ROOT I/O at CDF

US HENP ROOT Users Workshop
24 March 1999
Rob Kennedy

Main Topics

- ➪ **Object I/O - Streamer(), G__setup_memvarClass(), versions**
- ➪ **Event I/O - Trees, Branches, split-files and metadata**

Main Goals

- ➪ **Describe Our Current Experience, (Mis-)Understandings**
- ➪ **Seek Input on How Best to Interface ROOT I/O to CDF Offline**

CDF has officially decided to use ROOT to implement Event I/O, specifically in split-file, multi-branch mode with (almost) no dependence on Cint at run-time. We are implementing this now.

Background and Context

CDF Offline Software

- 100s of persistable classes, up to 100s of objects/event (Run 1)
- Objects passed between Framework/AC++ 'modules' via Event Record. I/O is performed by specialized 'I/O modules'.
- Much C++ already written, some F77 from Run 1 being re-used.
- Proposal in works to revise our 'Event Data Model' to permit to richer variety of C++ objects to be passed between modules and I/O.
- http://www-cdf.fnal.gov/upgrades/computing/projects/edm/edm.html

CDF Data Handling

- After lengthy review of competing proposals, CDF chose ROOT to implement Event I/O, specifically split-file, multi-branch mode to maximum event data throughput and resource utilization.
- One 'logical' event seen by the user. User selects which Branches to utilize for a particular task. Branches may be stored on separate media (disk, tape in robot, tape in vault, network). Unused Branches are not accessed (files not opened, tapes not mounted, etc.).
- http://www-cdf.fnal.gov/upgrades/cdfdh/welcome.html

Object I/O

➠ Goal: Decouple ROOT I/O from Offline objects and algorithms
➠ Goal: Minimize effort to maintain I/O methods like Streamer().

➠ Approach: Use I/O adapters for Offline objects. Do not derive Offline objects from TObject. Adapters contain the Streamer() methods, derived from or use TObject. Do not use Cint at run-time.

➠ Approach: Use rootcint to help create the Streamer() method for each particular version of an object class. Copy/paste Streamers for different versions together. Users must hand-code Streamer() methods in cases where member data is not yet supported by ROOT (string, STL, pointers), common for us. May require that each version of class be visible if ROOT I/O implementation changes.
--- OR ---
➠ Approach: Use an IDL-based approach. A generator has access to database of IDL descriptions by object class version and can generate entire Streamer(). Generator may require change if ROOT I/O implementation changes. More development work, but benefits.

Object I/O Experience

➠ One must supply at least part of the data dictionary to perform I/O with user-defined classes. Streamer() is not enough. Example:

```
static void G__setup_memvarCLASS(void) /* the "memvar" function */
{ G__tag_memvar_setup(G__get_linked_tagnum(&G__CLASS_dictLN_CLASS));
{ CLASS *p; p=(CLASS*)0x1000; if (p) { }
G__memvar_setup((void*)NULL,115,0,0,-1,
G__defined_typename("Version_t"),-1,2,"_bank_version=",0,(char*)NULL);
G__memvar_setup((void*)NULL,117,0,0,
G__get_linked_tagnum(&G__CLASS_dictLN_string),-1,-1,2,
"_bank_name=",0,(char*)NULL);
// MUCH snipped
   }
   G__tag_memvar_reset();
}
```

➠ Output by rootcint, even for class with hand-coded Streamer().

➠ Are there schema evolution issues with this memvar function?

➠ Using a pointer to Offline object in adapter class derived from TObject fails... fine detail why. Can this work w/ memvar change?

➠ Alternative to memvar()? TObject* TObject::clone() with covariant return types? Global Streamer/memvar combo in dict?

Event Organization and I/O

➠ Now: A Cdf Event is a collection of Ybos Banks maintained in an integer array. C++ and F77 interact with these Banks.
➠ Proposal: A Cdf Event is a list of smart pointers (handles) to C++ objects derived from a *small* base class Edm::StorableObject. A StorableBank will be supported for F77 interaction with Banks.
➠ Compatibility: Must be able to use existing non-ROOT datafiles.

➠ Input: objects in selected ROOT Branches will be read into Event... Branch structure no longer visible to user algorithms.

➠ Output: selected objects in Event will be output to ROOT Branches. ROOT Tree is created on the fly using object-Branch mapping. Either one class ... or a TCollection of classes... per Branch, with locally-supplied admin layer providing control of ‘coherent (in usage) group of classes’.

➠ Transition: Prototypes use single branch mode. When required admin layer ready, introduce multi-branch, single file mode. Finally, we will integrate split-file mode with Data Handling system.

Event I/O Concerns

➠ Our (mis-)understanding: In split-file mode, all files associated with all Branches must be opened, even those for inactive Branches.

➠ If true, this *defeats* our model for exploiting split-file mode for I/O efficiency. It would be unacceptable to us to have to open RawData Branches on archived tapes to look at micro-DST Branches on disk. Rene & Fons: Only files containing active Branches are opened.

➠ Our understanding: Once any one Branch's file reaches a maximum size, all files (split-file mode) are closed, regardless of size.

➠ This may interfere with our desire to open/close files on luminosity bookkeeping boundaries. Clearly, something must be done if file reaches max size. We will have to insure that never happens in the middle of an event or a luminosity bookkeeping unit

Event I/O Concerns

➠ Our expectation: Different analyses will have different definitions of ‘coherent groups of classes’.

➠ Are Trees/Branches flexible in split-file mode? Can users add Branches for their analyses after the basic structure has already been defined? The answers drive whether we have one class per Branch and more admin layer... or ... collections of classes per Branch and less admin layer.

➠ Split-file mode requires the use of Cint to access metadata describing at least the split-file structure of ‘logical’ file opened.

➠ Serious concern: Using Cint at run-time for CDF data acccess. Can there be a supported API for metadata? We envision several applications for this, including the avoidance of Cint with split-files.

And Last, but Not Least

➠ **We seek support for I/O of more Standard Library data types: STL containers, string, complex, ...**