Introduction to ART and ART modules

NOνA software tutorials – Fermilab

Christopher Backhouse

California Institute of Technology

April 2, 2014

Preliminaries

```
setupnova                              # Use development
newrel -t development dev_tute         # Make a test release
cd dev_tute
srt_setup -a                           # Tell SRT to look here
addpkg_svn -h Demo                     # Add development Demo
cd Demo
make                                   # Build the code
```

Modules

- Data is split into Runs, split into SubRuns, split into "Events"
- Very overloaded word, this type also called spills, snarls, triggers
- The 500μs window where the detector is taking data

- ART processes data event-by-event
- Each event is run through a sequence of "Modules"
- Each module does something to the event, normally add one or more "Products"
- Once a product is in the event it's immutable
- Modules can access anything previous modules have put into the Event
- The whole thing is controlled by a "fcl" file (or "macro" or "script")

TutFilter

- `$SRT_PUBLIC_CONTEXT/Demo/TutFilter_module.cc`
- `nusoft.fnal.gov/nova/novasoft/doxygen/html/classtut_1_1TutFilter.html`

- Class, within a namespace, inheriting from `art::EDFilter`
- Filters are relatively rare, but simplest to start with

- Defines various standard *virtual* functions
  - Constructor
  - `filter()`
  - `reconfigure()`
  - `beginJob()`
- Others are available, not used here

- Registers with ART via `DEFINE_ART_MODULE(TutFilter);`

TutFilter

- Constructor
- Is passed configuration as a `ParameterSet`

```
fGeantLabel = pset.get<std::string>("GeantLabel");
```

- We'll see where "GeantLabel" came from in a minute

- `beginJob()`
- Called once, after constructor, before any `Events`

- `filter()` function
- Gets called once for each ART `Event`

TutFilter::filter()

- Passed the current `art::Event`
- Can retrieve products that have previously been put in the event
- Not used here

- Accesses a service[1] via a `ServiceHandle`

  ```
  art::ServiceHandle<cheat::BackTracker> bt;
  ```

- See Brian's talk for BackTracker details
- Here we determine if any primary particle is a π^0 decaying $\rightarrow \gamma\gamma$

- An `art::EDFilter` decides if this `Event` proceeds to later modules
- Return `true` to keep, `false` to discard

[1] Think "singleton"

TutFilter.fcl

- Boilerplate defining the module
- Plus configuration parameter names and default values

tutfiltjob.fcl

- Describes how to run the job
- `@local::standard_services` brings in a block defined elsewhere
- Array of standard services, BackTracker is not one of them
- `services.user.BackTracker: @local::standard_backtracker`

- Get input from an ART .root file. Output to tut.root
- `SelectEvents: { SelectEvents: [``tutfilt''] }`

  magic incantation to get Filter results to apply to output file, not just what modules are run in the path

- Declare what modules we're using, give them names ("filt")
- Arrange modules into paths ("tutfilt")
- Specify what paths to run (`trigger_paths`)

Running the job

- `nova -c tutfiltjob.fcl -n20 /nova/data/mc/S13-06-18/genie/fd/fd_r0000001_s00_*_fhc_fluxswap*.sim.pid.root`

Inspecting the output

- How was it run? `config_dumper tut.root`
- What's in it? `nova -c eventdump.fcl tut.root`
- Low-level: `root tut.root`, `Events->GetEntries()`, `TBrowser`
- Understanding what most of this stuff is, see Evan's talk
- Event Display: see Michael's talk

- Confirm that we really did filter stuff out

TutProducer_module

- Now to actually do something with those π^0s
- Producer nearly the same as a Filter
  - Inherit from `EDProducer`
  - Have a `produce()` method

- Declare what you put in the Event in your constructor

```
produces<std::vector<rb::Prong>>();
```

- Boilerplate for something you're going to put into the Event

```
std::unique_ptr<std::vector<rb::Prong>> prongcol(new
std::vector<rb::Prong>);
```

- At the end, actually store it

```
evt.put(std::move(prongcol));
```

TutProducer_module

- Getting products to work with, more boilerplate

```
art::Handle<std::vector<rb::CellHit>> chits;
evt.getByLabel(fCellHitLabel, chits);
```

- `fCellHitLabel` here is "calhit", standard name, but can find with eventdump or config_dumper or TBrowser

Handles and Ptrs and PtrVectors

- `art::Handle` acts like a pointer, derefence with *
- Bonus feature, index in to get an `art::Ptr` like

```
art::Ptr<rb::CellHit> chit(chits, chitIdx);
```

- `Ptr` is just another type of reference into the Event
- `PtrVector` is very similar to `vector<Ptr>`, but all members must be from the same product

tutprodjob.fcl

- Just like tutfiltjob.fcl but we run more than one module in sequence
- TutProducer only tries to fit Prongs in Events TutFilter allows

- `nova -c tutprodjob.fcl /nova/data/mc/S13-06-18/genie/fd/fd_r0000001_s00_*_fhc_fluxswap*.sim.pid.root`

TutAnalyzer_module

- Analyzer the same again, but doesn't put anything in the Event
- Inherit from `EDAnalyzer`, implement `analyze()` function

- Putting histograms etc in the histogram file

```
art::ServiceHandle<art::TFileService> tfs;
fMassPeak = tfs->make<TH1F>(``massPeak'',
``;Reconstructed mass (MeV)'', 125, 0, 250);
```

- Have to wait until `beginJob()`[2]

- Get our prongs from the previous module with `getByLabel`
- Evaluate invariant mass $E = \sqrt{2E_aE_b(1-\cos\theta)}$ and fill histogram

[2] If you depend on Geometry use `beginRun()`, make sure you don't create histos twice

tutanajob.fcl

- This time, taking in our `tut.root` as input
- Definition of where `FileService` output goes

  `TFileService:{fileName:‘‘tut_hist.root’’ closeFileFast:false}`
- No ART output stream this time
- Path goes in `end_paths`, not `trigger_paths`, because it doesn't write to event
- Could also have combined this all (filt, prod, ana) in one job
- `nova -c tutanajob.fcl tut.root`

Tips

- Doxygen is your friend

  `http://nusoft.fnal.gov/nova/novasoft/doxygen/html/`

- Find examples in modules that already do something similar
  - I never start a module totally from scratch
  - I still often copy-paste syntax from somewhere
- grep/ack are your friends
- If you know one word related to what you're doing, you can probably find an example

- gdb/valgrind can be your friends, but often a ton of `cout` statements do the trick ("did we even get here?", "how many X's do I have?")

- You should be on `nova_offline` and the `novasvncommit`

Other resources

- Artists' tutorial

  `oink.fnal.gov/new_tut/tutorial.html`

- Our tutorial

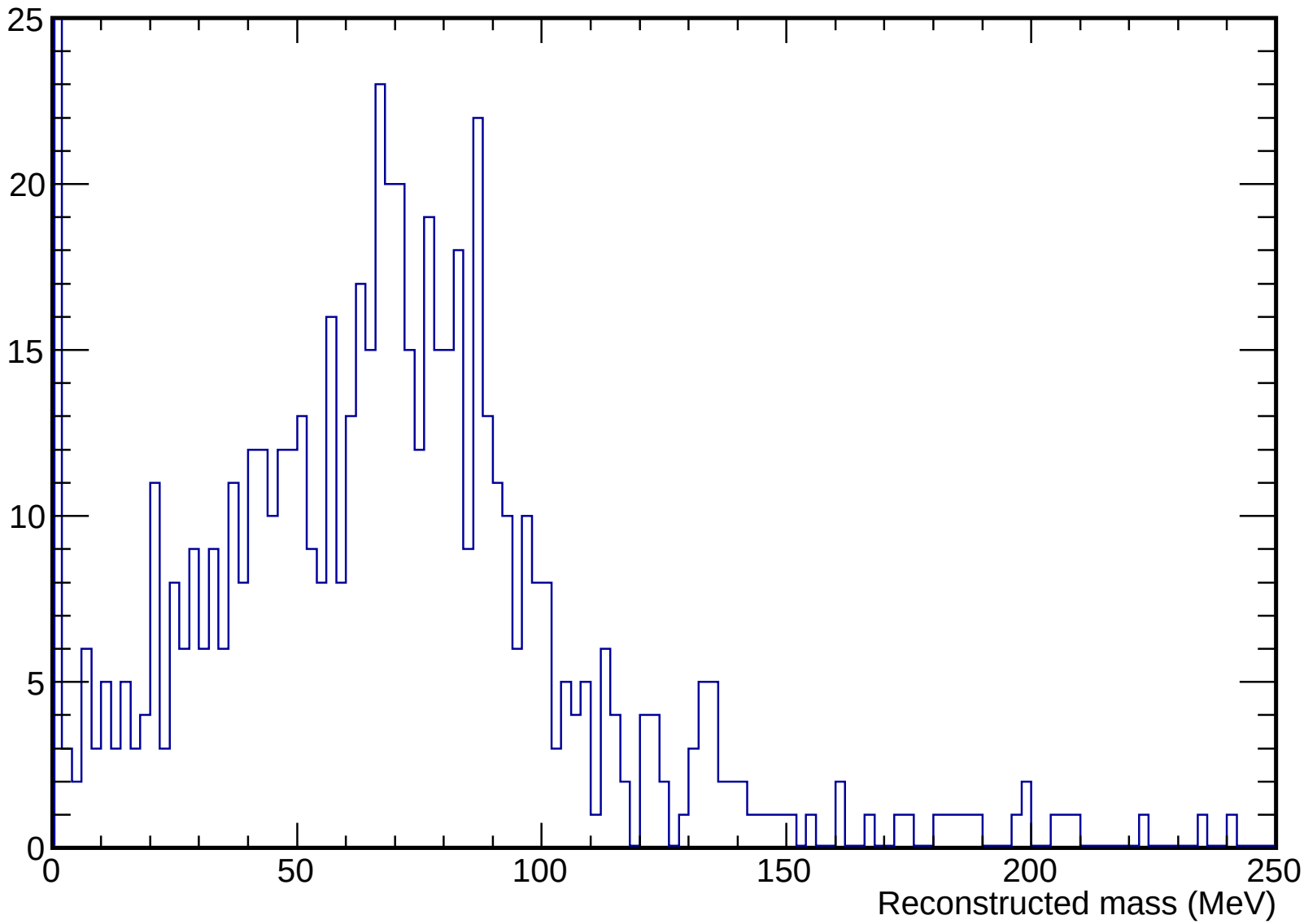
  `cdcvs.fnal.gov/redmine/projects/novaart/wiki/Using_the_Framework`

- NOνA FAQ

  `cdcvs.fnal.gov/redmine/projects/novaart/wiki/Trouble_Shooting_and_Gotchas`

Backup

TutAna result



Sequence of functions

- Constructor
- `beginJob()`
- `beginRun()`
- `beginSubRun()`
- `produce()/filter()/analyze()`
- `endSubRun()`
- `endRun()`
- `endJob()`

Dictionaries

- How does ART know how to write out/read in objects?
- Various packages have dictionary files `classes_def.xml`, `classes.h`
- Fairly simple example in `SummaryData`

MessageFacility

- Yes, this is a thing
- Look for users of `mf::LogInfo`, `mf::LogWarning` etc.

Assns

- Yes, there are a thing
- Form an "association" between two products
- Later module can find one from the other via the `Assn`
- Syntax is a bit tricky. Find a module that does what you want and copy

`cdcvs.fnal.gov/redmine/projects/novaart/wiki/Using_Associations`