

Using *Chunk* Classes in ROOT

A work in progress

Marc Paterno

FNAL/CD/CEPA

Goals & Concerns

- Many D0 collaborators want to use ROOT for analysis
- Several solutions considered
 1. store simple *structs* in the “standard tree”, unpack *chunks* into them, write code using them. Bad: loose OO advantage
 2. introduce base class, write *chunks* and “standard tree” classes to implement the common interface. Bad: maintenance nightmare
 3. store *chunk* instances in the “standard tree”. **Try a prototype of this!**
- Several are concerned with maintenance effort required
 - keeping development “in sync” between the D0reco environment and the ROOT environment seen as a potential problem
 - want to assure code developed for analysis of ROOT files is easily portable to the C++ framework/EDM environment
 - want to maintain as much ease-of-use from ROOT as possible

Requirements

- Very low impact on existing D0 code
 - backwards compatibility of data, subject to D0OM
 - no substantial modification of EDM interface
 - no impact on memory use or speed of D0Reco_x
 - no ROOT dependence introduced to D0Reco_x
- Crucial EDM features needed
 - support of existing chunks, with very limited modification
 - support of existing inter-chunk “pointer” (*LinkIndex* and *LinkPtr* templates of the D0 EDM)
- No conditional compilation; no separate ROOT and D0OM builds of the D0 code

What is already done

- Milestone 1: Able to write *chunks* to a Root file
- Milestone 2: Able to read the Root file and recover *chunks*... but not yet final form
 - with a specially built Root executable
 - linked with several D0 libraries
 - linked with specially-built Root dictionaries for select classes
 - user recovers *edm::ChunkWrapper<X>*, where *X* is the *chunk* in question
 - no ability to navigate from *chunk* to *chunk*

Recent progress

- Implementing prototype interface for use from compiled code
- Goals
 - interoperability with “framework” code
 - full access to *chunks*
 - full functioning of inter-*chunk* “pointers”
 - **ease of use!**

Ease of use!

- User should not have to:
 - manipulate *TClonesArrays*, *TBranches*, *TTrees*, ...
 - deal with casts, and and handle failures
 - be concerned with explicit lifetime management
- Error conditions should be diagnosed in a friendly fashion
- Handling multiple files, written by different programs, should be transparent

We can provide most of this, but not perfectly...

Prototype interface for *TreeReader*

```
typedef edm::THandle<MyChunk> handle_t;
typedef std::vector<handle_t> chunks_t;
typedef chunks_t::size_type size_type;
TreeReader tr(...); // several c'tors
chunks_t handles;
while ( tr.next() ) // loop over entries
{
    size_type nfound = tr.get(handles);
    if ( nfound != 0 ) handles[0]->doWork();
}
// also provide random access
if ( tr.get(handles, 3) ) { ... }
```

But there's another requirement...

It was only lately appreciated that an important requirement is not yet met

- Users want easy use of *TTree::Draw(...)* to make histograms of qualities of tracks, muons, jets, *etc.*
- *TTree::Draw(...)* provides the ability to call “simple” functions in the interface *of the class that is stored in a branch*
- The “things” stored on the branches contain *collections* of tracks, muons, *etc.*
 - *TTree::Draw(...)* provides no way to iterate

Status

- Another meeting to further define requirements and design choices is being scheduled
- There seem to be several paths along which progress can be made, perhaps in combination
 - enhance the “language” *TTree::Draw(...)* understands
 - modify the structure of what is stored in the *branches* to support this use *as well as* the currently-supported use
 - provide “helper objects” that know about *chunks* and can provide the right functionality for the ROOT prompt