Daya Bay I/O using RootIOSvc

Brett Viren

Physics Department

Daya Bay CN/US Offline, 2009/07/10

Overview

- Transient and Persistent Data Model
- Conversion
- Data Streams
- Output & Input Mechanisms
- Conversion Service and Event Selector
- Daya Bay specifics
- Going further

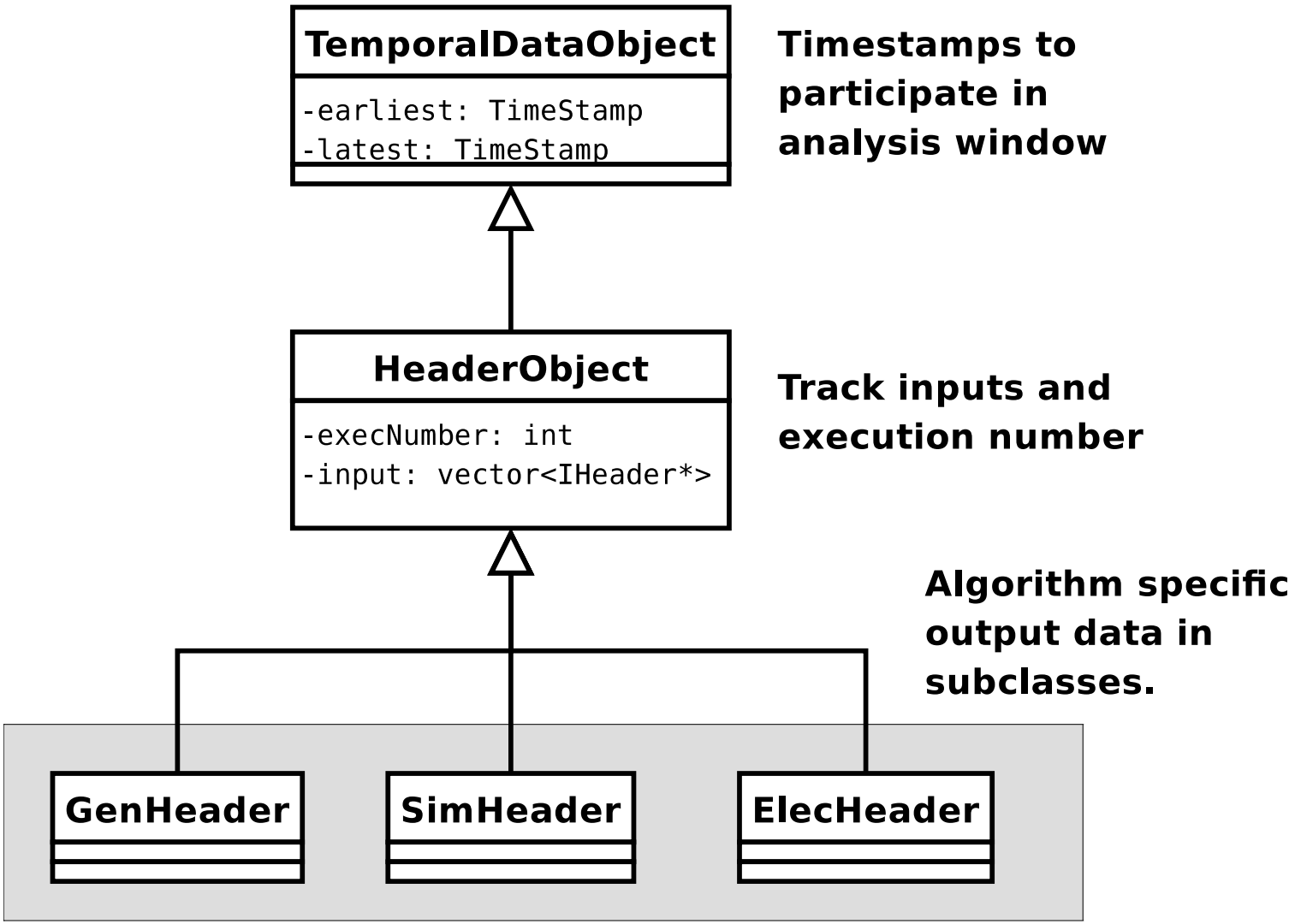
Software in `dybgaudi/RootIO/`

Main Package: `RootIOSvc`

Test Package: `RootIOTest`

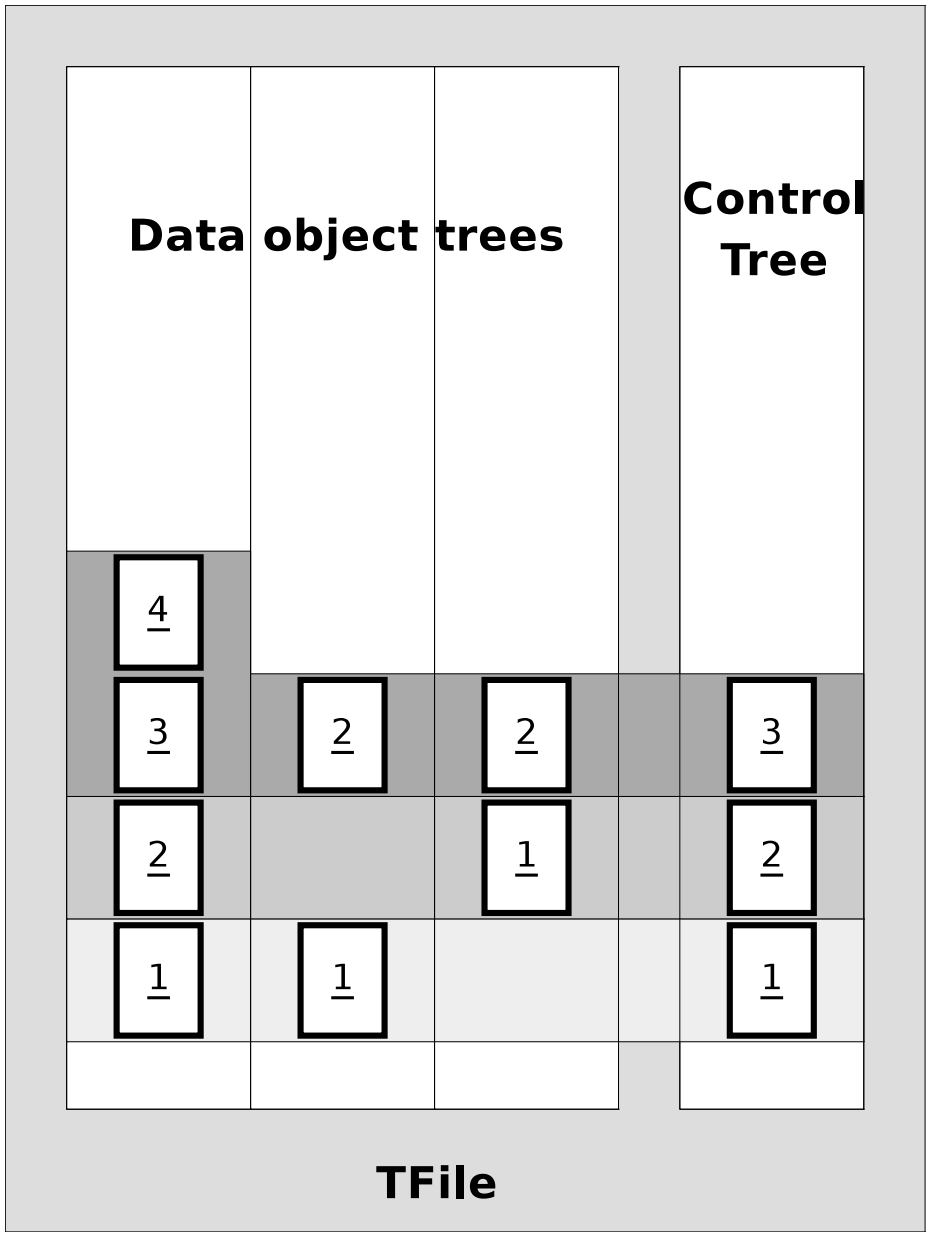
Data and Converters: `TBaseEvent` (corresponding to: `DataModel/BaseEvent`)

Transient Data Model: Base Objects

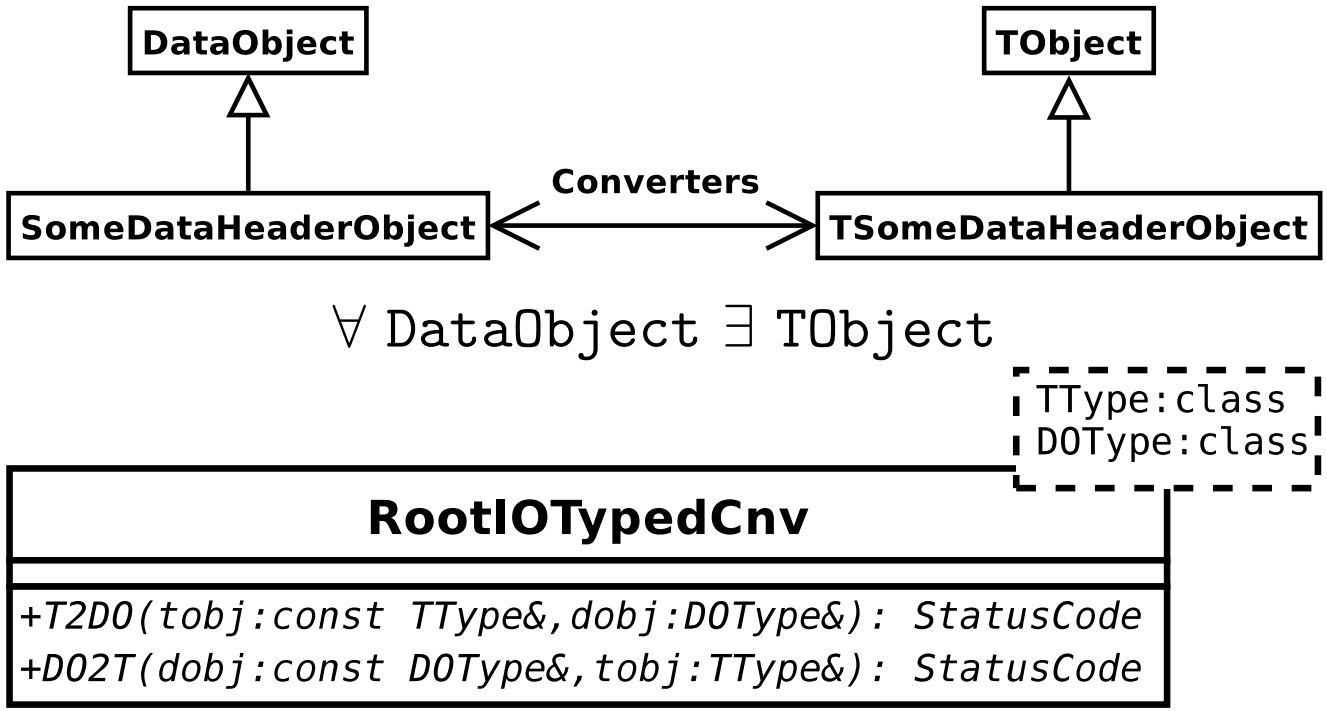


Persistent Data Model

- 1-to-1 correspondence between DataObject at a TES path and a ROOT TObject entry in a TTree in a TDirectory path
- Cycle may produce 0, 1 or more objects of a given type.
- Data produced in one execution cycle correlated with control tree (RegistrationSequence object).



Conversion



Converters subclass a templated base class. This base takes care of all Gaudi I/O business. Subclass only needs to copy data from one object to the other including any references to other data.

Data Streams

Streams:

- An abstract serial sequence of one type of data.
  - Hides all the crazy ROOT `T*` classes
- Streams are independent from one another.
  - I/O of many streams may come from / go to one or many files.
- Some file level control:
  - Output may be broken into different files (limit file size, implement sub-runs).
  - Input may be taken from multiple, sequential files (logical concatenation)

Stream Classes

RootOutputStream
+RootOutputStream(addr:void*,classname:string, treepath:string,branchname:string)
+newFile(filename:string): bool
+write(): bool
+close(): bool

RootInputStream
+RootInputStream(addr:void*,clid:int,treepath:string, branchname:string)
+append(filename:string): bool
+read(): bool
+setEntry(entry:int): bool

- Maintains address of pointer to object held to temporarily store an entry for `read()/write()`.
- Simple methods to provide necessary file-level information
- Input has navigation methods but `setEntry(int)` is main one. Entry number is global, not file-specific.
- Although we always use TObject, can support arbitrary object types (with corresponding dictionary)

Input/Output mechanism: overview

Follows usual Gaudi-way (in a nut-shell):
Output:

- "Something" takes a DataObject and uses the Conversion Service to convert it to persistent form.
- Conversion Service delegates conversion to a per-type converter

Input:

- Event loop manager says "next" and asks event selector to make "`/Event`"
- Event selector populates TES with addresses
- Algorithms access TES location and address are converted to an object.
- Conversion done with the conversion service which delegates to per-type converter.

Daya Bay specific mechanism

Want to save subset of all TES paths, possibly multiple objects per path and preserve ordering: `RegistrationSequence`[1] (RS).
Output:

- Given an RS, convert and save just those objects it lists
- Convert and save the RS.
- The "something" in previous slide is an implementation of `IDybStorageSvc`

Input:

- Read in and convert RS.
- Use RS to read in required data in proper order.
- A custom event selector is needed for this (more below)

This is still being worked on.

[1] S.Patton

The RootIOCnvSvc Conversion Service

- Manages a map from TES paths to their input and output streams.
- Configured with:

  InputStreamMap map from TES path to input filename.
  OutputStreamMap same for output
  DefaultInput default input file - all streams from file potentially read.
  DefaultOutput file to place any streams not registered in output map.
  EventDataService set TES or AES.

To do:

- Need to change InputStreamMap to allow for a list of filenames.
- Need to provide some way to break output streams into multiple files.
- Haven't yet tested with AES.

The `RootIOEvtSelector` Event Selector

- Responsible for populating event store and high level input navigation.
- Works with `RootIOCnvSvc` to access input streams.
- Maintains a high level entry number.
- By default policy will read in entries of that number and of all known streams.
- Subclass overrides `setEntry()` to provide other I/O policy
  - As will be done to use the `RegistrationSequence` organization.

Going further...

We now have a rudimentary I/O mechanism that has been tested with our basic DataModel classes. It handles all the ROOT and Gaudi I/O details and exposes a simple interface for Daya Bay specifics.

The goal is now to apply this to the simulation stages and for this we need:

1. Complete the TObjects and converters for the rest of the DataModel
   - Write by hand (they are small, but many) or modify GOD + DataModel XML files to generate object and converter classes.
   - Need to understand how to do inter-object referencing.
2. Write hooks to use RegistrationSequence, (can proceed parallel with the above).
3. Longer term: develop support for read-back of intermediate data in the "pull" processing mode.