Toward Open TAIGA Raw Data Formats

I. Bychkov[1], O. Fedorov[2], A. Hmelnov[1],
E. Korosteleva[3], D. Kostunin[4], Kryukov[3],
A. Mikhailov[*,4], Alexey Shigarov[**,4]
[*]mikhailov@icc.ru, [**]shigarov@icc.ru

[1]ISDCT SB RAS, Irkutsk, Russia
[2]API ISU, Irkutsk, Russia
[3]SINP MSU, Moscow, Russia
[4]INP KIT, Karlsruhe, Germany

Joint Meeting on Karlsruhe-Russian Astroparticle
Data Life Cycle Initiative (RSF-Helmholtz Project)
October 29-30, 2018

Motivation

Best practices in scientific data curation recommend *"Keep Raw Data Raw"*. Why?

- **Reproducibility**: *Our peers can confirm our results*
- **Reuse**: *We can use old raw data with new methods in future*

Badly-Curated Raw Data Formats

- No documentation or poor (incomplete) documentation
- Unpublished (proprietary) software for reading data
- Risk to lose the source code

Well-Curated Raw Data Formats

- Well-formed documentation and/or metadata
- Open data access APIs for popular software development platforms (e.g. `C++`, `Java`, or `Python`)

Motivation

TAIGA facilities generate raw data in 5 unique file formats

- `TAIGA-IACT`
- `Tunka-HiSCORE`
- `Tunka-133` and `GRANDE`
- `Tunka-REX`

The main reason to start this work

- *"There are two experts in the world: one knows how to write the raw data and second knows how to read them"*

Our goal is to open TAIGA file formats

- Open metadata for specifying the formats
- Open software (APIs) for parsing and verifying the raw data

Solution

1. Explore the raw data

```
FD 3F C5 5C 20 9D B1 A2 3A 4D A8 B9
3F 45 04 6D DE E2 89 EC 67 71 74 14
09 80 F7 80 D0 A1 95 B8 95 66 F8 2A
53 8B 57 3D 8F F7 7F 44 69 20 C3 9A
```

2. Write a file format specification

```
meta:
  id: hiscore
  title: HiSCORE
seq:
  - id: packages
    type: package
    repeat: eos
types:
  package:
    seq:
      - id: hdr
        type: header
        ...
```

3. Get an auto-generated source code for parsing binary data in a target programming language

```
...
vector<hiscore_t::package_t*>* packages = hiscore.packages();
vector<hiscore_t::package_t*>::iterator it = packages->begin();
// Print some infos
for (it; it != packages->end(); ++it) {
  hiscore_t::package_t* package = (hiscore_t::package_t*)*it;
  hiscore_t::header_t* header = package->hdr();
  printf("Event number: %d\n", header->event_number());
  printf("IP:           %d\n", header->ip());
  printf("Magic:        %d\n", header->magic());
}
...
```

Binary Data Format Description Languages

- `DFDL` (Data Format Description Language)
  - developed by *OGF* (Open Grid Forum) and *IBM*
  - XML-based (too complicated to be read by human)
  - `Daffodil`, *OGF* implementation
    - `Java/Scala` API to interpret spec.
    - free & open-source
- `FlexT`, a binary data description language
  - developed by *ISDCT SB RAS*
  - original expressive syntax (human-readable spec.)
  - generates source code by spec. in `Delphi` and `C++`
  - proprietary
- `Kaitai Struct`, a declarative binary format parsing language
  - developed by *Mikhail Yakshin*
  - YAML-based (human-readable spec.)
  - generates source code by spec. in `C++`, `Java`, `Python`, etc.
  - free & open-source

Raw Data Format Specification (KS Sample)

`HiSCORE` format specification expressed in `Kaitai Struct`

Part I

```
meta:
  id: hiscore
  title: HiSCORE data
  license: Unlicensed
seq:
  - id: packages
    type: package
    repeat: eos
types:
  package:
    seq:
    - id: hdr
      type: header
    - id: data
      type: data_block
      repeat: expr
      repeat-expr: 9
    - id: end_of_package
      contents:
      [0xFF, 0xFF, 0xFF, 0xFF]
```

Part II

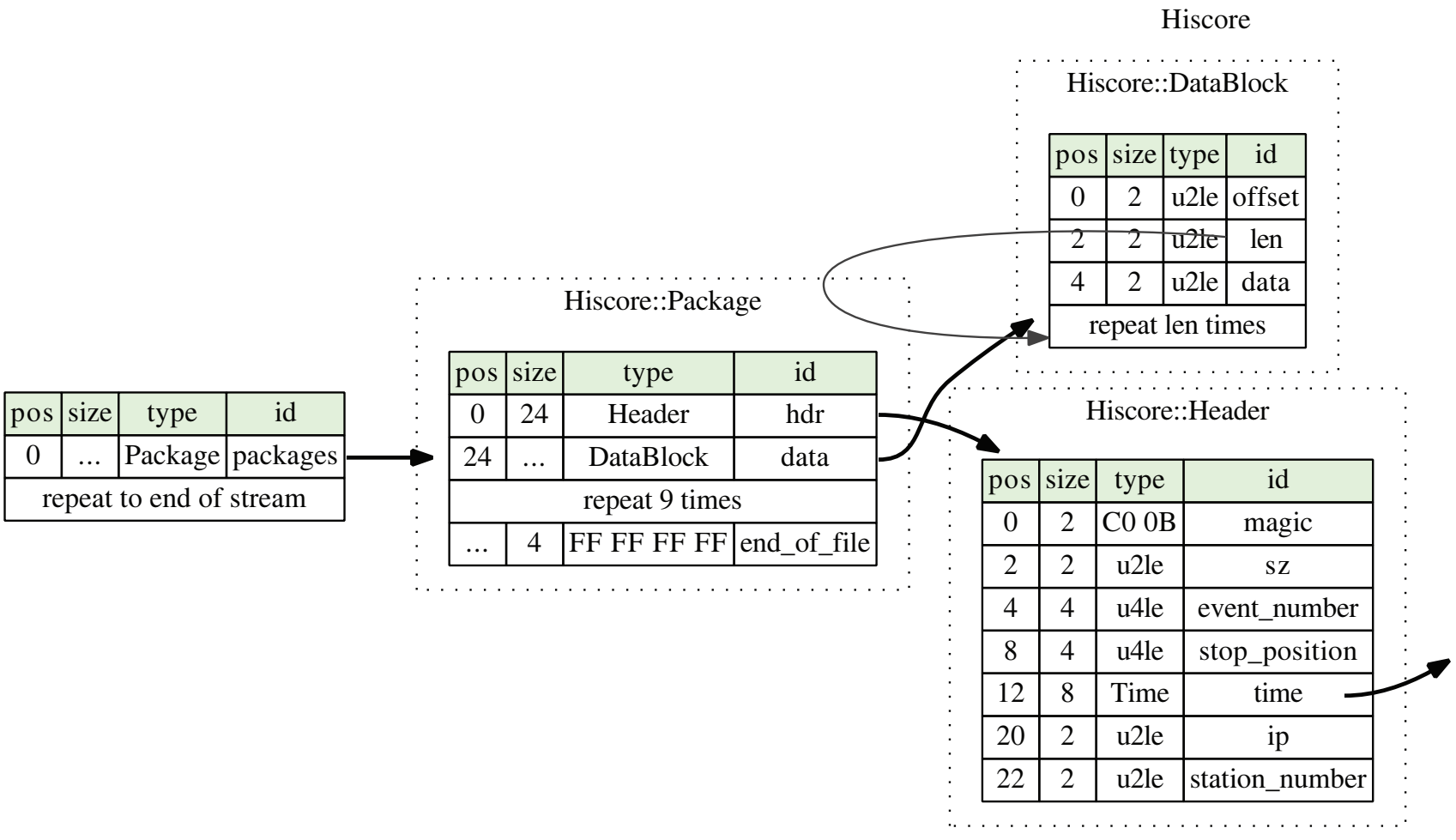
```
header:
  seq:
  - id: magic
    contents: [0xC0, 0x0B]
  - id: sz
    type: u2le
  - id: event_number
    type: u4le
  - id: reserved
    type: u4le
  - id: time
    type: time
  - id: ip
    type: u2le
  - id: station_number
      type: u2le
data_block:
  seq:
  - id: offset
  type: u2le
  - id: len
    type: u2le
  - id: data
    type: u2le
    repeat: expr
    repeat-expr: len
```

Part III

```
time:
  seq:
  - id: dat0
    type: u2le
  - id: dat1
    type: u2le
  - id: dat2
    type: u2le
  - id: dat3
    type: u2le
  instances:
    dns:
    value: '(dat0 & 0x7f) * 10'
    mks:
    value: '((dat0 & 0xff80) >> 7) | (dat1 & 1) << 9'
    mls:
    value: '(dat1 & 0x7fe) >> 1'
    s:
    value: '((dat1 & 0xf800) >> 11) | ((dat2 & 1) << 5)'
    m:
    value: '(dat2 & 0x7e) >> 1'
    h:
    value: '(dat2 & 0xf80) >> 7'
```

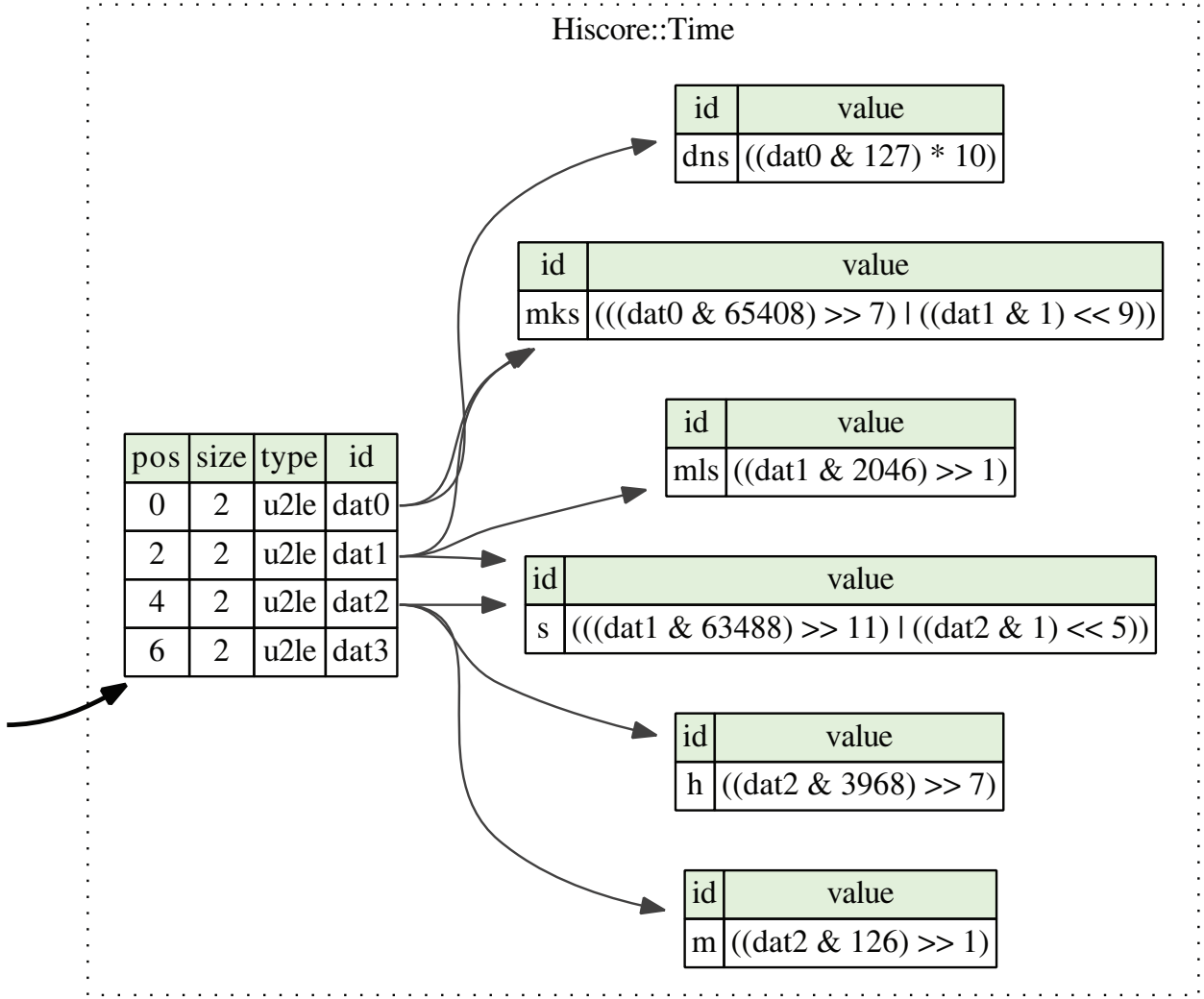
Auto-Generated Raw Data Format Diagram (Part I)

HiSCORE format KS-specification as a diagram



Auto-Generated Raw Data Format Diagram (Part II)

HiSCORE format KS-specification as a diagram (continue)



Auto-Generated Raw Data Parsing Libraries (C++ Sample)

Source code of `HiSCORE` data parsing lib (a fragment of `.h` file)

```
class hiscore_t : public kaitai::kstruct {
public:
    class package_t;
    class header_t;
    class data_block_t;
    class time_t;

    hiscore_t(kaitai::kstream* p__io, kaitai::kstruct* p__parent = 0, hiscore_t*
          p__root = 0);

private:
    void _read();
public:
    ~hiscore_t();
    class package_t : public kaitai::kstruct {
    public:
        package_t(kaitai::kstream* p__io, hiscore_t* p__parent = 0, hiscore_t*
             p__root = 0);
    private:
        void _read();
...
```

Raw Data Reading using Generated API (C++ Sample)

```
// Add KS runtime library
...
#include <kaitaistruct.h>
...
// Include generated class
#include "hiscore.h"
...
ifstream ifs(fileName, ifstream::binary);
// Make Kaitai Struct stream
kaitai::kstream ks(&ifs);
// Read HiSCORE file by generated library
hiscore_t hiscore = hiscore_t(&ks);
// Get all packages
vector<hiscore_t::package_t*>* packages = hiscore.packages();
vector<hiscore_t::package_t*>::iterator it = packages->begin();
// Print some infos
for (it; it != packages->end(); ++it) {
  hiscore_t::package_t* package = (hiscore_t::package_t*)*it;
  hiscore_t::header_t* header = package->hdr();
  printf("Event number: %d\n", header->event_number());
  printf("IP:           %d\n", header->ip());
  printf("Magic:        %d\n", header->magic());
}
```

Results

- All file format specifications (`HiSCORE`, `IACT`, `T133`, `GRANDE`, and `TREX`)
- File format diagrams
- Parsing libraries in `C++`, `Java`, `Python`, etc.
- Reading samples in `C++`

Testing on Real Raw Data

- `T133`, `GRANDE`, and `TREX` spec. were tested on $\approx$ **89K** files
- `HiSCORE` and `IACT` spec. were tested on $\approx$ **120K** files

Raw Data Verification

- **1.2 %** of `T133`, `GRANDE`, and `TREX` files have **BAD** format
- **0.6 %** of `HiSCORE` and `IACT` files have **BAD** format

Conclusion & Further Work

Advantages of Open TAIGA raw data formats

- Make TAIGA raw data well-curated (reproducible and reusable)
- Exclude the reverse engineering of the formats in future
- Multi-messenger analysis
  - Raw data aggregation from various sources
  - Raw data exchange between astroparticle physics data services

Further work

- Add human-understandable comments to the specifications
- Describe formally directory structures of TAIGA raw data
- Involve the specification in TAIGA metadata management

Thanks

This work was financially supported by RSF Grant No. 18-41-06003