height ratio: 1:1
Global Color Table	13	FF 80 00	First color in the palette: orange (#ff8000)
	16	FF FF FF	Second color in the palette: white (#ffffff)

Header

	Offset (dec)	Byte sequence (hex)	Byte sequence meaning
Image data	19	2C	Image descriptor start marker
	20	00 00	x-position of the left edge of the image: column zero
	22	00 00	y-position of the top edge of the image: line zero
	24	01 00	Image width: one pixel
	26	01 00	Image height: one pixel
	28	00	Bit field: no color palette and no interlacing
	29	02	Initial LZW code size in bits minus 1
	30	02	LZW-encoded block size in bytes
	31	44 01	Encoded image data (one pixel with zero index)
	33	00	Terminator of the LZW-encoded block
	34	3B	GIF file terminator: the ‘;’ character

2

Binary analysis

2 / Parsing and serialization

GIF format

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	47	49	46	38	37	61	20	02	06	00	e7	00	00	18	b5	f7	GIF87a ...ç..µ÷
00000010	18	c6	ef	18	ce	ef	18	d6	ef	18	de	ef	18	e7	ef	18	.Æï.Îï.Öï.Þï.çï.
00000020	ef	ef	18	f7	e7	18	ff	21	18	ff	29	18	ff	de	18	ff	ïï.÷ç.ÿ!.ÿ).ÿÞ.ÿ
00000030	e7	21	31	f7	21	39	ff	21	42	ff	21	4a	ff	21	52	ff	ç!1÷!9ÿ!Bÿ!Jÿ!Rÿ
00000040	21	5a	f7	21	5a	ff	21	63	f7	21	6b	f7	21	73	f7	21	!Z÷!Zÿ!c÷!k÷!s÷!
00000050	7b	f7	21	84	f7	21	8c	f7	21	94	f7	21	9c	f7	21	a5	{÷!.÷!.÷!.÷!.÷!¥
00000060	f7	21	ad	f7	21	b5	f7	21	ff	18	21	ff	29	21	ff	31	÷!.÷!µ÷!ÿ.!ÿ)!ÿ1
00000070	21	ff	39	21	ff	42	21	ff	4a	21	ff	52	21	ff	5a	21	!ÿ9!ÿB!ÿJ!ÿR!ÿZ!
00000080	ff	63	21	ff	6b	21	ff	73	21	ff	7b	21	ff	84	21	ff	ÿc!ÿk!ÿs!ÿ{!ÿ.!ÿ

serialization
(writing)

```
header [Header]
logicalScreenDescriptor [LogicalScreenDescriptor]
  screenWidth = 0x220 = 544
  screenHeight = 0x6 = 6
  flags = 0xE7 = 231
  bgColorIndex = 0x0 = 0
  pixelAspectRatio = 0x0 = 0
  hasColorTable = true
  colorTableSize = 0x100 = 256
globalColorTable [GlobalColorTable]
  entries
    0 [ColorTableEntry]
    1 [ColorTableEntry]
    2 [ColorTableEntry]
    3 [ColorTableEntry]
```

parsing
(reading)

2 / Getting a parser

Dedicated format library

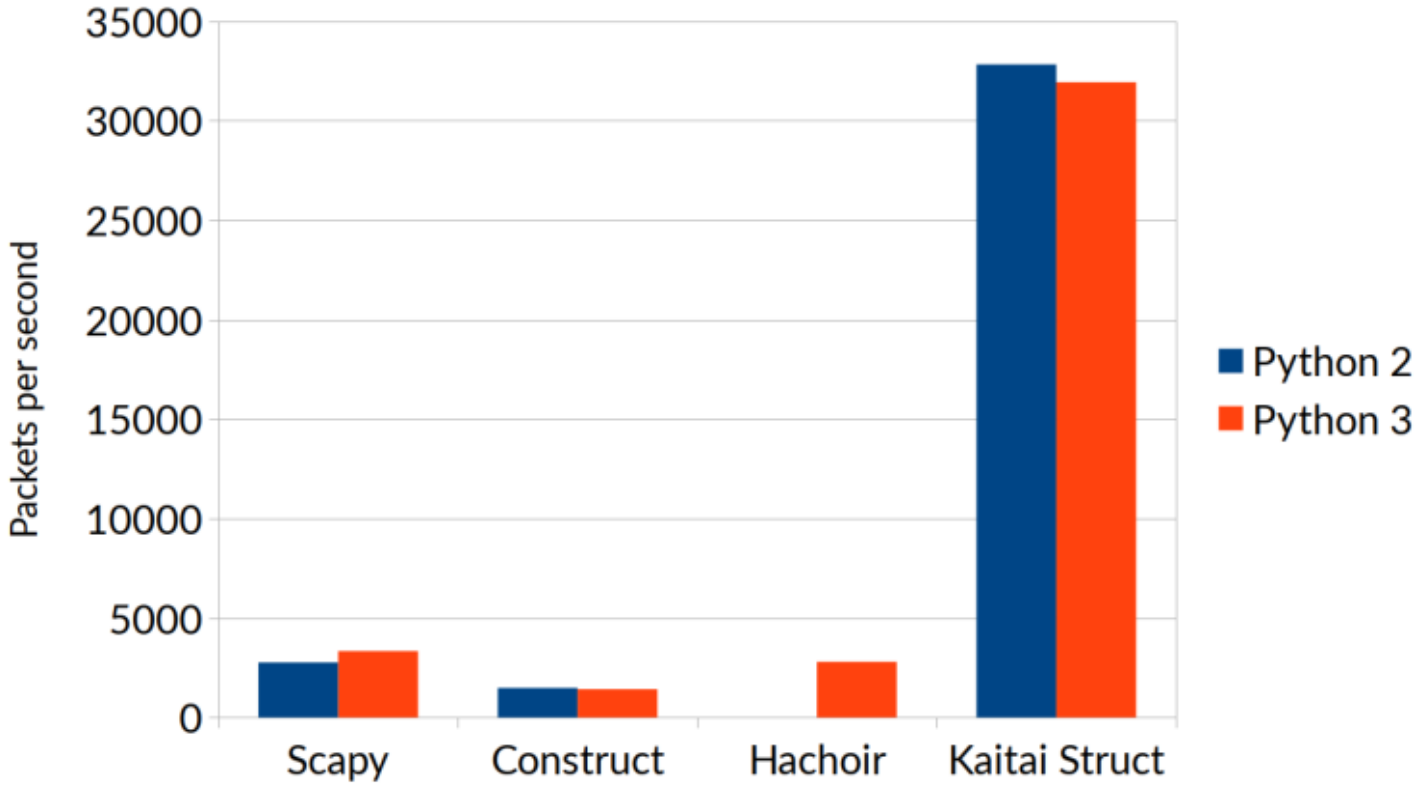
Own parser

Parser combinator → **Construct** (Python), **Binary-parser** (JavaScript), **BinData** (Ruby)

Parser generator → **Kaitai Struct**, **Spicy** (C++), **EverParse** (C), **Apache Daffodil** (Java, C)

2 / Parser combinators vs. generators

https://pythonistac.wordpress.com/2017/03/09/python-network-packet-dissection-frameworks-shootout-scapy-vs-construct-vs-hachoir-vs-kaitai-struct/



Comparison of the speed of combinators (Scapy, Construct and Hachoir) and the Kaitai Struct generator in processing network packets

kaitai.io

3

Kaitai Struct | What is it? | Quick Start | Download | News | Format Gallery | Try it — Web IDE | Documentation

Kaitai Struct

A new way to develop parsers for binary structures.

Declarative: describe the very structure of the data, not how you read or write it

Language-neutral: write once, use in all supported languages:

- C++/STL
- C#
- Go
- Java
- JavaScript
- Lua
- Nim
- Perl
- PHP
- Python
- Ruby

entry-level support

0.10 released 2022-07-08

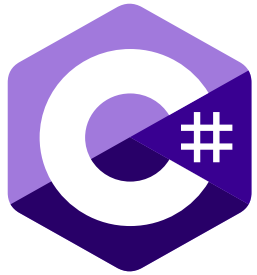
Download

Packed with tools and samples: includes a compiler, an IDE, a visualizer and massive library of popular formats

Free & open source: feel free to use, modify and join the project

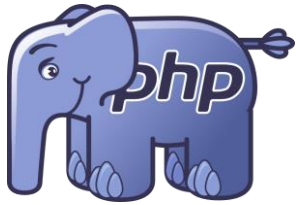
What is Kaitai Struct?

- parser generator for 11 programming languages
- other parsing targets – Construct, Awkward Arrays
- serialization in 2 languages – Java and Python
- declarative language (.ksy) for specifying arbitrary binary formats





3 / History of Kaitai Struct

2014 — origin of the project (Mikhail Yakshin) – reverse engineering

2016 — open-source release → Java, Ruby

2017 — FOSDEM 2017 → 8 supported languages ☆ 400+

My involvement: 2019 – Kaitai team
2020 – admin

2022 – 1st NLnet grant (serialization) nlnet FOUNDATION

2023 — FOSDEM 2023
serialization for Java and Python → 11 supported languages ☆ 3600+

600+

2023 – 2nd NLnet grant (Rust, …) nlnet FOUNDATION

3 / Kaitai workflow

1. Compilation

hello_world.**ksy**

```
meta:
  id: hello_world
seq:
  - id: one
    type: u1
```

kaitai-struct-compiler

hello_world.**py**

```
class HelloWorld(KaitaiStruct):
    # ...
    def _read(self):
        self.one = self._io.read_u1()
```

2. Parsing

input binary file

sample.bin

```
         0  1  2  3  4  5  6  7   01234567
00000000 ff 01                    ÿ.
```

hello_world.**py**

```
class HelloWorld(KaitaiStruct):
    # ...
    def _read(self):
        self.one = self._io.read_u1()
```

parsed data

```
└─ one = 0xFF = 255
```

kaitaistruct.py (runtime library)

```
class KaitaiStream:
    # ...
    def read_u1(self):
        return struct.unpack('B', self.read_bytes(1))[0]
```

3 / Kaitai features

- 1 format spec = 11 parsers
- standard way to describe binary formats
- library of format specifications
- GraphViz diagram
- .ksy language simplicity
- serialization
- testing
- visualization and dumping tools
  - console visualizer (ksv)
  - ksdump
  - Web IDE
- distribution

3 / Kaitai features

⟶ 1 .ksy specification = 11 parsers

.ksy spec

```
meta:
  id: hello_world
seq:
  - id: one
    type: u1
```

```
public class HelloWorld extends KaitaiStruct {
    // ...
    private void _read() {
        this.one = this._io.readU1();
    }
```

Java

```
class HelloWorld(KaitaiStruct):
    # ...
    def _read(self):
        self.one = self._io.read_u1()
```

Python

```
class HelloWorld < Kaitai::Struct::Struct
  # ...
  def _read
    @one = @_io.read_u1
  end
```

Ruby

and others (C++, C#, Go, JavaScript, Lua, Nim, Perl, PHP)

⟶ standard way to describe binary formats

GIF

18. Logical Screen Descriptor.

a. Description. The Logical Screen Descriptor contains the parameters necessary to define the area of the display device within which the images will be rendered. The coordinates in this block are given with respect to the top-left corner of the virtual screen; they do not necessarily refer to absolute coordinates on the display device. This implies that they could refer to window coordinates in a window-based environment or printer coordinates when a printer is used.

This block is REQUIRED; exactly one Logical Screen Descriptor must be present per Data Stream.

b. Required Version. Not applicable. This block is not subject to a version number. This block must appear immediately after the Header.

c. Syntax.

	7 6 5 4 3 2 1 0	Field Name	Type
0	| |	Logical Screen Width	Unsigned
1	| |		
2	| |	Logical Screen Height	Unsigned
3	| |		
4	| | | | |	<Packed Fields>	See below
5	| |	Background Color Index	Byte
6	| |	Pixel Aspect Ratio	Byte

Microsoft Word .doc

2.9.161 OcxInfo

The **OcxInfo** structure specifies an **OLE control** (such as a checkbox, radio button, and so on) in the document. The data that is contained in **OcxInfo** structures SHOULD<229> be ignored.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
dwCookie																															
ifld																															
hAccel																															
cAccel											A	B	C	D	E	F	G	H													
idoc											reserved																				

dwCookie (4 bytes): An integer value that specifies the index location of this **OcxInfo** in the **RgxOcxInfo** array. This value MUST be unique for all **OcxInfo** structures in the document.

ifld (4 bytes): An unsigned integer value that specifies an index location in the **PlcFld** structure. The value MUST be a valid **FLD** index in the correct **PlcFld** structure.

The PlcFld that is used is dependent on the value of **idoc**, as specified following.

Value	Location
1	The Main Document (FibRgFcLcb97.**fcPlcfFldMom**).
2	The Header Document (FibRgFcLcb97.**fcPlcfFldHdr**).
3	The Footnote Document (FibRgFcLcb97.**fcPlcfFldFtn**).
4	The Textbox Document (FibRgFcLcb97.**fcPlcfFldTxbx**).
6	The Endnote Document (FibRgFcLcb97.**fcPlcfFldEdn**).
7	The Comment Document (FibRgFcLcb97.**fcPlcfFldAtn**).
8	The Header Textbox Document (FibRgFcLcb97.**fcPlcfHdrtxbxTxt**).

3 / Kaitai features

⟹ library of .ksy format specifications

185 format specifications by 81 contributors

formats.kaitai.io

3D Models

gltf_binary , quake_mdl

Archive Files

android_bootldr_asus , android_bootldr_huawei , android_bootldr_qcom , android_dto , android_img , android_sparse , chrome_pak , cpio_old_le , gzip , lzh , mozilla_mar , phar_without_stub , rar , rpm , xar , zip , zisofs

Commonly Used Data Types

bcd , bytes_with_io , dos_datetime , riff , utf8_string , vlq_base128_be , vlq_base128_le

DOS-specific

dos_datetime , dos_mz , mbr_partition_table , vfat

Filesystems

android_super , apm_partition_table , apple_single_double , btrfs_stream , cramfs , ext2 , gpt_partition_table , iso9660 , luks , lvm2 , mbr_partition_table , tr_dos_image , vdi , vfat , vmware_vmdk , zisofs , zx_spectrum_tap

Fonts

grub2_font , pcf_font , ttf

Android-specific

android_bootldr_asus , android_bootldr_huawei , android_bootldr_qcom , android_img , android_nanoapp_header , android_opengl_shaders_cache , android_sparse , android_super , dex

CAD

monomakh_sapr_chg

Databases

dbf , gettext_mo , sqlite3 , tsm

Executables and Byte-code

android_nanoapp_header , dex , dos_mz , elf , java_class , mach_o , mach_o_fat , microsoft_pe , python_pyc_27 , swf , uefi_te

Firmware

andes_firmware , broadcom_trx , ines , uefi_te , uimage

Game Data Files

allegro_dat , doom_wad , dune_2_pak , fallout2_dat , fallout_dat , ftl_dat , gran_turismo_vol , heaps_pak , heroes_of_might_and_magic_agg , heroes_of_might_and_magic_bmp , minecraft_nbt , quake_mdl , quake_pak , renderware_binary_stream , saints_row_2_vpp_pc , warcraft_2_pud

3 / Kaitai features

⟹ library of .ksy format specifications

185 format specifications by 81 contributors

formats.kaitai.io

Geospatial (Maps)

shapefile_index , shapefile_main

Image Files

bmp , dicom , exif , gif , gimp_brush , icc_4 , ico , jpeg , nitf , pcx , pcx_dcx , png , psx_tim , tga , wmf , xwd

Logs

aix_utmp , glibc_utmp , hashcat_restore , mcap , sudoers_ts , systemd_journal , windows_evt_log

macOS-specific

apm_partition_table , apple_single_double , compressed_resource , dcmp_0 , dcmp_1 , dcmp_2 , dcmp_variable_length_integer , ds_store , mac_os_resource_snd , resource_fork

Networking Protocols

bitcoin_transaction , dime_message , dns_packet , ethernet_frame , hccap , hccapx , icmp_packet , ipv4_packet , ipv6_packet , microsoft_network_monitor_v2 , packet_ppi , pcap , protocol_body , rtcp_payload , rtp_packet , rtpdump , some_ip , some_ip_container , some_ip_sd , some_ip_sd_entries , some_ip_sd_options , tcp_segment , tls_client_hello , udp_datagram , websocket

Security

efivar_signature_list , openpgp_message , ssh_public_key

Windows-specific

avi , bmp , ico , microsoft_pe , regf , wav , windows_evt_log , windows_lnk_file , windows_minidump , windows_resource_file , windows_shell_items , windows_systemtime , wmf

Hardware Protocols

dtb , edid , mifare_classic

GNU/Linux-specific

btrfs_stream , cramfs , dtb , elf , ext2 , gettext_mo , glibc_utmp , luks , lvm2 , sudoers_ts , systemd_journal

CPU / Machine Code Disassembly

code_6502

Multimedia Files

android_opengl_shaders_cache , au , avi , blender_blend , creative_voice_file , fasttracker_xm_module , genmidi_op2 , id3v1_1 , id3v2_3 , id3v2_4 , magicavoxel_vox , ogg , quicktime_mov , s3m , standard_midi_file , stl , swf , vp8_ivf , wav

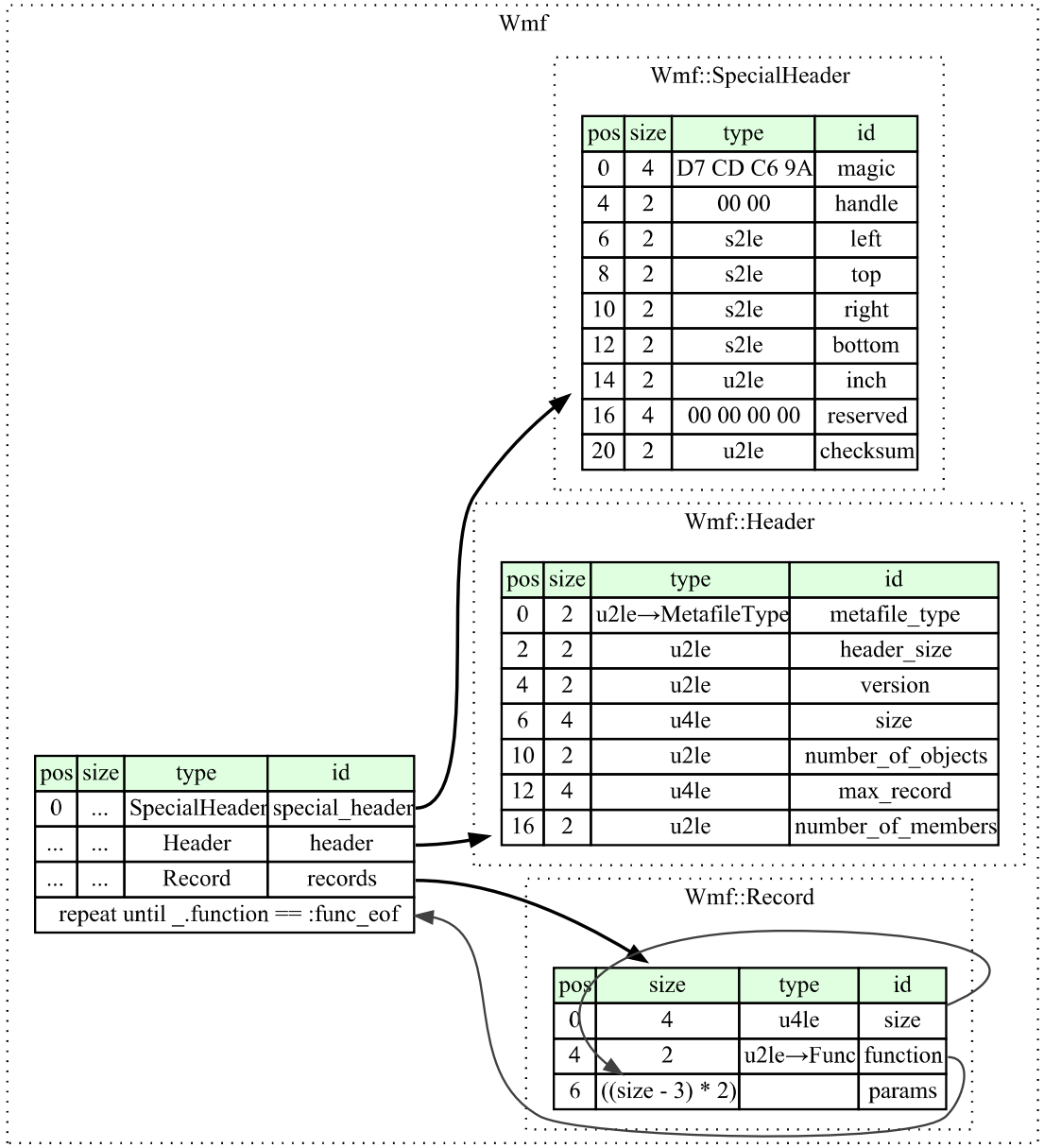
Scientific Applications

avantes_roh60 , nt_mdt , nt_mdt_pal , specpr

Serialization Protocols

asn1_der , bson , chrome_pak , dtb , google_protobuf , microsoft_cfb , minecraft_nbt , msgpack , php_serialized_value , python_pickle , ruby_marshal

- GraphViz diagram



3 / Kaitai features

⟶ .ksy language simplicity

but powerful: unaligned bit types, type switch, byte processing, imports, …

```
meta:
  id: ftl_dat
  endian: le
seq:
  - id: num_files
    type: u4
  - id: files
    type: file
    repeat: expr
    repeat-expr: num_files
```

little endian

attribute name

unsigned 4-byte integer

number of repetitions

```
types:
  file:
    seq:
      - id: ofs_data
        type: u4
    instances:
      data:
        pos: ofs_data
        type: file_data
        if: ofs_data != 0
```

byte offset

expression language

```
file_data:
  seq:
    - id: len_file
      type: u4
    - id: len_filename
      type: u4
    - id: filename
      size: len_filename
      type: str
      encoding: UTF-8
    - id: body
      size: len_file
```

char. string

no ~~type~~ ⇒ byte array

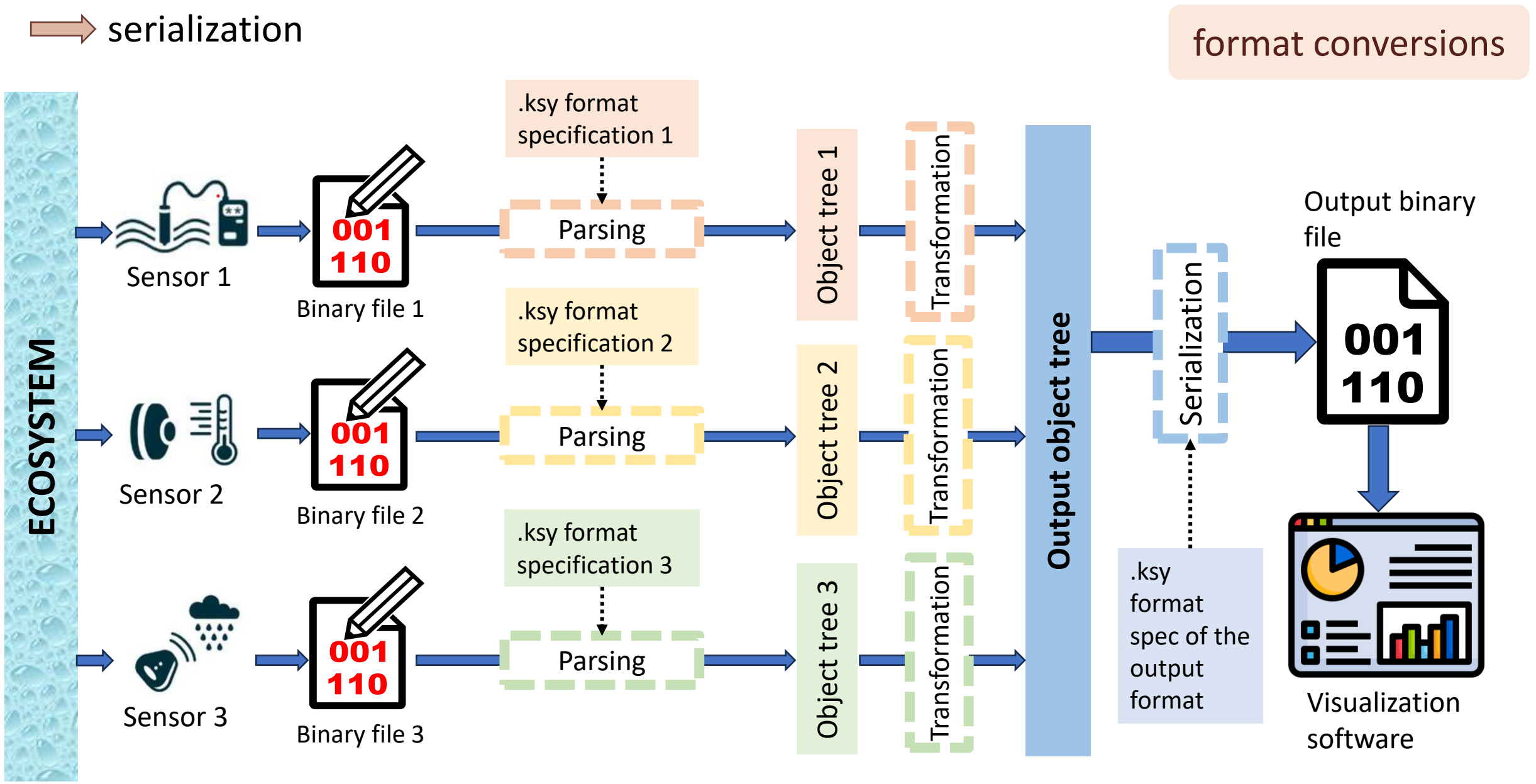
3 / Kaitai features

→ serialization

1. **editing** an existing file
2. creating a **new** file

Areas of application

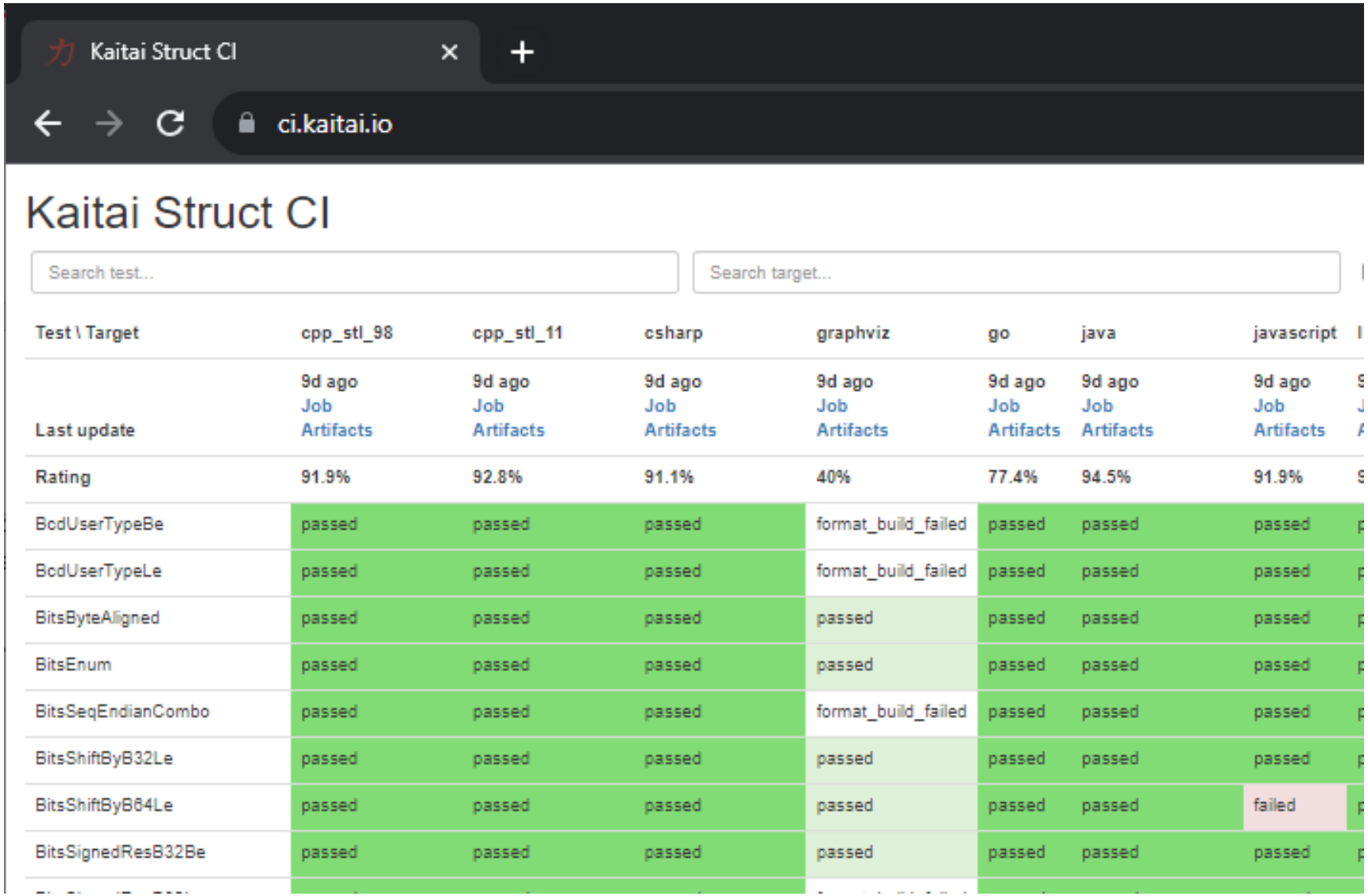
- format conversions (parse → transform → serialize to another format)
- fuzzing
- video games modding


serialization
format conversions
ECOSYSTEM
Sensor 1
Sensor 2
Sensor 3
001
110
Binary file 1
Binary file 2
Binary file 3
.ksy format specification 1
.ksy format specification 2
.ksy format specification 3
Parsing
Parsing
Parsing
Object tree 1
Object tree 2
Object tree 3
Transformation
Transformation
Transformation
Output object tree
Serialization
.ksy format spec of the output format
Output binary file
001
110
Visualization software

3 / Kaitai features

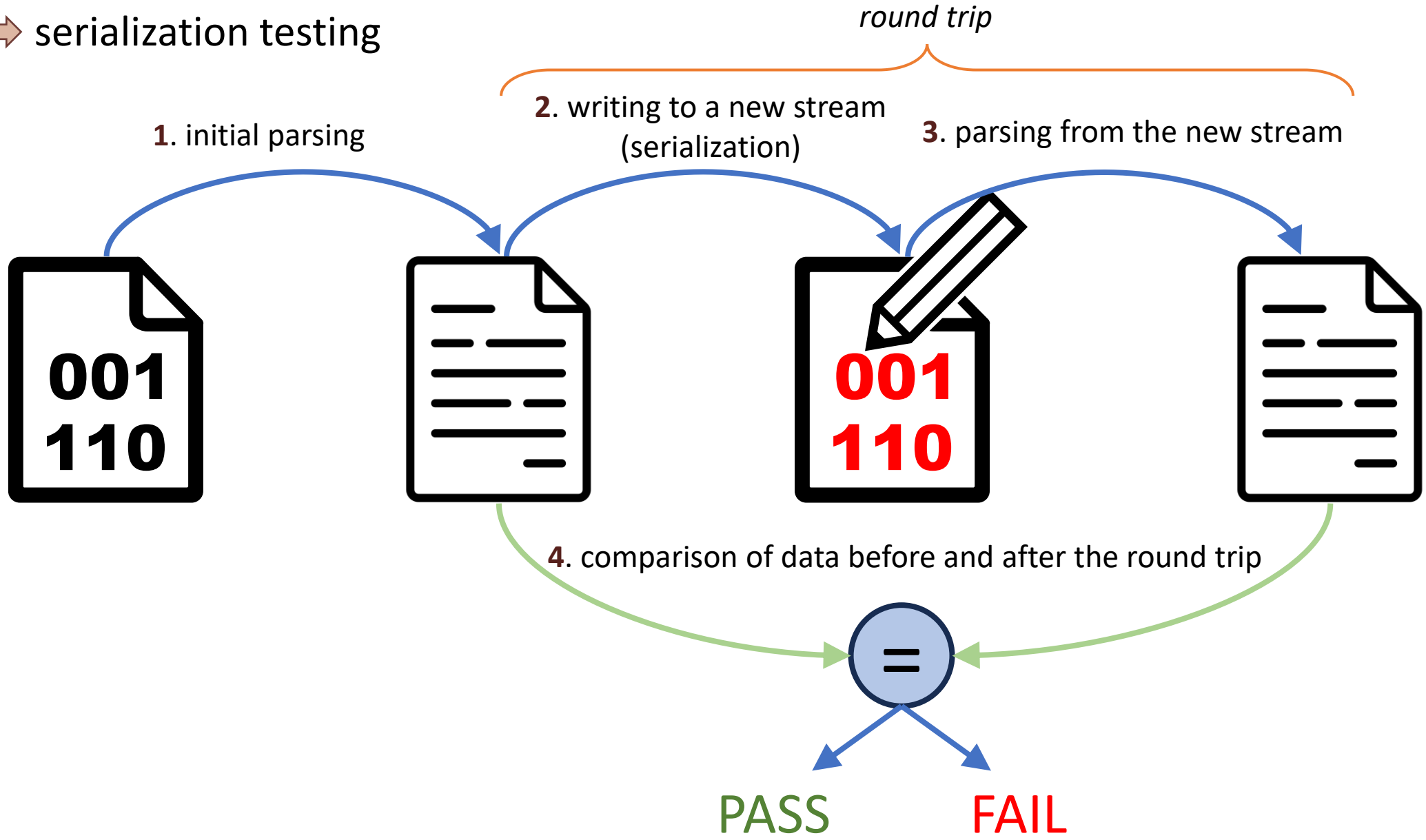
- testing

292 tests



3 / Kaitai features

⟹ serialization testing



3 / Kaitai features

- console visualizer (ksv)

github.com/kaitai-io/kaitai_struct_visualizer

```
[-] [root]                                                  00000000: 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 | .PNG........IHDR
  [.] magic = 89 50 4e 47 0d 0a 1a 0a                       00000010: 00 00 01 2c 00 00 01 2c 08 02 00 00 00 f6 1f 19 | ...,...,........
  [.] ihdr_len = 13                                         00000020: 22 00 00 03 97 49 44 41 54 78 9c ed d9 31 8a c3 | "....IDATx...1..
  [.] ihdr_type = 49 48 44 52                               00000030: 40 14 44 c1 1e e3 fb 5f 59 8a 9d 09 1c bc 40 55 | @.D...._Y.....@U
  [-] ihdr                                                  00000040: 6c b4 20 70 f2 68 98 7f b6 6b bb ce ef df b6 f3 | l. p.h...k......
    [.] width = 300                                         00000050: e8 9f f3 ad 6f 7d fb e7 b7 9f 01 a9 ef 4e fd 13 | ....o}.......N..
    [.] height = 300                                        00000060: e0 dd 44 08 31 11 42 4c 84 10 13 21 c4 bc 8e 42 | ..D.1.BL...!...B
    [.] bit_depth = 8                                       00000070: cc 12 42 4c 84 10 13 21 c4 44 08 31 11 42 4c 84 | ..BL...!.D.1.BL.
    [.] color_type = color_type_truecolor                   00000080: 10 73 a2 80 98 25 84 98 08 21 26 42 88 89 10 62 | .s...%...!&B...b
    [.] compression_method = 0                              00000090: 22 84 98 08 21 e6 44 01 31 4b 08 31 11 42 4c 84 | "...!.D.1K.1.BL.
    [.] filter_method = 0                                   000000a0: 10 13 21 c4 44 08 31 af a3 10 b3 84 10 13 21 c4 | ..!.D.1.......!.
    [.] interlace_method = 0                                000000b0: 44 08 31 11 42 4c 84 10 13 21 c4 9c 28 20 66 09 | D.1.BL...!..( f.
  [.] ihdr_crc = f6 1f 19 22                                000000c0: 21 26 42 88 89 10 62 22 84 98 08 21 26 42 88 39 | !&B...b"...!&B.9
  [-] chunks (2 = 0x2 entries)                              000000d0: 51 40 cc 12 42 4c 84 10 13 21 c4 44 08 31 11 42 | Q@..BL...!.D.1.B
    [-] 0                                                   000000e0: cc eb 28 c4 2c 21 c4 44 08 31 11 42 4c 84 10 13 | ..(.,!.D.1.BL...
      [.] len = 919                                         000000f0: 21 c4 44 08 31 27 0a 88 59 42 88 89 10 62 22 84 | !.D.1'..YB...b".
      [.] type = "IDAT"                                     00000100: 98 08 21 26 42 88 89 10 62 4e 14 10 b3 84 10 13 | ..!&B...bN......
      [.] body = 78 9c ed d9 31 8a c3 40 14 44 c1 1e e3 fb 5f…00000110: 21 c4 44 08 31 11 42 4c 84 10 f3 3a 0a 31 4b 08 | !.D.1.BL...:.1K.
      [.] crc = 21 cb 54 d1                                 00000120: 31 11 42 4c 84 10 13 21 c4 44 08 31 11 42 cc 89 | 1.BL...!.D.1.B..
    [-] 1                                                   00000130: 02 62 96 10 62 22 84 98 08 21 26 42 88 89 10 62 | .b..b"...!&B...b
      [.] len = 0                                           00000140: 22 84 98 13 05 c4 2c 21 c4 44 08 31 11 42 4c 84 | ".....,!.D.1.BL.
      [.] type = "IEND"                                     00000150: 10 13 21 c4 bc 8e 42 cc 12 42 4c 84 10 13 21 c4 | ..!...B..BL...!.
      [.] body =                                            00000160: 44 08 31 11 42 4c 84 10 73 a2 80 98 25 84 98 08 | D.1.BL..s...%...
      [.] crc = ae 42 60 82                                 00000170: 21 26 42 88 89 10 62 22 84 98 08 21 e6 44 01 31 | !&B...b"...!.D.1
                                                            00000180: 4b 08 31 11 42 4c 84 10 13 21 c4 44 08 31 af a3 | K.1.BL...!.D.1..
                                                            00000190: 10 b3 84 10 13 21 c4 44 08 31 11 42 4c 84 10 13 | .....!.D.1.BL...
                                                            000001a0: 21 c4 9c 28 20 66 09 21 26 42 88 89 10 62 22 84 | !..( f.!&B...b".
                                                            000001b0: 98 08 21 26 42 88 39 51 40 cc 12 42 4c 84 10 13 | ..!&B.9Q@..BL...
```

3 / Kaitai features

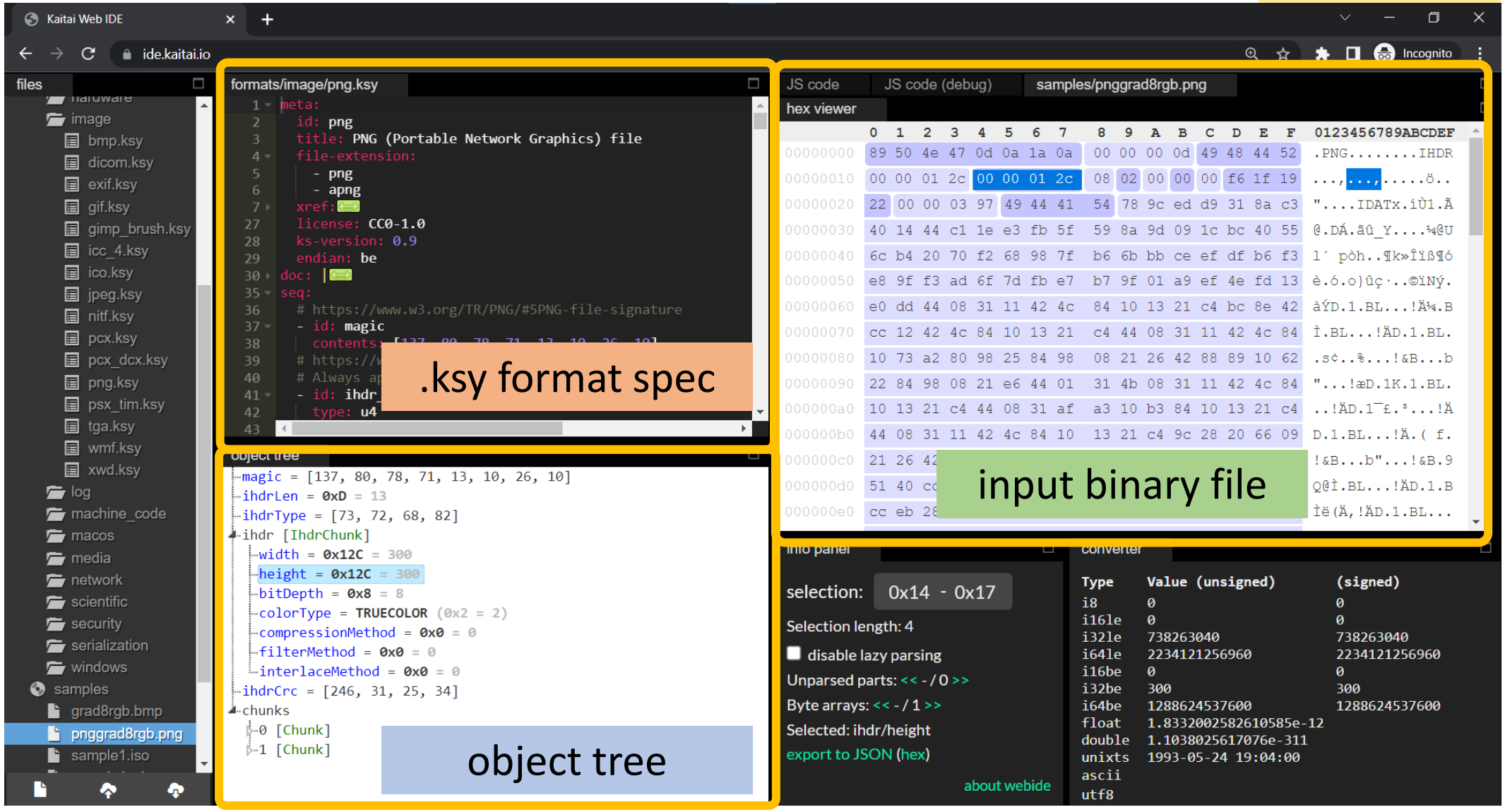
→ ksdump (JSON output)

github.com/kaitai-io/kaitai_struct_visualizer


```
{
  "magic": "89 50 4E 47 0D 0A 1A 0A",
  "ihdr_len": 13,
  "ihdr_type": "49 48 44 52",
  "ihdr": {
    "width": 300,
    "height": 300,
    "bit_depth": 8,
    "color_type": "color_type_truecolor",
    "compression_method": 0,
    "filter_method": 0,
    "interlace_method": 0
  },
  "ihdr_crc": "F6 1F 19 22",
  "chunks": [
    {
      "len": 919,
      "type": "IDAT",
      "body": "78 9C ED D9 31 8A C3 40 14 44 C1 1E E3 FB 5F…",
      "crc": "21 CB 54 D1"
    },
    {
      "len": 0,
      "type": "IEND",
      "body": "",
      "crc": "AE 42 60 82"
    }
  ]
}
```

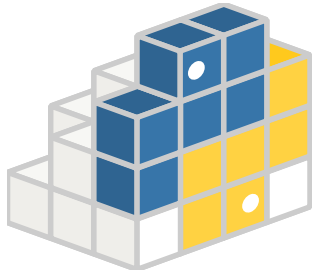
Web IDE

ide.kaitai.io



- distribution – runtime libraries

npm – JavaScript

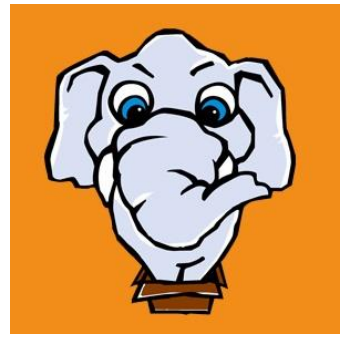


PyPI – Python

NuGet – C#

Maven Central Repository – Java

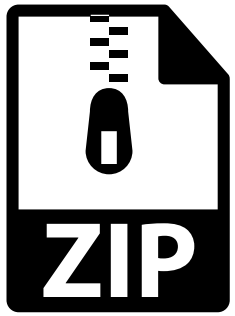
RubyGems – Ruby

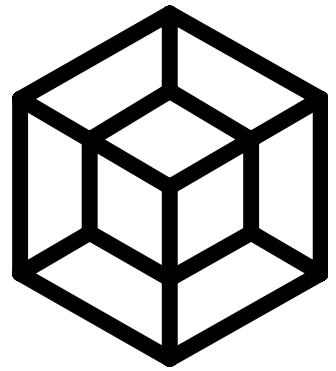


Packagist – PHP

⟹ distribution – compiler

https://repology.org/project/kaitai-struct/versions





WinGet

Homebrew

Void Linux

AUR

nixpkgs

4

Other binary analysis tools

4 / Binary analysis tools

Dirk Loss

list of parsing tools, hex editors...

https://github.com/dloss/binary-parsing

dloss/**binary-parsing**

A list of generic tools for parsing binary data structures, such as file formats, network protocols or bitstreams

11 Contributors · 2 Issues · 781 Stars · 62 Forks

Awesome binary parsing

A list of generic tools for parsing binary data structures, such as file formats, network protocols or bitstreams.

Parser generators, parsing libraries and frameworks

- Kaitai Struct (DSL): declarative language used for describe various binary data structures, laid out in files or in memory
- Nom (Rust): Rust parser combinator framework
- Hammer (C): bit-oriented parsing library
- Construct (Python): library for parsing and building of data structures (binary or textual). Define your data structures in a declarative manner
- Spicy (DSL, C/C++, Zeek): a next-generation parser generator for network protocols and file formats
- Hachoir (Python): view and edit a binary stream field by field. Long list of parsers for all

decoding and encoding.

- EverParse: a framework for generating verified secure parsers and formatters from domain-specific format specification languages

Stand-alone software

Hex editors with grammars

- Synalyze It!
- Hexinator
- 010 Editor
- Kiewtai: plugin for the Hiew hex editor that makes the Kaitai parsers available
- Hobbits: multi-platform GUI for bit-based analysis, processing, and visualization. Has a Kaitai plugin.
- ImHex: A Hex Editor for Reverse Engineers, Programmers and people who value their retinas when working at 3 AM.

Wireshark

- Hobbits: multi-platform GUI for bit-based analysis, processing, and visualization. Has a Kaitai plugin.
- ImHex: A Hex Editor for Reverse Engineers, Programmers and people who value their retinas when working at 3 AM.

Wireshark

Wireshark is a network protocol analyzer that includes dissectors for over two thousand protocols.

- TShark: command line version, can easily be called from shell scripts.
- Wireshark Generic Dissector: add-on, allows dissection of a protocol based on a text description of the protocol elements
- Wireshark Lua: dissectors can be written in Lua (Examples)
- pyreshark: plugin providing a simple interface for writing Wireshark dissectors in Python
- Sharktools (Python, Matlab): Tools for programmatic parsing of packet captures using Wireshark functionality

Other Stand-alone Software

- Wireshark Lua: dissectors can be written in Lua (Examples)
- pyreshark: plugin providing a simple interface for writing Wireshark dissectors in Python
- Sharktools (Python, Matlab): Tools for programmatic parsing of packet captures using Wireshark functionality

Other Stand-alone Software

- GNU poke: The extensible editor for structured binary data
- Netzob: open source tool for reverse engineering, traffic generation and fuzzing of communication protocols
- Cat Karat Packet Builder: packet generation tool that allows to build custom packets for firewall or target testing
- radare2 (C, with bindings/pipe for almost all languages): Unix-like reverse engineering framework and commandline tools. See Parsing a fileformat with radare2 and Types.
- Veles: open source tool for binary analysis

Research papers

- LangSec Platform: Towards a Platform to Compare Binary Parser Generators. Olivier Levillain, Sébastien Naud, Aina Toky Rasoamanana (Video)

Parser generators, parsing libraries and frameworks

- Kaitai Struct (DSL): declarative language used for describe various binary data structures, laid out in files or in memory
- Nom (Rust): Rust parser combinator framework
- Hammer (C): bit-oriented parsing library
- Construct (Python): library for parsing and building of data structures (binary or textual). Define your data structures in a declarative manner
- Spicy (DSL, C/C++, Zeek): a next-generation parser generator for network protocols and file formats
- Hachoir (Python): view and edit a binary stream field by field. Long list of parsers for all kinds of formats

- RecordFlux: toolset for the formal specification of messages and the generation of verifiable binary parsers and message generators (Ada-inspired).
- DataScript Tools (DSL): DataScript is a formal language for modelling binary datatypes,

decoding and encoding.

- EverParse: a framework for generating verified secure parsers and formatters from domain-specific format specification languages

Stand-alone software

Hex editors with grammars

- Synalyze It!
- Hexinator
- 010 Editor
- Kiewtai: plugin for the Hiew hex editor that makes the Kaitai parsers available
- Hobbits: multi-platform GUI for bit-based analysis, processing, and visualization. Has a Kaitai plugin.
- ImHex: A Hex Editor for Reverse Engineers, Programmers and people who value their retinas when working at 3 AM.

Wireshark

fq

Mattias Wadman
inspired by jq, a tool for working with JSON data
dealing with multimedia formats

github.com/wader/fq

wader/**fq**

jq for binary formats - tool, language and decoders for working with binary and text formats

26 Contributors | 19 Used by | 9k Stars | 214 Forks

4 / Kaitai collaboration with other tools

BANG

Armijn Hemel

firmware analysis

- provenance detection
- security scans

github.com/armijnhemel/binaryanalysis-ng

armijnhemel/

binaryanalysis-ng

Binary Analysis Next Generation (BANG)

6 Contributors

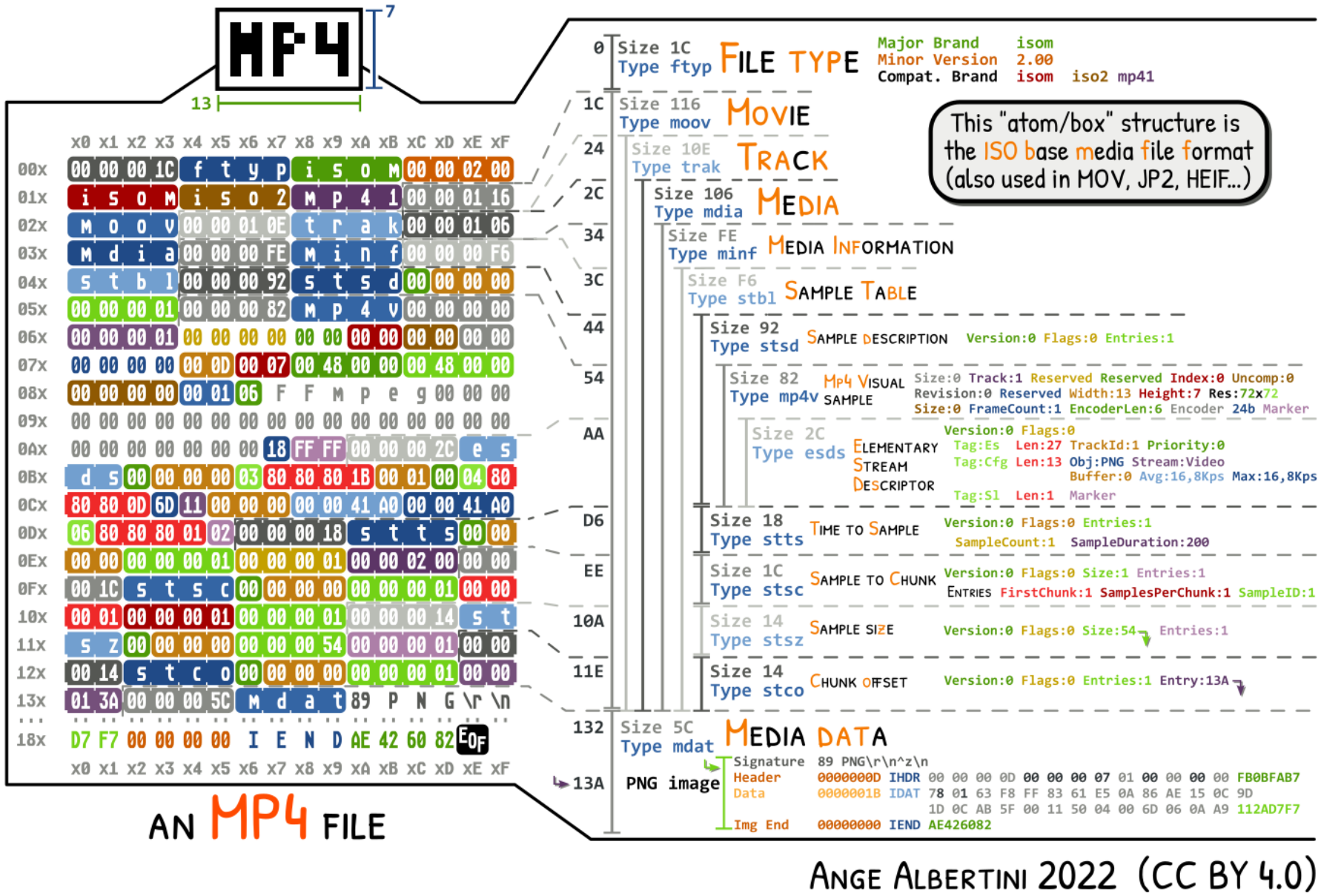
18 Issues

428 Stars

66 Forks

Corkami posters

Ange Albertini



Don't write parsers by hand, use a suitable binary tool.

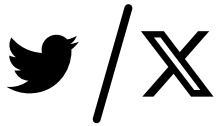
There are many to choose from
and they will save you a lot of time.

Thanks!

 https://kaitai.io/

 https://github.com/kaitai-io

 https://gitter.im/kaitai_struct/Lobby

 @kaitai_io