Analysis Environments for CMS

Christopher D Jones
Cornell University
Luca Lista
INFN
Benedikt Hegner
DESY

On behalf of the CMS Offline and Computing Team

Overview

Goals

Support multiple analysis strategies so physicists can pick the one to which they are most comfortable

Entice physicists to use CMS standard data files rather than creating their own NTuple formats

Data Model

Needs to support direct ROOT access

Strategies

Full Framework

Used for grid and local work

TFWLiteSelector

used with ROOT and PROOF for long running local and LAN work

FWLite

used with ROOT and full object dictionaries for local short-duration work

'Bare' ROOT

used for simple interactive activities

Data Model

Have events which contain data products

list of tracks

Data are constructed by a module while running in a process

the track finder module creates the list of tracks while in the reconstruction process

Data products are identified by

C++ class type: provides type-safety

std::vector<Track>

module label: a string which was assigned to the module which constructed the object

"trackFinder"

product instance label: if a module inserts multiple objects of the same type into the event, the products are differentiated via this string

"" or "failFit"

process name: a string assigned to the process being run. Keeps data from different processing steps (e.g., HLT and RECO) from interfering.

"Reco"

Must use edm::Ref<> when referring to data between products

A smart pointer which knows how to find data in the Event

ROOT File representation

Persistent and transient representations are identical

- Can read objects in ROOT using standard dictionaries

Events are entries in one TTree

Data products are individual TBranches of the Events TTree

- branch names are formed by concatenating a simplified class name with the three strings

  “Tracks_trackFinder__Reco”
  “Tracks_trackFinder_failFit_Reco”

The TBranches hold an edm::Wrapper<> instance

- holds the actual data which was inserted into the Event
- contains status information about object (e.g., is present)
- inherits from the base class EDProduct which gives a virtual destructor

Use Reflex to auto-generate the dictionaries

Use full splitting of objects to improve compression and readback

Framework adds full provenance information into files

Full Framework

Must be used when running on the grid

Works fine locally as well

Event loop is handled implicitly by the framework

Data access code lives inside a module

Call functions of edm::Event to get data

finds the most recent process with this label

```
void MyModule::analyze(const edm::Event& iEvent, ..) {
   edm::Handle<std::vector<Track> > tracks;
   iEvent.getByLabel("trackFinder",tracks);
```

specify all the labels explicitly

```
void MyModule::analyze(const edm::Event& iEvent,..) {
   edm::Handle<std::vector<Track> > tracks;
   iEvent.getByLabel("trackFinder", "", "Reco", tracks);
```

use a selection functor to find a match

```
void MyModule::analyze(const edm::Event& iEvent,..) {
   std::vector<edm::Handle<std::vector<Track> > > trackH;
   iEvent.getMany(MySelector(), trackH);
```

TFWLiteSelector

Run in ROOT or PROOF using the TSelector system

- TFWLiteSelector inherits from TSelector

Useful for medium length jobs run locally or on LAN

Event loop is handled implicitly by ROOT

Data access uses edm::Event

- Allows direct reuse of what physicists learned when using full framework
- Simplified interface to make it obvious which parts are run on client vs servers when using PROOF
  - client and server part communicate by explicitly passing TList’s
  - a new worker is instantiated for each server

The client side part

```
class MySelector :public TFWLiteSelector<MyWorker> {
  void begin(TList*&) {}
  void terminate(TList& out) {out.FindObject(“a”)->Draw();} };
```

The server side part

```
class MyWorker {
  MyWorker(TList& out): h_a(new TH1F(“a”,...) ){out.Add(h_a); }

  void process( const edm::Event& iEvent ) {
     edm::Handle<std::vector<Track> > tracks;
     iEvent.getByLabel(“trackFinder”,tracks);
     h_a->Fill(tracks.size()); }
  void postProcess(TList&) {} };
```

FWLite

Run in ROOT using object dictionaries and helper classes

Useful for interactive analysis running on one machine

- Support CINT or compiled code
- Supports use of CMS' intra-Event smart pointer (edm::Ref<>)
- Provide automatic loading of proper dynamic libraries via CMS' plugin system

Physicists explicitly control the event loop

Multiple mechanisms supported

- dictionaries only
- fwlite::Event
- Python via PyROOT

Dictionaries Only

Framework

```
edm::Handle<std::vector<Track> > tracks;
iEvent.getByLabel(“trackFinder”,tracks);
```

Dictionaries Only

```
TFile f(“...”);
TTree* ev = (TTree*) f.Get(“Events”)
ev.GetEntry()

std::vector<reco::Track> tracks;
TBranch* bTracks = ev->GetBranch(
 “Tracks_trackFinder__Reco.obj”);
bTracks->SetAddress(&tracks);

for(int index=0; ...) {
   bTracks->GetEntry(index);
   ev->GetEntry(index,0); //Needed for edm::Ref
```

Difficulties

Must know how to interpret the branch names

How do you know what C++ class goes with which branch?

If there are multiple branches which only differ by the last word (Process name) how do you know which one is the latest?

How do you know exactly which data member held by the branch should be read?

Setting the branch address is error prone

Can pass the wrong object type to the branch

Must call TTree::GetEntry before calling TBranch::SetAddress

When do you pass to TBranch::SetAddress a pointer and when do you pass a pointer to a pointer?

Getting the data can be error prone

Must not forget to call TBranch::GetEntry before trying to read the data

How do you know if the data object was not put into the edm::Event at all?

Dealing with edm::Refs is difficult

Must call TTree::GetEntry for each event before ever asking for an edm::Ref (which might happen internally to a member function)

To cover all cases, the access is not super fast

The access can fail under certain conditions because there are insufficient 'hooks' in ROOT

fwlite::Event

Framework

```
edm::Handle<std::vector<reco::Track> > tracks;
iEvent.getByLabel(“trackFinder”,tracks);
```

Using the helper fwlite::Event

```
#include “DataFormats/FWLite/interface/Handle.h”
TFile f(“...”);
fwlite::Event ev(&f);
for( ev.toBegin(); !ev.atEnd(); ++ev) {
  fwlite::Handle<std::vector<Track> > tracks;
  tracks.getByLabel(ev,“trackFinder”);
```

Improvements

Do not have to know about meaning of branch names

- Can give ‘module label’, ‘product instance label’ and optionally ‘process name’ just like in the Framework
- Can even deal with the case when ‘simplified class name’ changes

Branch memory handling is done for you

- Results are cached for fast results
- Not possible to assign wrong C++ class object to the branch

Getting data is safe

- Data are only retrieved the first time the ‘getByLabel’ is called for that object for that event
- Will throw exceptions if have problems

edm::Refs work under all conditions

- Since code has control of data access it can safely and quickly setup the edm::Ref’s
- edm::Ref access will be faster

Python

Framework

```
edm::Handle<std::vector<Track> > tracks;
iEvent.getByLabel("trackFinder",tracks);
```

Python

```
import PhysicsTools.PythonAnalysis as pa
events = pa.EventTree("...root")
for e in events:
  tracks = e.Tracks_trackFinder__Reco()
```

Improvements

Similar to fwlite::Event except must specify branch name

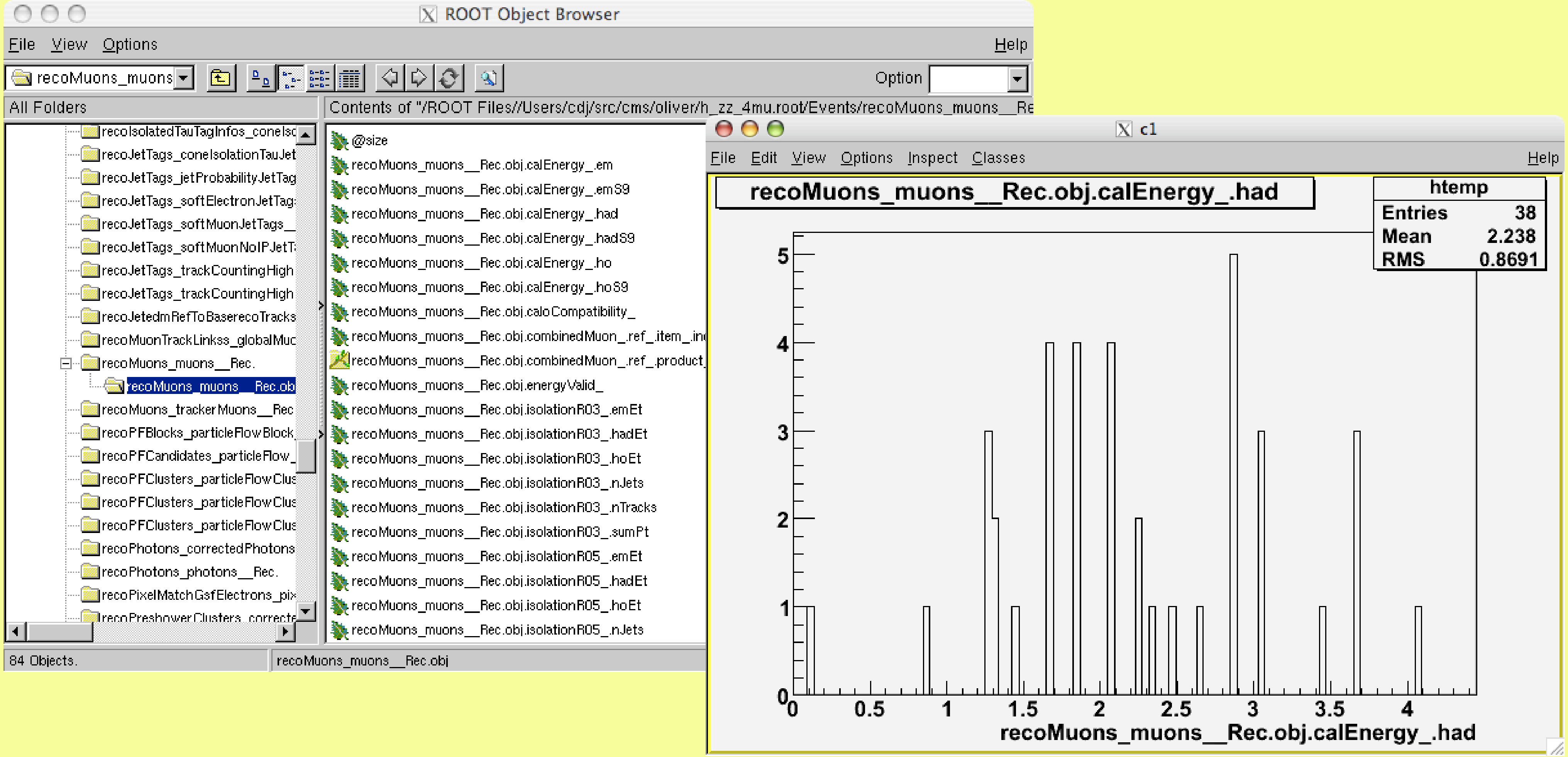
However C++ type is deduced automatically from the TBranch's contents

Bare ROOT

TBrowser and TTree::Draw can operate without dictionary support

Used for quick interactive analysis

- Allows fast checks on simple quantities
- With dictionaries TBrowser and TTree::Draw can also access methods of objects



Contents of "/ROOT Files//Users/cdj/src/cms/oliver/h_zz_4mu.root/Events/recoMuons_muons__Re

```
recoIsolatedTauTagInfos_coneIsc
recoJetTags_coneIsolationTauJet
recoJetTags_jetProbabilityJetTag
recoJetTags_softElectronJetTag
recoJetTags_softMuonJetTags_
recoJetTags_softMuonNoIPJetT
recoJetTags_trackCountingHigh
recoJetTags_trackCountingHigh
recoJetedmRefToBaserecoTracks
recoMuonTrackLinkss_globalMuc
recoMuons_muons__Rec.
recoMuons_muons__Rec.ob
recoMuons_trackerMuons__Rec
recoPFBlocks_particleFlowBlock
recoPFCandidates_particleFlow_
recoPFClusters_particleFlowClus
recoPFClusters_particleFlowClus
recoPFClusters_particleFlowClus
recoPhotons_correctedPhotons
recoPhotons_photons__Rec.
recoPixelMatchGsfElectrons_pix
recoPreshowerClusters_correcte
```

```
@size
recoMuons_muons__Rec.obj.calEnergy_.em
recoMuons_muons__Rec.obj.calEnergy_.emS9
recoMuons_muons__Rec.obj.calEnergy_.had
recoMuons_muons__Rec.obj.calEnergy_.hadS9
recoMuons_muons__Rec.obj.calEnergy_.ho
recoMuons_muons__Rec.obj.calEnergy_.hoS9
recoMuons_muons__Rec.obj.caloCompatibility_
recoMuons_muons__Rec.obj.combinedMuon_.ref_.item_.in
recoMuons_muons__Rec.obj.combinedMuon_.ref_.product
recoMuons_muons__Rec.obj.energyValid_
recoMuons_muons__Rec.obj.isolationR03_.emEt
recoMuons_muons__Rec.obj.isolationR03_.hadEt
recoMuons_muons__Rec.obj.isolationR03_.hoEt
recoMuons_muons__Rec.obj.isolationR03_.nJets
recoMuons_muons__Rec.obj.isolationR03_.nTracks
recoMuons_muons__Rec.obj.isolationR03_.sumPt
recoMuons_muons__Rec.obj.isolationR05_.emEt
recoMuons_muons__Rec.obj.isolationR05_.hadEt
recoMuons_muons__Rec.obj.isolationR05_.hoEt
recoMuons_muons__Rec.obj.isolationR05_.nJets
```

84 Objects. | recoMuons_muons__Rec.obj

recoMuons_muons__Rec.obj.calEnergy_.had

htemp	
Entries	38
Mean	2.238
RMS	0.8691

Data Model Challenges

Schema evolution

ROOT provides some basic schema evolution ability for split objects

- addition or removal of a member data
- change of a member data from one primitive type to another primitive type

Additional abilities would be useful

- apply a transform to an old member data
  - *e.g., change an internal value previously stored in 'mm' to 'cm'*
- apply a transformation to several old member data
  - *e.g., previously stored x,y,z but now are using r, theta, phi*
- change a data member from one class to an equivalent class
  - *e.g., previously used a CLHEP vector as a member data but was later changed to be a ROOT's vector class*

Discussing with ROOT team how to handle these other types of changes

Caching and object reuse

TTree::Draw and most macro examples reuse object memory without calling constructors/dtrs

We do not want to store data members which are caches

- *E.g., the value of phi for a jet were the internal representation is x,y,z*

Problem: When calling a method which access the cache, you always get back the value for the cache which was set by the first object to use that memory location

Solution: Create a special cache class

- All internals are transient
- Defines its own TStreamer so on readback the cache can be reset

Conclusion

CMS's framework file format is directly useable in ROOT

Physicists have been using the format and giving feedback

- Most feedback for FWLite strategies
- Data challenge this Fall will be producing physics skims only in this format

Creating a compatible Data Model remains a challenge

Ultimately physicists will vote with their feet

- Constant interaction with physicists is the only way to insure success