fq

jq for binary formats

Mattias Wadman

Background

- Use various tools to extract data
    - ffprobe, gm identify, mp4dump, mediainfo, wireshark, one off programs, ...
- Convert to usable format and do queries
    - jq, grep, sqlite, sort, awk, sed, one off programs, ...
- Digging into and slicing binaries
    - Hexfiend, hexdump, dd, cat, one off programs, ...

Wishlist

"Want to see everything about this picture except the picture"

- A very verbose version of file(1)
- gdb for files
- Select and query things using a language
- Make all parts of a file symbolically addressable
- Nested formats and binaries
- Convenient bit-oriented decoder DSL

Experiments and prototypes

- Decoder DSL
    - TCL, lisp, tengo, Starlark, JavaScript, Go
- Query language
    - JSONPath, SQL, jq, JavaScript
- How to use
    - IR-JSON: `fq file | jq ... | fq`
    - Extend existing project
    - Decode and query in same tool

Result

Go

- Tests showed fast enough to decode big files
- Found gojq
- Previous good experience
- Good tooling

jq

"The JSON indenter"

- JSON in/out
- Syntax kind of a superset of JSON with same types
- Functional language based on generators and backtracking
    - Expressions can return or "output" zero, one or more values
    - No more outputs backtracks
- Implicit input and output similar to shell pipes
- Extraordinary iteration and combinatorial abilities
- Great for traversing tree structures

Examples

```
# Literals
> 123
123

> "abc"
"abc"

> [1,2,3]
[
  1,
  2,
  3
]

> {a: (1+2+3), b: ["abc", false, null]}
{
  "a": 6,
  "b": [
    "abc",
    false,
    null
  ]
}
```

Examples

```
# Pipeline using pipe operator "|" and identity function "." for current input
> "hello" | length | . * 2
10

# Multiple outputs using output operator ","
> 1, 2 | . * 2
2
4

# Index array or object using .[key/index] or just .key for objects
> [1,2,3][1]
2

# Collect outputs into array using [...]
> [1,empty,2]
[1,2]

# Iterate array or object using .[]
> [[1,2,3][]]
[1,2,3]
```

Examples

```
# Generators and backtracking
> 1, (2, 3 | . * 2), 4
1
4
6
4

# Conditional, boolean operators and comparsion
> if 1 == 2 and true then "a" else "b" end
"b"

# Reduce and foreach
> reduce (1,2,3) as $i (0; . + $i)
6
> foreach (1,2,3) as $i (0; . + $i; .)
1
3
6

# Bindings (variables)
> 1 as $a | 2 as $b | $a + $b
3
```

Examples

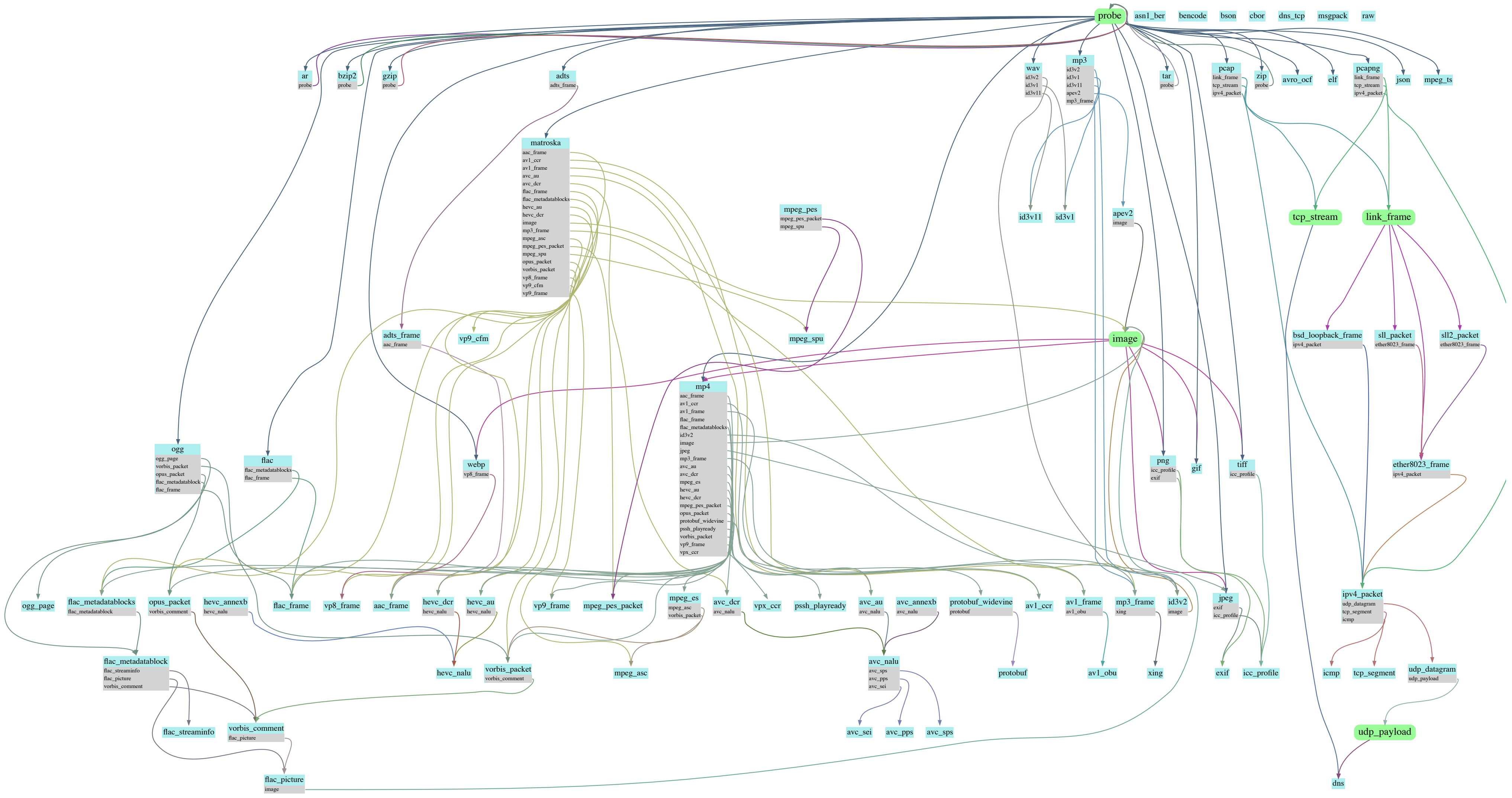
```
# Function using lambda argument. map from standard library:
def map(f): [.[] | f];
> [1,2,3] | map(. * 2)
[
  2,
  4,
  6
]
# select from standard library:
def select(f): if f then . else empty end;
> [1,2,3] | map(select(. % 2 == 0))
[
  2
]

# Function using argument binding and recursion to output multiple values
def down($n):
  if $n >= 0 then $n, down($n-1)
  else empty
  end;
```

fq

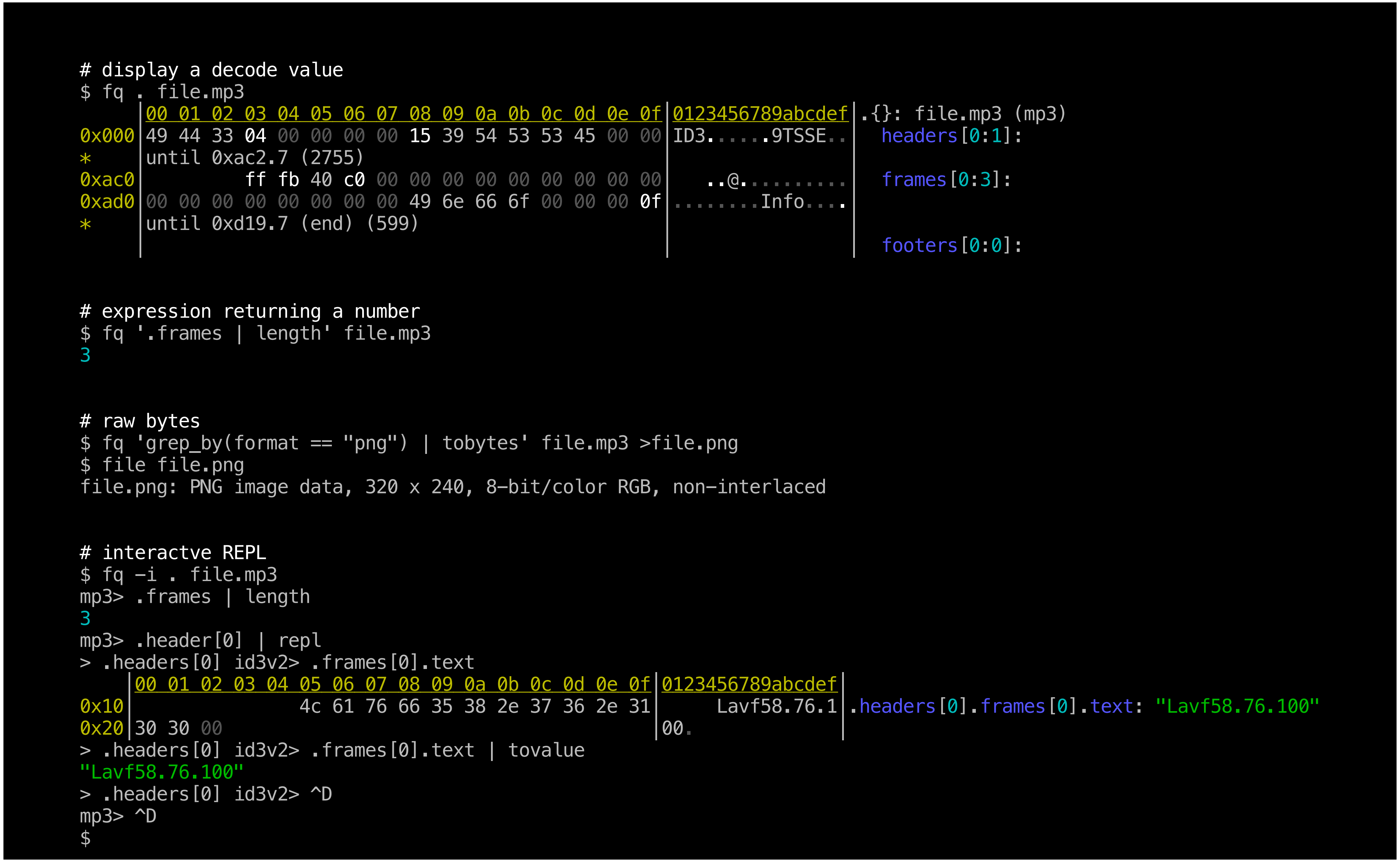
"The binary indenter"

- Superset of jq
- Re-implements most of jq's CLI interface
- 83 input formats, 22 supports probe
- Additional standard library functions
- Additional types that act as standard jq types but has special abilities
    - *Decode value* has bit range, actual and symbolic value, description, ...
    - *Binary* has a unit size, bit or bytes, and can be sliced
- Output fancy hexdump, JSON and binary
- Interactive REPL with completion and sub-REPL support


probe
asn1_ber
bencode
bson
cbor
dns_tcp
msgpack
raw
ar
probe
bzip2
probe
gzip
probe
adts
adts_frame
wav
id3v2
id3v1
id3v11
mp3
id3v2
id3v1
id3v11
apev2
mp3_frame
tar
probe
pcap
link_frame
tcp_stream
ipv4_packet
zip
probe
avro_ocf
elf
pcapng
link_frame
tcp_stream
ipv4_packet
json
mpeg_ts
matroska
aac_frame
av1_ccr
av1_frame
avc_au
avc_dcr
flac_frame
flac_metadatablocks
hevc_au
hevc_dcr
image
mp3_frame
mpeg_asc
mpeg_pes_packet
mpeg_spu
opus_packet
vorbis_packet
vp8_frame
vp9_cfm
vp9_frame
mpeg_pes
mpeg_pes_packet
mpeg_spu
id3v11
id3v1
apev2
image
tcp_stream
link_frame
adts_frame
aac_frame
vp9_cfm
mpeg_spu
image
bsd_loopback_frame
ipv4_packet
sll_packet
ether8023_frame
sll2_packet
ether8023_frame
mp4
aac_frame
av1_ccr
av1_frame
flac_frame
flac_metadatablocks
id3v2
image
jpeg
mp3_frame
avc_au
avc_dcr
mpeg_es
hevc_au
hevc_dcr
mpeg_pes_packet
opus_packet
protobuf_widevine
pssh_playready
vorbis_packet
vp9_frame
vpx_ccr
ogg
ogg_page
vorbis_packet
opus_packet
flac_metadatablock
flac_frame
flac
flac_metadatablocks
flac_frame
webp
vp8_frame
png
icc_profile
exif
gif
tiff
icc_profile
ether8023_frame
ipv4_packet
ogg_page
flac_metadatablocks
flac_metadatablock
opus_packet
vorbis_comment
hevc_annexb
hevc_nalu
flac_frame
vp8_frame
aac_frame
hevc_dcr
hevc_nalu
hevc_au
hevc_nalu
vp9_frame
mpeg_pes_packet
mpeg_es
mpeg_asc
vorbis_packet
avc_dcr
avc_nalu
vpx_ccr
pssh_playready
avc_au
avc_nalu
avc_annexb
avc_nalu
protobuf_widevine
protobuf
av1_ccr
av1_frame
av1_obu
mp3_frame
xing
id3v2
image
jpeg
exif
icc_profile
ipv4_packet
udp_datagram
tcp_segment
icmp
flac_metadatablock
flac_streaminfo
flac_picture
vorbis_comment
hevc_nalu
vorbis_packet
vorbis_comment
mpeg_asc
avc_nalu
avc_sps
avc_pps
avc_sei
protobuf
av1_obu
xing
exif
icc_profile
icmp
tcp_segment
udp_datagram
udp_payload
flac_streaminfo
vorbis_comment
flac_picture
avc_sei
avc_pps
avc_sps
udp_payload
flac_picture
image
dns

Usage

- Basic usage
  - `fq . file`, `cat file | fq`
- Multiple input files
  - `fq 'grep_by(format == "exif")' *.png *.jpeg`
- Hexdump, JSON and binary output
  - `fq '.frames[10] | d' file.mp3`
  - `fq '[grep_by(format == "dns").questions[].name.value]' file.pcap`
  - `fq 'first(grep_by(format == "jpeg")) | tobytes' file > file.jpeg`
- Interactive REPL
  - `fq -i . *.png`



```
# display a decode value
$ fq . file.mp3
     |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|.{}: file.mp3 (mp3)
0x000|49 44 33 04 00 00 00 00 15 39 54 53 53 45 00 00|ID3......9TSSE..|  headers[0:1]:
*    |until 0xac2.7 (2755)                           |                |
0xac0|         ff fb 40 c0 00 00 00 00 00 00 00 00 00|   ..@.........|  frames[0:3]:
0xad0|00 00 00 00 00 00 00 00 49 6e 66 6f 00 00 00 0f|........Info....|
*    |until 0xd19.7 (end) (599)                      |                |
     |                                               |                |  footers[0:0]:


# expression returning a number
$ fq '.frames | length' file.mp3
3


# raw bytes
$ fq 'grep_by(format == "png") | tobytes' file.mp3 >file.png
$ file file.png
file.png: PNG image data, 320 x 240, 8-bit/color RGB, non-interlaced


# interactve REPL
$ fq -i . file.mp3
mp3> .frames | length
3
mp3> .header[0] | repl
> .headers[0] id3v2> .frames[0].text
    |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|
0x10|               4c 61 76 66 35 38 2e 37 36 2e 31|     Lavf58.76.1|.headers[0].frames[0].text: "Lavf58.76.100"
0x20|30 30 00                                       |00.             |
> .headers[0] id3v2> .frames[0].text | tovalue
"Lavf58.76.100"
> .headers[0] id3v2> ^D
mp3> ^D
$
```

fq specific functions

- Standard library
    - `streaks`, `count`, `delta`, `chunk`, `diff`, `grep`, `grep_by`, ...
    - `toradix`, `fromradix`, `hex`, `base64`, ...
- Decode value
    - `display` (alias `d`, `dv`, `da` ...)
    - `parent`, `format`, ...
    - `tobytes`, `tovalue`, `toactual`, ...
    - `torepr`, ...
- Binary
    - Regexp functions `test`, `match`, ...
    - Decode functions `probe`, `mp3_frame`, ...

Binary and binary array

- A binary is created using `tobits`, `tobytes`, `tobitsrange` or `tobytesrange`.
    - From decode value `.frames[1] | tobytes`
    - String or number `"hello" | tobits`
    - Binary array `[0xab, ["hello", .name]] | tobytes`
- Can be sliced using normal jq slice syntax.
    - `"hello" | tobits[8:8+16]` are the bits for `"el"`
- Can be decoded
    - `[tobytes[-10:], 0, 0, 0, 0] | flac_frame`

Example queries

- Slice and decode
  - `tobits[8:8+8000] | mp3_frame | d`
  - `match([0xff,0xd8]) as $m | tobytes[$m.offset:] | jpeg`
- ASN1 BER, CBOR, msgpack, BSON, ... has `torepr` support
  - `fq -d cbor torepr file.cbor`
  - `fq -d msgpack '[torepr.items[].name]' file.msgpack`
- PCAP with TCP reassembly, look for GET requests
  - `fq 'grep("GET .*")' file.pcap`
- Parent of scalar value that includes bit 100
  - `grep_by(scalars and in_bits_range(100)) | parent`

Use as script interpreter

```
#!/usr/bin/env fq -d mp4 -f

( first(.boxes[] | select(.type == "moov")?)
| first(.boxes[] | select(.type == "mvhd")?) as $mvhd
| { time_scale: $mvhd.time_scale,
    duration: ($mvhd.duration / $mvhd.time_scale),
    tracks:
      [ .boxes[]
      | select(.type == "trak")
      | [("mdhd", "stsd", "elst") as $t | first(grep_by(.type == $t))] as [$mdhd, $stsd, $elst]
      | { data_format: $stsd.boxes[0].type,
          media_scale: $mdhd.time_scale,
          edit_list:
            [ $elst.entries[]
            | { track_duration: (.segment_duration / $mvhd.time_scale),
                media_time: (.media_time / $mdhd.time_scale)
              }
            ]
        }
      ]
  }
)
```

Use as script interpreter

```
$ ./editlist file.mp4
{
  "duration": 60.095,
  "time_scale": 600,
  "tracks": [
    {
      "data_format": "mp4a",
      "edit_list": [
        {
          "media_time": 0,
          "track_duration": 60.095
        }
      ],
      "media_scale": 22050
    },
    {
      "data_format": "avc1",
      "edit_list": [
        {
          "media_time": 0,
          "track_duration": 60.095
        }
...
```

Implementation

- Library of jq function implemented in Go
  - Decoders, decode value, binary, bit reader, IO, tty, ...
- CLI and REPL is mostly written in jq

```
( open
| decode
| if $repl then repeat(read as $expr | eval($expr) | print)
  else eval($arg) | print
  end
)
```

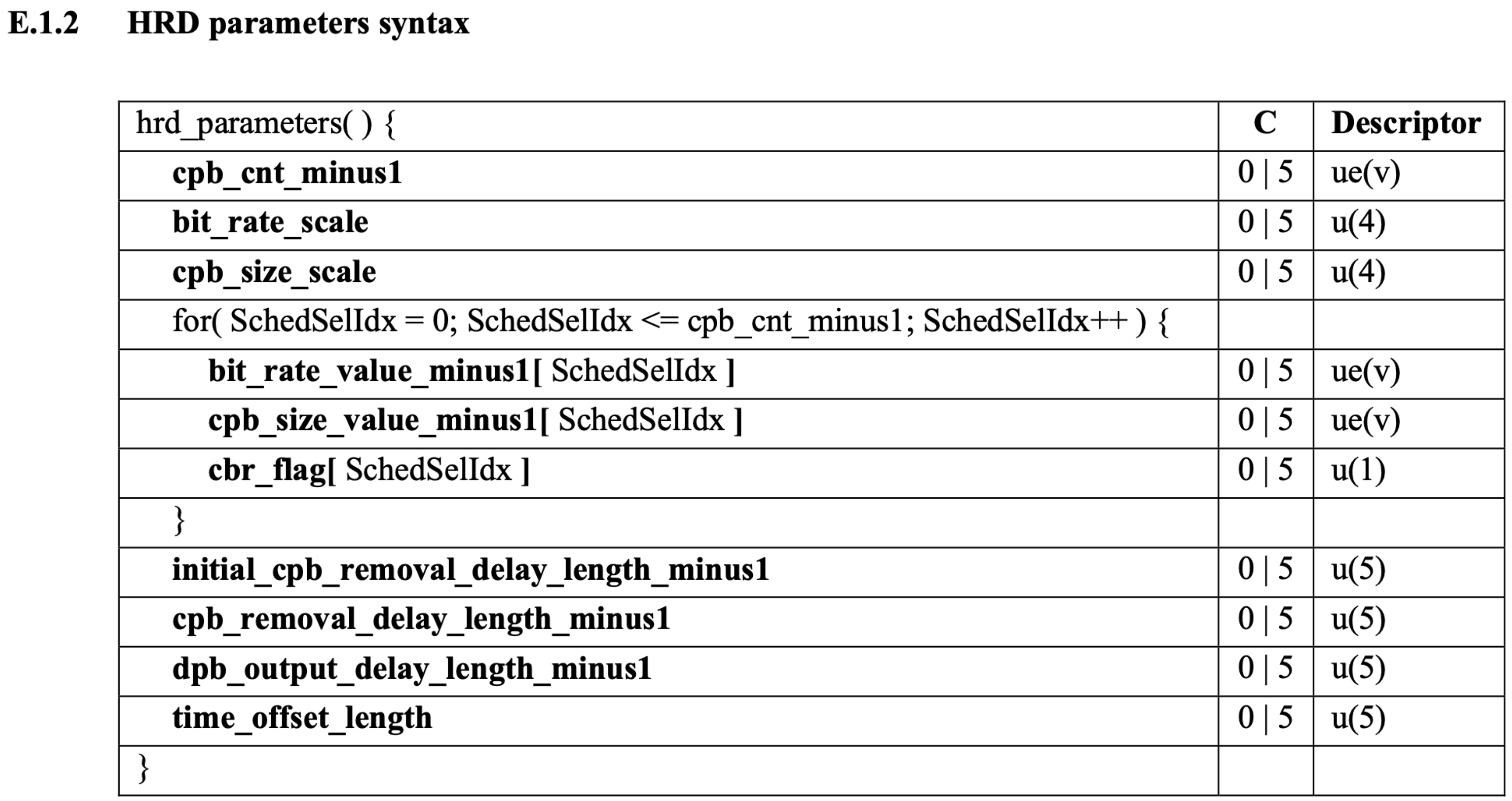
- All current decoders in Go
- Uses a forked version of gojq
  - Helped add native functions and iterators support
  - JQValue interface, bin/hex/oct literals, reflection, query AST functions, ...

Decode API

SPS HRD parameters from ITU-T H.264 specification

```
func avcHdrParameters(d *decode.D) {
    cpbCnt := d.FieldUFn("cpb_cnt", uEV, scalar.UAdd(1))
    d.FieldU4("bit_rate_scale")
    d.FieldU4("cpb_size_scale")
    d.FieldArray("sched_sels", func(d *decode.D) {
        for i := uint64(0); i < cpbCnt; i++ {
            d.FieldStruct("sched_sel", func(d *decode.D) {
                d.FieldUFn("bit_rate_value", uEV, scalar.UAdd(1))
                d.FieldUFn("cpb_size_value", uEV, scalar.UAdd(1))
                d.FieldBool("cbr_flag")
            })
        }
    })
    d.FieldU5("initial_cpb_removal_delay_length", scalar.UAdd(1))
    d.FieldU5("cpb_removal_delay_length", scalar.UAdd(1))
    d.FieldU5("dpb_output_delay_length", scalar.UAdd(1))
    d.FieldU5("time_offset_length")
}
```

Decode API



E.1.2 HRD parameters syntax

hrd_parameters( ) {	**C**	**Descriptor**
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Decode API

Formats can use other formats. Simplified version of mp3 decoder:

```
func decode(d *decode.D, in interface{}) interface{} {
    d.FieldArray("headers", func(d *decode.D) {
        for !d.End() {
            d.TryFieldFormat("header", headerGroup)
        }
    })

    d.FieldArray("frames", func(d *decode.D) {
        for !d.End() {
            d.TryFieldFormat("frame", mp3Group)
        }
    })

    d.FieldArray("footers", func(d *decode.D) {
        for !d.End() {
            d.TryFieldFormat("footer", footerGroup)
        }
    })

    return nil
}
```

Future

- Declarative decoding like kaitai struct, decoder in jq
- Nicer way to handle checksums, encoding, validation etc
- Schemas for ASN1, protobuf, ...
- Better support for modifying data
- More formats like tls, http, http2, grpc, filesystems, ...
- Encoders
- More efficient, lazy decoding, smarter representation
- GUI
- Streaming input, read network traffic `tap("eth0") | select(...)`?
- Hope for more contributors

Thanks and useful tools

- @itchyny for gojq
- Stephen Dolan and others for jq
- HexFiend
- GNU poke
- Kaitai struct
- Wireshark
- vscode-jq (https://github.com/wader/vscode-jq)
- jq-lsp (https://github.com/wader/jq-lsp)

Thank you

jq for binary formats

Mattias Wadman
mattias.wadman@gmail.com (mailto:mattias.wadman@gmail.com)
https://github.com/wader/fq (https://github.com/wader/fq)
@mwader (http://twitter.com/mwader)