# MR 317: Output Architecture

R. Prechelt

April 8, 2021

# Summary

- MR:317 contains an implementation of:
  - A standardized output architecture for writing C8 data into a standard directory and file format.
  - An implementation of this architecture for several modules using Apache Parquet as the file format.
  - A Python analysis library that parses and loads the above structure.
- It has been available on Gitlab for ~6 weeks and all comments have been addressed - would like to merge by the end of this week.
- Currently an issue with the CI related to a dependency but that will be fixed soon.

# Directory Format

- Each C8 *library* (i.e. a collection of many showers) is stored in a single directory.
- Each module gets its own subdirectory that contains the data for *all the showers* in the libary (as well as metadata in `*.yaml`).

```
<library name>/
    config.yaml
    obsplane/ # name of module
        config.yaml
        summary.yaml
        particles.parquet
    radio/  # name of module
        config.yaml
        summary.yaml
        waveforms.parquet
    ...
```

# `OutputManager`

- The output system is coordinated by an instance of an `OutputManager`; this creates the directories, writes the metadata files, and organizes the C8 modules that want to write data.
- Modules (i.e. an `ObservationPlane`) are created as usual and are then registered with the `OutputManager` using the `add()` method.
- The `OutputManager` is now an additional argument to the `Cascade` (so that it can hook into the `Run()` method).

# `OutputManager` Implements a Simple FSM

- The `OutputManager` implements a simple state machine that allows modules to *hook* into the current simulation state:
  1. Start of Library (before the first shower)
  2. Start of Shower (start of each shower)
  3. End of Shower (end of each shower)
  4. End of Library (called after the last shower)
- At each state transition, the corresponding method from each registered output is called allowing modules to hook into the simulation state.
- At the end of the simulation, the `EndOfLibrary` method of the `OutputManager` must be called by the user to indicate that no more showers will be run; this closes out the outputs and ensures everything is written to disk.

# Output Interface

- To register with the `OutputManager`, each class must inherit from `BaseOutput` and *can* provide the following methods (*most* are optional):
  - `StartOfLibrary`
  - `StartOfShower`
  - `EndOfShower`
  - `EndOfLibrary`
  - `GetConfig`
  - `GetSummary`
- Classes only need to implement whatever they need for their particular use case.
- The only one that is required is `GetConfig` (which I will discuss later).

# Core C8 Modules - Can Swap Out File Format

- `ObservationPlane` and `TrackWriter` have already been ported to this new output system; other modules will follow next week.
- To allow the output writer (i.e. file format) to be swapped out, each of the above modules now has an additional template argument, `TOutputWriter`, that implements the actual file format.
- `TOutputWriter` is defaulted to Parquet writers so for most use cases you don't need to worry about templates (i.e. `TrackWriter writer;` will work). However, the file format can be easily replaced on a module-by-module basis, i.e. `TrackWriter<TrackWriterROOTWriter> writer;` if users want to write their own output writers.
- Future implementations should try to follow this pattern where possible.

# Config and Summary

- All modules must provide a `getConfig` method that returns a *complete* specification of their configuration (in YAML); this is written to a `config.yaml` file in the module directory.
- The goal is to make the config file contain all the information needed to reproduce a particular simulation setup/config.
- Modules can optionally provide a `getSummary` method which is called at the end of the last shower; this is written to a `summary.yaml` file for that module. This is intended to be used for text-based summary statistics or small datasets.

# Python Analysis Library

- The MR also provides a `corsika` Python library. This loads the directory format and provides access to the underlying data in a variety of formats.
- This is `pip`-installable and doesn't depend on the C8 C++ library.
- Pandas is the default format for tabular data but library can also provide access to raw NumPy arrays or Parquet blobs.

```
>>> lib = corsika.Library("path/to/library") # open a library
>>> lib.config  # full simulation configuration as dictionary
>>> # here, "obsplane" is the registered name of the module
>>> df = lib.get("obsplane1").astype() # data as pandas dataframe
>>> lib.get("obsplane2").astype("numpy") # raw data as NumPy array
```

# Questions?

Questions?