Discussion: CORSIKA8 Output Format

Remy L. Prechelt

July 30, 2020

Summary

- Ralf put together a very comprehensive Google Doc containing some detailed discussion and many useful links here.
- I've extracted some of the key discussion points into these slides to help start the meeting discussion.
- All of this information is taken from the Google Doc so they don't necessarily reflect my (personal) opinions.
- These slides are a WIP! As we have the discussion during the meeting, I will update the content and post an updated version to the Indico afterwards.

Why We Need a Default Format

- While some large collaborations (CTA, IceCube, etc.) will want to develop their own custom output formats to interface with their infrastructure, there is a large body of users who will want to run C8 and immediately get production-quality output in a standard format suitable for analysis.
- The actual implementation (currently as part of the `Detector` hierarchy discussed last week) will allow users to implement custom output formats relatively easily (to support these larger collaborations).

What should be the default file format for "standard" C8 users?

Things to Consider

A lot of the discussions in the Google Doc are framed around the three standard I/O use cases:

1. Writing data
    - Since most of the C8 usage will be *write-once*, this is probably the least important use-case.
2. Sharing & distributing data.
    - This includes sharing a library of showers, a single shower, or just a specific subcomponent (i.e. radio waveforms) of a shower.
3. Reading data.
    - This is probably the most important considering the longevity of large shower libraries (data is saved once but analyzed many times).

Wish List

These are the I/O wish list items currently in the GDoc:

1. Human-readable metadata.
2. Accepts all types of (raw) data.
3. Robust against failures (native checksumming?)
4. Flexible on different systems (portable).
5. Fast (parallel I/O).
6. Easy to share & distribute.
7. Explicit versioning (archival ready).
8. Hierarchical - groups, subdirectories, etc..

Wish List (cont.)

9. Simple to develop and maintain.
10. Useable from at least C++ and Python.
11. ???

File Formats

The file formats that have currently been proposed (with links):

Single-file binary or text+binary:

- HDF5
- ROOT
- ASDF
- Parquet
- Protobuf

Filesystem-based formats:

- A standardized filesystem-based format like Exdir (or sim.)
- A completely custom filesystem-based format.
- The C7 directory/file format.

Single File vs. Filesystem-based Formats

The formats that are under consideration fall into two categories:

- Single file formats (HDF5, ROOT, Parquet, ASDF, ProtoBuf) with an internal hierarchy. These are typically binary or text+binary (ASDF).
  - Each of these formats has their own advantages and disadvantages (outlined below).
  - Easy to share the whole file but can be difficult to share a specific subcomponent.
- Filesystem-based formats where the hierarchy is created using directories on the filesystem (ExDir, C7, other).
  - This includes a completely custom format or something that is based on an open-source standard like ExDir.

Filesystem-based Formats

Advantages:

- Filesystems are easy to work with and allow for easy inspection of the full hierarchy.
- Metadata is often text-based so is inspectable with standard Unix tools.
- Easy to extract a specific subcomponent to share if a user doesn't need access to the full data output.
- ???

Disadvantages:

- (Midly) awkward to share large shower libraries (tar, share, untar).
- Without standardization between subcomponents, can easily become convoluted to read. See the motivation for the Exdir format (standardized metadata formats).
- ???

ExDir

ExDir is a WIP open-source standard for hierarchical array data. It is based on the HDF5 model but uses filesystem directories instead of HDF5 groups.

Advantages:

- Uniform directory structure with extensive YAML metadata that is inspectable with Unix tools.
- Stores arrays in ‘.npy‘ files that are easily read in any language (simple format).
- Directly convertible to a HDF5 file for sharing (still in beta).
- ???

Disadvantages:

- Still WIP and the project has not been updated in over a year.
- Limited support for non-array-like data.
- ???

ROOT

Advantages:

- Extremely fast to read and write (fastest proposed I/O format) - both sequentially and in-parallel.
- Good support for various compression algorithms on only the needed subcomponents.
- Good support for checksumming to guarantee (long term) data accuracy.
- With `uproot`, reading can be done without ROOT installed - only machines that actually run C8 would require ROOT.

Disadvantages:

- Introduces a hard-dependency on a large software library that has many versions still in use - this will require extensive CI testing.
- Can we guarantee backwards compatibility over several ROOT & CORSIKA versions?
- ???

HDF5

Advantages:

- Relatively fast to read and write (ROOT and Parquet are still faster)
- Widespread use in a wide number of fields - already used for CoREAS output from C7.
- Has a sophisticated group and hierarchy system.
- Basic support for row-wise and column-wise compression (difficult to tune).
- Good support for N-dimensional arrays.
- ???

Disadvantages [a]:

- Cannot work/extract/modify with standard Unix tools - everything must go through HDF5-specific tools.
- Introduces a hard-dependency on a large software library.
- Parallel I/O performance is not great and can be difficult to tune.
- Very hard to extract a specific subcomponent (deleting a group does not remove it from the file).
- ???

[a] Moving Away from HDF5

Other Formats

There are several other formats that were considered that are (probably) not worth discussing in detail due to various major weaknesses (for C8) - they are included in the following slides for reference. Let me know if you disagree.

Parquet

Parquet is a columnar storage format for the Apache Hadoop ecosystem:

Advantages:

- Very fast sequential & parallel I/O (second only to ROOT).
- Very fast compression & decompression.
- Instantaneous conversion to Apache Arrow for in-memory processing.
- ???

Disadvantages:

- Another hard dependency on an additional software library - multiple C++ implementations (neither are 100% feature complete).
- Poor support for hierarchy and grouping in a single file.
- Poor or non-existent support for checksumming, compression, and reading particle files.
- ???

ASDF

ASDF is a new text+binary file format designed to replace FITS:

Advantages:

- Text-based YAML metadata at start of file is inspectable with Unix tools.
- It is a simple format that can be written/read without any large external libraries.
- ???

Disadvantages:

- Still very young and rapidly changing - may be too immature for C8.
- Poor or non-existent support for checksumming, compression, and reading partial files.
- ???

Protobuf

Advantages:

- Great support for versioning and backwards compatibility.
- Fast to serialize data structures.
- ???

Disadvantages:

- Not designed as a disk format (designed as a message serializer). The mapping to disk is not defined as part of the standard and is therefore not portable.
- Difficult to support custom data types as message specification is specified in custom DSL and then transpiled to C++ with `protoc`.
- ???