Daya Bay File I/O

Brett Viren, Simon Patton, Xinchun Tian

June 11, 2008

DRAFT[1]

Abstract

This note gives requirements for and describes the implementation of the Daya Bay file I/O mechanism and the format used to store objects in ROOT TFiles.

1 Introduction

User code does not need to know details of file formats since their data access is through Gaudi transient (TES) or the archive (AES) event stores[2]. However in the end, we must be able to write data to persistent storage for later read-back.

Work has already been done [1] in understanding basic read-back mechanism of Gaudi. This note builds on that otherwise focuses on writing data out to file.

2 Requirements

Requirements for file input/output.

Reproducibility : it must be possible to create files for later read-back in a way that reproduces the state of the TES and AES. Any algorithms that run downstream from where a file is written should be able to run in an identical manner whether they follow subsequent file read-back or the original "live" data production.

References : it must be possible to allow inter-object references to persist through file I/O.

Ownership : ownership through pointers must persist.

Selectability : it must be possible to limit what data is written out.

Flexibility : the types of data to be sent through I/O must be extensible.

[1] Draft status removed after implementation is complete and this note is updated to match.

In addition it is desirable, but desirable but not required that:

File streams : different streams of data can be written to different files with a mechanism to properly merge them on read-back.

Concatenation : files from distinct runs can be concatenated on input or output.

3 File Organization

Using ROOT for our low-level file format is a forgone conclusion. But, there is a need to settle on how to organize the data within the file. ROOT can store data in a variety of ways. The two simplest are:

Single tree: All data can be stored in a single tree[2] (aka. "ntuple"). This allows for optimized fast read-back of only the quantities one is interested in as well as browsing the file in a bare ROOT session. But, it also means that one must have a well defined "event" readout time and that all the data that comprises an "event" is well defined ahead of time and does not change from "event" to "event". Since Daya Bay does not have the concept of well defined events in all parts of its processing and because flexibility of data types is required, this is not an adequate model.

Blob chain: Each data object can be serialized one by one resulting in a simple linear chain of objects in the file. This allows for complete freedom on what data is written and when, but it allows for no optimized read-back nor does it lend itself to interrogating the data in a bare ROOT session.

3.1 Multiple Trees

Instead of these simple organizations, we use multiple trees, one for each `DataObject` in the TES. This solves several important problems:

- Each individual `DataObject` is well defined from the start which is a requirement for using ROOT trees.
- It allows the data model to be extended with new data types.
- It allows each `DataObject` to maintain its own production rate independent of the others.
- It allows for bare ROOT session browsing and optimized read-back.
- It makes possible the resyncing between trees on read-back (more on this below).

[2]Specifically a ROOT TTree class.

- It allows references and selectability.
- It support file streams as one can put the trees in separate files or merge them at read-back.

3.2 Synchronization during File Read-back

There are two types of synchronization needed during read-back.

Normal Normal read-back happens for post-trigger simulation and real detector data as well as intermediate simulation results where only a single type of kinematics is considered. In these cases, there is a single data stream and it is simple and linear in time.

Simulation Read-back of intermediate simulation results in the case where multiple kinematics types are mixed together using the "pull" staged processing method [3]. In this case the stage processors must access the data first and explicitly add it to the TES.

To support normal read-back synchronization, a ControlHeader object is introduced. It records what other HeaderObjects[4] are added to the TES and in what order. It is also the repository for any selections that determine if any given HeaderObject is to be written out to file or not. It is also saved to the output file in its own tree. Then during read-back it is used to correctly populate the TES as if the code that originally produced the file had been re-run.

For the special case of reading back intermediate data from a simulation run using multiple kinematics types, additional framework code is needed to allow for the data to be read back in the context of the "pull" processing model[3]. Each header object already carries an `execution number`[4] and this is sufficient to allow for proper ordering on read-back. However, in order to correctly populate the processing stages, it is the stage processors that must drive the read-back ordering. In principle, these processors could read from the input file directly but to maintain the Gaudi design philosophy that algorithms do not manage files directly a Gaudi data service is needed to stage this data to memory[3]. As a side effect of reading data from this service, the processors will fill the TES with the data in the same fashion as they do when they are instead generating "live" data.

This new store is dubbed the Simulation Event Store (SES). The SES must work slightly different than the TES and AES in the sense that it must load, on demand, new objects when their are accessed by path name. That is, the SES must work passively in response to requests driven by the stage processor algorithms. Each time a path is accessed, any previously existing object should be dropped and replaced by the appropriate one in the file with the next lowest execution number.

[3]This may but need not operate like an event store service.

4 File write out

Figure 1 shows a cartoon of how writing out to file works.

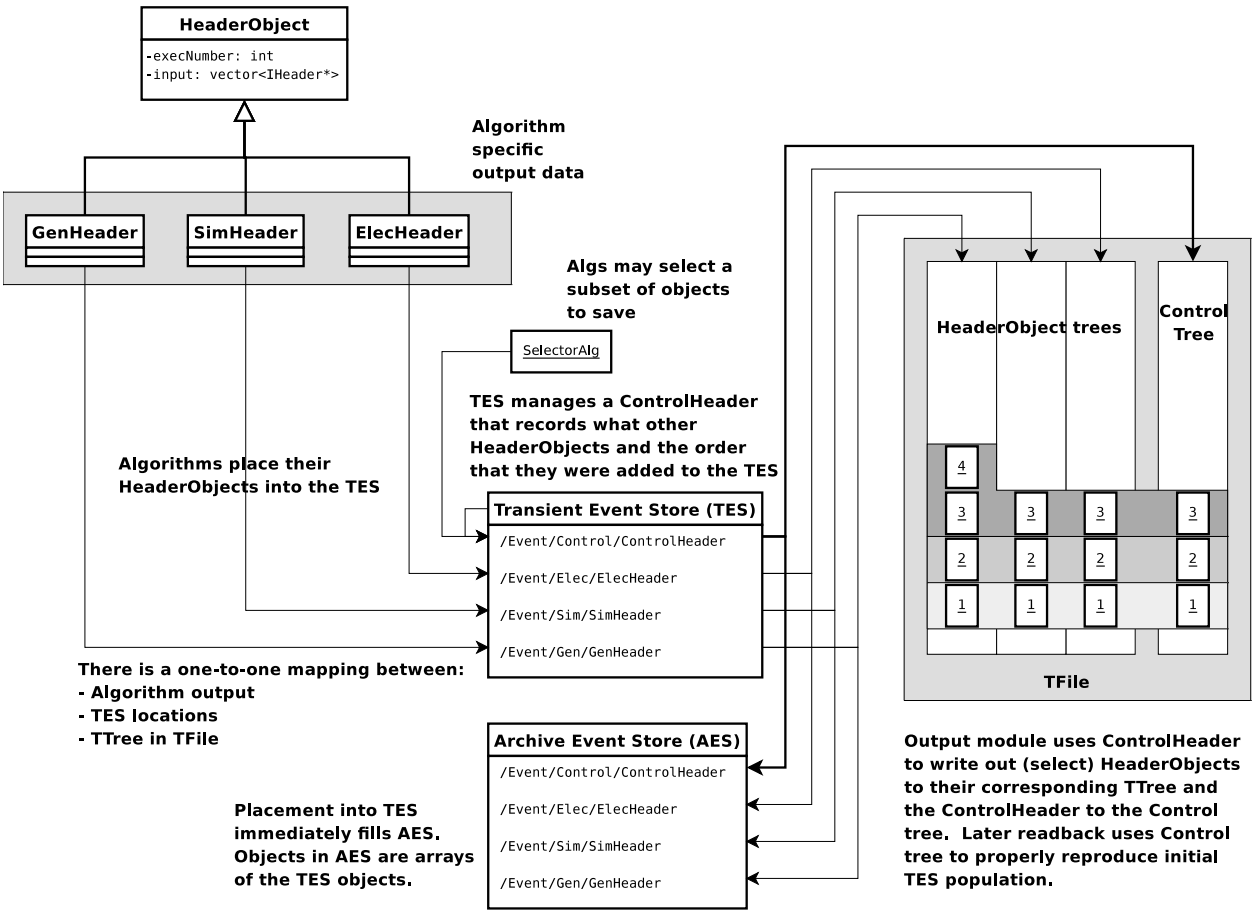

Figure 1: Cartoon showing the writing of HeaderObject data to file. See text for description.

The cartoon shows this data flow:

1. All algorithm specific data inherits from a common HeaderObject[4]. It provides various bookkeeping. Important here is the local execution number and the collection of other input HeaderObjects used.

2. Algorithms place their output objects into the TES (all placement here is by pointer) at a well defined location (path). Algorithms must not share the same TES location.

3. The TES places the object into the corresponding AES location. The AES locations contain ordered collections of all the objects that previously came from the same location in the TES and fit within the analysis time window.

[4]HeaderObject is a concrete base class which provides the IHeader interface.

4. The TES places the object into its current ControlHeader
5. Other algorithms may mark TES paths to be saved to file or not.
6. At the end of the top-level algorithm execution cycle, the OutputStream algorithm[5] will use the information in the ControlHeader to write out selected objects to their corresponding TTree in the output TFile.

5 Roadmap

The steps to implementation are:

1. Implement required changes to DataModel header objects (done).
2. Investigate automatically generated DataModel converters and if the correct ROOT dictionaries are produced by `GaudiObjectDesc`.
3. Fall back to manual converters if needed.
4. Modify `EventLoopManager` as required.
5. Implement `OutputStream` algorithm.
6. Implement event selectors to read-back in "normal" mode.
7. Implement SES for "simulation" mode read-back.
8. Modify staged processing algorithms to optionally take data from SES or generate new data like usual.

References

[1] "The Basic Event Selector Mechanism in Gaudi", S.Patton, DocDB-2157.

[2] "Daya Bay Transient Event Store", S.Patton DocDB-1965.

[3] "15 Minutes of Fame", D.Jaffe, B.Viren. DocDB-1532.

[4] "Daya Bay Offline Data Header Object", B.Viren, Z.Wang. DocDB-2119.

[5] To be developed, name may change.