

Dissecting media file formats with Kaitai Struct

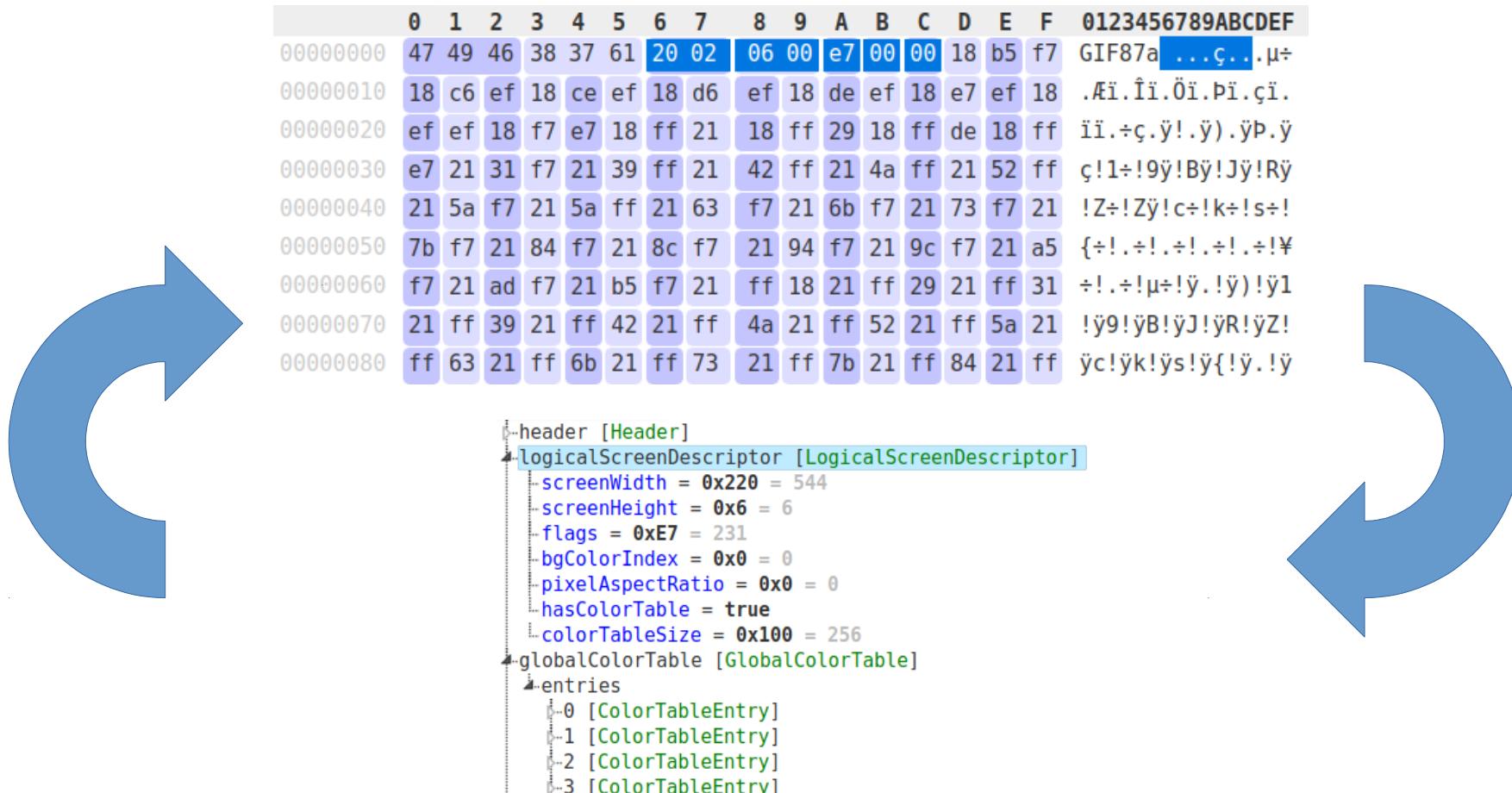
FOSDEM 2017
Mikhail Yakshin (GreyCat)
Kaitai Project

<http://kaitai.io/>
Twitter: @kaitai_io

File formats: a problem?

- Media software developers have to deal with multitude of different media file formats
- Some of them are proprietary and undocumented → need to be reverse engineered
- Some of them are documented, but still parsing binary files is pain

The mission: from stream to memory (and back)



Typical development workflow

- Write some parsing code in a certain programming language
- Write some extra debugging code (dump to screen, check assertions, etc)
- Debug it till you drop
 - with dumping
 - with debugger
 - with asserts, etc
- Want to support some other programming language? Redo from start.

Almost every media format library has these “dumping” tools

- libpng (PNG)
 - pnginfo, pngcp, pngchunkdesc, pngchunks
- openjpeg2 (JPEG 2000)
 - opj_decompress, opj_compress, opj_dump
- libogg (Ogg)
 - ogginfo
- swftools (Adobe Flash)
 - swfdump, swfextract, swfrender, ...

Errors in file format libraries are devastatingly dangerous

- Almost always remotely exploitable, frequently provide arbitrary code execution, information leaking, DoS
- libpng: since 2010:
 - 22 vulnerabilities
 - 15 DoS
 - 13 overflow / code execution
- libjpeg
 - 4 vulnerabilities
 - 3 infoleaks
 - 1 code execution

File formats description: no single standard

ELF Header

Some object file control structures can grow, because the ELF header contains their actual sizes. If the object file format changes, a program may encounter control structures that are larger or smaller than expected. Programs might therefore ignore “extra” information. The treatment of “missing” information depends on context and will be specified when and if extensions are defined.

Figure 1-3: ELF Header

```
#define EI_NIDENT      16

typedef struct {
    unsigned char   e_ident[EI_NIDENT];
    Elf32_Half     e_type;
    Elf32_Half     e_machine;
    Elf32_Word     e_version;
    Elf32_Addr    e_entry;
    Elf32_Off     e_phoff;
    Elf32_Off     e_shoff;
    Elf32_Word     e_flags;
    Elf32_Half     e_ehsize;
    Elf32_Half     e_phentsize;
    Elf32_Half     e_phnum;
    Elf32_Half     e_shentsize;
    Elf32_Half     e_shnum;
    Elf32_Half     e_shstrndx;
} Elf32_Ehdr;
```



e_ident

The initial bytes mark the file as an object file and provide machine-independent data with which to decode and interpret the file’s contents. Complete descriptions appear below, in “ELF Identification.”

e_type

This member identifies the object file type.

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_DYN	2	Executable file

File formats description: no single standard

: 768

J. Postel

ISI

28 August 1986

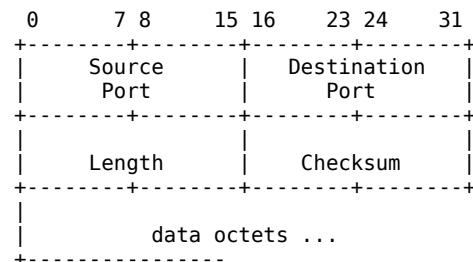
User Datagram Protocol

Introduction

The User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection is not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



User Datagram Header Format

Fields

The Source Port is an optional field, when meaningful, it indicates the port the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If unused, a value of zero is inserted.

File formats description: no single standard

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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 [PlcFId](#) structure. The value MUST be a valid [FLD](#) index in the correct [PlcFId](#) structure.

The PlcFId 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) .

Debugging networking protocols: they've got Wireshark

The screenshot shows the Wireshark interface with the following details:

- File menu:** File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Print, and zoom.
- Search bar:** "Apply a display filter ... <Ctrl-/>" and "Expression..." field.
- Table:** A list of network packets. The columns are No., Time, Source, Destination, Protocol, Length, and Info. The table shows 9 packets of type SCTP, with lengths ranging from 78 to 1102 bytes and various descriptions like INIT, INIT_ACK, and DATA.
- Packet details pane:** Shows the selected packet (Frame 10) in expanded view. It includes fields for Version (4), Header Length (20 bytes), Differentiated Services Field (0x10), Total Length (1088), Identification (0x0000), Flags (0x02), Fragment offset (0), and Time to live (64).
- Hex editor pane:** Displays the raw hex and ASCII data for the selected packet.
- Bottom status bar:** Shows "Total Length (ip.len), 2 bytes", "Packets: 74 · Displayed: 74 (100.0%) · Load time: 0:0.2 · Profile: Default".

Enter Kaitai Struct

- Declarative file format specification language (.ksy)
- Compiles into ready-made parsers in many supported target programming languages
- Visualization, dumping and debugging tools
- .ksy is YAML-based → easy to write your own tools
- Free & libre:
 - GPLv3 for compiler
 - MIT/Apache2 for runtime

Supported target languages

- C++ (STL)
- C#
- Java
- JavaScript
- Perl
- PHP
- Python
- Ruby

Bonus: GraphViz support

Natural API generated by KS

```
↳ 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]
```

```
// Java

Gif myFile = Gif.fromFile("/path/to/file.gif");
System.out.println(
    "This file is " +
    myFile.logicalScreenDescriptor.screenWidth +
    " x " +
    myFile.logicalScreenDescriptor.screenHeight
);
```

A picture worth a thousand words: Web IDE

The screenshot displays the WebIDE interface, a web-based tool for reverse engineering binary files using Kaitai Struct. The interface is divided into several panels:

- files**: A sidebar listing various file formats and samples.
- .ksy editor**: The main editor pane showing Kaitai Struct code for an AVI file. The code defines a structure with fields like meta, id, endian, title, ks-version, seq, magic1, contents, file_size, type, magic2, contents, data, and blocks. It also includes size, types, and block definitions.
- parsed as tree**: A tree view of the parsed structure, showing nodes for magic1, fileSize, magic2, data [Blocks], entries, and a detailed view of the first entry.
- JS code**, **JS code (debug)**, **input binary**: Tabs for viewing the generated JavaScript code and the input binary data.
- hex viewer**: A hex dump of the binary data, showing bytes from 0 to F. The first few bytes are RIFF (41 49 46 4D), followed by file header information.
- info panel**: Displays the selection (0x14 - 0xd3), parsing options (disable lazy parsing), unparsed parts (<> - /0 >>), byte arrays (<> - /112 >>), and selected data (data/entries/0/data).
- converter**: A table for converting values between different types. It shows pairs like 18 (unsigned) and 104 (signed), and other conversions for larger types like i64le, float, and double.

Console visualizer: JPEG

```
[-] [root]
[-] @segments (11 = 0xb entries)
[+] 0
[-] 1
[.] @magic = ff
[.] @marker = marker_app0
[.] @length = 16
[-] @data
[.] @magic = "JFIF\0000"
[.] @version_major = 1
[.] @version_minor = 1
[.] @density_units = density_unit_pixels_per_inch
[.] @density_x = 72
[.] @density_y = 72
[.] @thumbnail_x = 0
[.] @thumbnail_y = 0
[.] @thumbnail =
[?] image_data
[+] 2
[+] 3
[+] 4
[-] 5
[.] @magic = ff
[.] @marker = marker_sof0
[.] @length = 17
[-] @data
[.] @bits_per_sample = 8
[.] @image_height = 3264
[.] @image_width = 2448
[.] @num_components = 3
[+] @components (3 = 0x3 entries)
[?] image_data
[+] 6
[+] 7
[+] 8
[+] 9

00001c20: ff 00 3f f3 12 c8 f0 cb 68 9f 7d 4d e3 3f 82 e8 | ..?....h.}M.?...
00001c30: 77 37 8b 2f 49 3d 71 71 39 ff 00 d9 6b 8b f1 3f | w7./I=qq9...k..?
00001c40: c5 cf 80 3e 16 d2 8e ad ad f8 af 50 8a 05 70 85 | ...>.....P..p.
00001c50: bc db 93 c9 c9 1f 75 33 ce 0d 7c 3e ff 00 a0 c9 5b | .....u3..|>..z.
00001c60: 3f f6 84 ff 00 92 6b 27 fd 7d c3 ff 00 a0 c9 5b | ?....k'}.}....[.
00001c70: e1 78 8b 17 39 46 2e 5b fa 8a 79 3e 19 45 be 53 | .x..9F.[..y>.E.S
00001c80: f4 42 db f6 8f fd 99 f5 24 2f a7 f8 af 55 94 03 | B.....$/..U..
00001c90: 8f 95 ae bd 3d d0 75 aa 52 fc 74 fd 9f ae 4f cb | ....=u.R.t...0.
00001ca0: af 6b 2d ff 00 03 9c 57 e3 8f c3 cf f8 f4 6f f7 | .k....W.....o.
00001cb0: 97 ff 00 41 15 ea b6 bf 7d 7e 87 f9 8a db 17 9e | ...A.....}~.....
00001cc0: e2 a1 2e 55 33 38 e5 b4 35 f7 51 fa 5d 2f c6 5f | ...U38..5.Q.]/_.
00001cd0: 80 f0 80 ff 00 da 9a cb b1 ff 00 6a 53 c7 e2 6b | .....jS..k
00001ce0: 3e 4f da 5f e0 8d 94 67 4d fb 66 b6 d1 90 df 28 | >0._...gM.f....(
00001cf0: 2c 01 07 86 c1 2e 2b e0 89 fa 47 fe e9 fe 95 c4 | ,....+...G....
00001d00: ea 9f f1 f9 17 fd 73 97 f9 ad 78 78 8c ee bd 6d | .....s...xx...m
00001d10: 2a bb db 5d 75 d4 f4 28 e5 b4 12 ba 89 ff d9 ff | *..]u..(.....
00001d20: db 00 43 00 03 02 02 03 02 02 03 03 03 03 04 03 | ..C.....
00001d30: 03 04 05 08 05 05 04 04 05 0a 07 07 06 08 0c 0a | .....
00001d40: 0c 0c 0b 0a 0b 0b 0d 0e 12 10 0d 0e 11 0e 0b 0b | .....
00001d50: 10 16 10 11 13 14 15 15 15 0c 0f 17 18 16 14 18 | .....
00001d60: 12 14 15 14 ff db 00 43 01 03 04 04 05 04 05 09 | .....C.....
00001d70: 05 05 09 14 0d 0b 0d 14 14 14 14 14 14 14 14 14 | .....
00001d80: 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 | .....
00001d90: 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 | .....
00001da0: 14 14 14 14 14 14 14 14 14 14 ff c0 00 11 08 0c c0 | .....
00001db0: 09 90 03 01 11 00 02 11 01 03 11 01 ff c4 00 1d | .....
00001dc0: 00 00 02 03 00 03 01 01 00 00 00 00 00 00 00 00 | .....
00001dd0: 00 02 03 00 01 04 05 06 07 08 09 ff c4 00 4a 10 | .....J.
00001de0: 00 02 01 03 03 02 05 02 04 05 04 01 03 01 00 13 | .....
00001df0: 01 02 11 00 03 21 04 12 31 41 51 05 06 13 22 61 | .....!..1AQ...a
00001e00: 07 71 32 81 91 a1 08 14 23 42 b1 52 c1 d1 f0 e1 | .q2....#B.R.....
00001e10: 15 33 62 f1 16 24 17 72 43 18 25 34 82 92 a2 26 | .3b..$.rC.%4...&
00001e20: 27 53 35 36 63 b2 c2 ff c4 00 1b 01 01 01 01 01 | 'S56c.....
00001e30: 01 01 01 01 00 00 00 00 00 00 00 00 00 01 02 03 | .....
00001e40: 04 05 07 06 ff c4 00 2a 11 01 01 01 01 00 03 00 | .....*.....
00001e50: 03 01 01 01 00 03 00 00 07 00 01 11 02 03 21 31 | .....!1
```

Declarative, not imperative

```
seq:  
- id: title_len  
  type: u4  
- id: title  
  type: str  
  size: title_len  
  encoding: UTF-8  
- id: num_elements  
  type: u2  
- id: elements  
  type: u4  
  repeat: eos
```

```
title_len = io.read_uint32();  
title = io.read_utf8_string(title_len);  
num_elements = io.read_uint16();  
  
elements = new ArrayList();  
while (!io.eof()) {  
    int tmp = io.read_uint32();  
    elements.add(tmp);  
}
```

Kaitai Struct data types

- Built-in data types:
 - Integers
 - Floats
 - Unaligned bit integers and bit fields (0.6+)
 - Strings: fixed size, terminator-delimited, up to end of stream
 - Raw byte arrays
 - Enums
- User-defined data types

Kaitia Struct features

- Sequential parsing (“seq”)
- Out-of-order parsing (“instances”)
- Calculated attributes
- Checking for magic signatures (fixed content)
- Conditional parsing (“if”)
- Type switching on a condition (“switch”)
- Repetitions:
 - until the end of stream
 - predefined number of iterations
 - until a condition is met

Expression language to C++

```
seq:  
  - id: full_len  
    type: u4  
  - id: elements  
    type: element  
    repeat: expr  
    repeat-expr: (full_len - 4) / 6
```

```
m_full_len = m__io->read_u4le();  
int l_elements = (full_len() - 4) / 6;  
m_elements = new std::vector();  
m_elements->reserve(l_elements);  
for (int i = 0; i < l_elements; i++) {  
    m_elements->push_back(new element_t(m__io));  
}
```

Expression language to Python

```
seq:  
  - id: full_len  
    type: u4  
  - id: elements  
    type: element  
    repeat: expr  
    repeat-expr: (full_len - 4) / 6
```

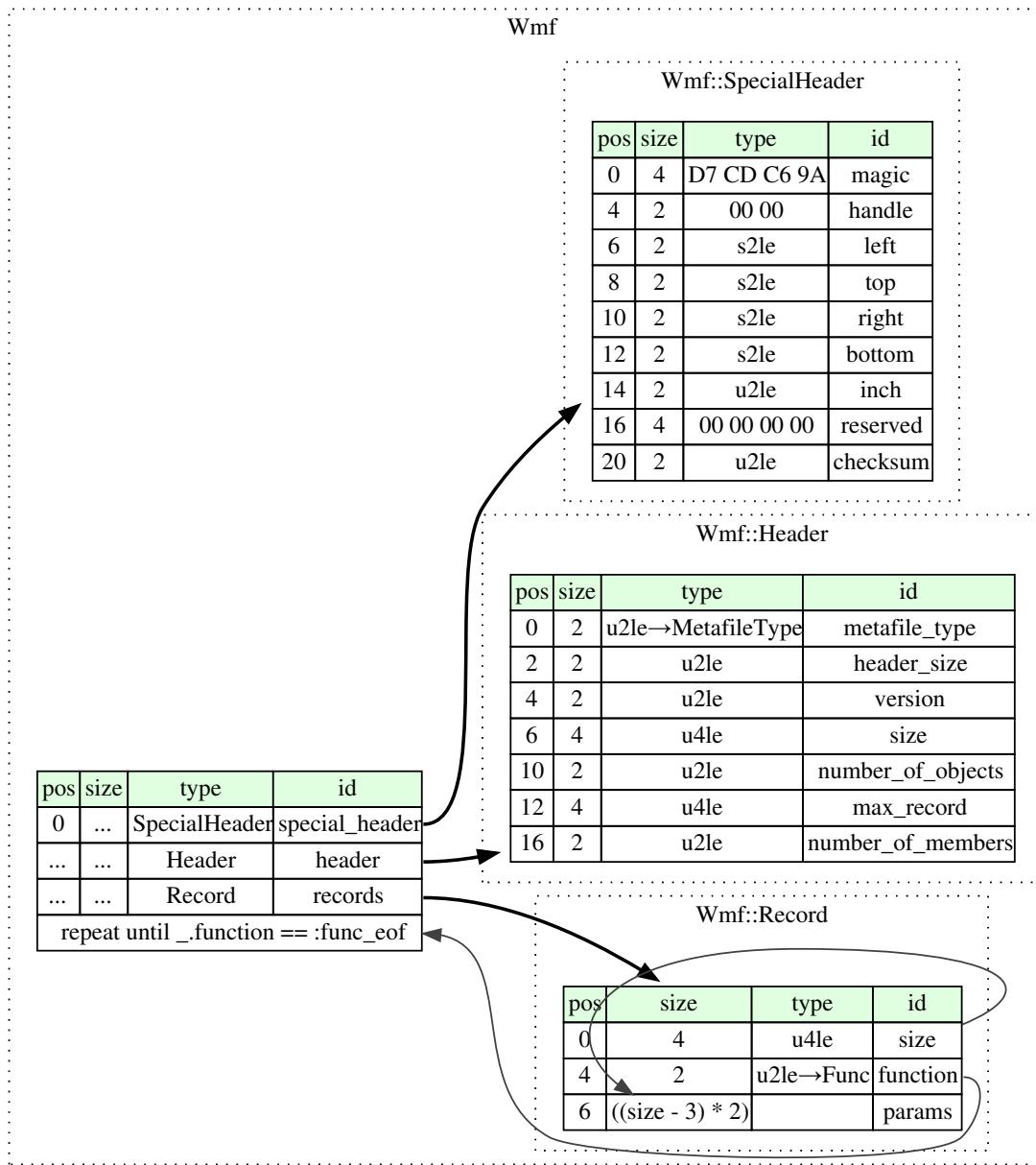
```
self.full_len = self._io.read_u4le()  
self.elements = [None] * ((self.full_len - 4) // 6)  
for i in range((self.full_len - 4) // 6):  
    self.elements[i] = Element(self._io)
```

Expression language to JavaScript

```
seq:  
  - id: full_len  
    type: u4  
  - id: elements  
    type: element  
    repeat: expr  
    repeat-expr: (full_len - 4) / 6
```

```
this.fullLen = this._io.readU4le();  
this.elements = new Array(Math.floor((this.fullLen - 4) / 6));  
for (var i = 0; i < Math.floor((this.fullLen - 4) / 6); i++) {  
  this.elements[i] = new Element(this._io);  
}
```

GraphViz visualization: WMF



Is it production-ready?

- We've got a growing repository of formats
- Image files: bmp, cr2, exif, gif, jpeg, pcx, png, tiff, tim (PlayStation), wmf, xwd
- Video files: Microsoft AVI (.avi), QuickTime .mov / MP4 / ISO/IEC 14496-14:2003
- Audio files: Standard MIDI (.mid), RIFF (.wav), ID3 tags, Amiga .mod modules
- More media files: Blender's .blend, 3D Systems Stereolithography (.stl)

And more...

- Archives: .lzh, .zip
- Documents: Microsoft's Compound File Binary (CFB, AKA OLE)
- Executables: DOS MZ, Windows PE, ELF, Mach-O, Python bytecode (.pyc), Java classes (.class), Adobe Flash (.swf)
- Filesystems: cramfs, ext2, iso9660, MBR partition tables, VirtualBox disk images (.vdi)
- Networking

Thanks for your attention! Questions?

<http://kaitai.io/>

GitHub: http://github.com/kaitai-io/kaitai_struct/

Twitter: @kaitai_io

Gitter: https://gitter.im/kaitai_struct/