Data Products and Reconstruction in NOvA

Evan Niner

4/02/14

Provide an overview of basic use and what reconstruction programs/objects exist. Follow all the links for additional wikis, source code, tech notes, and presentations.

What are data products in NOvA?

- NOvA uses the ART framework (https://sharepoint.fnal.gov/project/ArtDoc-Pub/SitePages/Home.aspx) on .root files
- Each Producer module (see talk by C. Backhouse for details on ART and running jobs) must add an output product to the file
- Data products in ART are persistent and are not typically removed from a file*. So the ART file maintains a record of each step of processing along the way.

*It is possible to setup a job to drop specified data products to reduce file size. This is an advanced topic. Ask me later if interested. An example job that uses this is here.

Let's look at data products in a file

- On a NOvA virtual machine setup the nova environment
- nova -c eventdump.fcl /nova/prod/mc/S14-03-06/reco/validation/fardet_genie_fhc_swap_3000_r01000007_s61_S14-03-06_v2_20140213_171334_hero5005.rc.fas.harvard.edu_1392893629_4316_0.sim.daq.reco.root
- This is a recently produced file that contains all reconstruction objects from the official file production currently in progress.
- List of products on next slide (I removed associations from the list, will discuss them later).

1.	2.	3.	4.	5.
PROCESS NAME..	MODULE LABEL.....	PRODUCT INSTANCE NAME	DATA PRODUCT TYPE.....	.SIZE
test.....	generator.....	.....	sumdata::SpillData.....	....-
BatchRecoValid	TriggerResults....	.....	art::TriggerResults.....	....-
test.....	rns.....	.....	std::vector<art::RNGsnapshot>.....	3
BatchRecoValid	elasticarmshs.....	.....	std::vector<rb::Vertex>.....	1
BatchRecoValid	multihough.....	.....	std::vector<rb::HoughResult>.....	2
BatchRecoValid	kalmantrackmerge..	.....	std::vector<rb::Track>.....	...19
test.....	daq.....	.....	std::vector<rawdata::RawTrigger>.....	1
BatchRecoValid	slicer.....	.....	std::vector<rb::Cluster>.....	2
BatchRecoValid	cosmictrack.....	.....	std::vector<rb::Track>.....	1
BatchRecoValid	kalmantrack.....	.....	std::vector<rb::Track>.....	...28
BatchRecoValid	ifdbspillinfo.....	.....	sumdata::SpillData.....	....?
BatchRecoValid	fuzzykvertex.....	Prongs3D.....	std::vector<rb::Prong>.....	5
test.....	daq.....	.....	std::vector<rawdata::RawDigit>.....	20560
test.....	generator.....	.....	std::vector<simb::MCFlux>.....	1
BatchRecoValid	fuzzykvertex.....	Prongs2D.....	std::vector<rb::Prong>.....	8
test.....	photrans.....	.....	std::vector<sim::PhotonSignal>.....	.9028
BatchRecoValid	slicemef.....	.....	std::vector<me::MichelECluster>.....	4
BatchRecoValid	veto.....	.....	rb::FilterList<rb::Cluster>.....	....-
test.....	geantgen.....	.....	std::vector<sim::Particle>.....	..293
test.....	geantgen.....	.....	std::vector<sim::FLSHitList>.....	1
BatchRecoValid	elasticarmshs.....	.....	std::vector<rb::Prong>.....	3
test.....	generator.....	.....	std::vector<simb::MCTruth>.....	1
BatchRecoValid	michelecosmictrack	.....	std::vector<me::MichelECluster>.....	0
BatchRecoValid	michelekalmantrack	.....	std::vector<me::MichelECluster>.....	0
test.....	generator.....	.....	std::vector<simb::GTruth>.....	1

What does this mean?

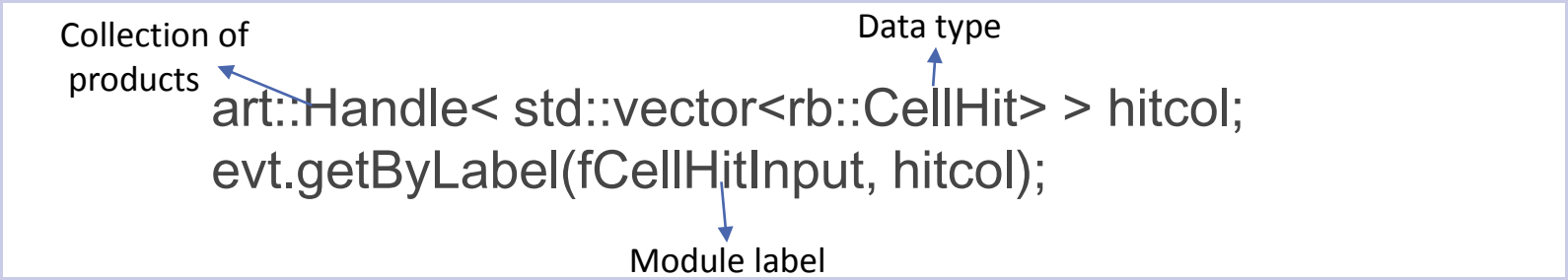
1. Process Name: Specified in the "process_name" field of each fcl job file. Helps identify what job was associated with those products. NOTE: Each job you run on a given file must have a unique process name.
2. Module Label: In the fcl job file, the identifying label of a particular module being run (different from its name). This name can be anything , but there are conventions for production modules. See wiki for more: https://cdcvs.fnal.gov/redmine/projects/novaart/wiki/Documentation_FOR_BEGINNERS#Module-Labels
3. Instance Name: If you want one module to produce multiple products of the same type (ex: separate output for 2d and 3d tracks) then an instance name is needed in addition to the module name.
4. Data Product Type: What was the product?
5. Size: Number of products stored of that type in the event.

Notes

- The combination of process name, module name, instance, and data type is unique.
- If you run a module (ex. Slicer) twice under two different process names but the same module name, you will only be able to access the most recent output later on.
- If you want to compare the output of one module under two configurations in the same file, you need 2 different module names.

Retrieving data products: Get by label

- Retrieve collection of data products from an event associated with a module label
- Example from line 144-145 https://cdcvs.fnal.gov/redmine/projects/novaart/repository/entry/trunk/Slicer/Slicer4D_module.cc



- NOvA convention is to define module labels as fcl parameters that way they can be changed without recompiling code
- If there is an instance label the format is: evt.getByLabel(moduleLabel,instanceLabel,artHandle)
- https://cdcvs.fnal.gov/redmine/projects/novaart/wiki/Using_the_Framework#artHandle

Retrieving data products: Get by association

- https://cdcvs.fnal.gov/redmine/projects/novaart/wiki/Using_Associations
- https://cdcvs.fnal.gov/redmine/projects/novaart/wiki/Using_the_Framework#artAssns
- Associations talk by B.Rebel: docdb-7747
- Ask for a set of objects in an event associated with another object.
  - Ex: Get collection of track objects in an event that came from a specific slice

Get collection of slices

Find tracks associated with the slice

```
art::Handle<std::vector<rb::Cluster> > slice;
evt.getByLabel(fSliceLabel,slice);
art::FindManyP<rb::Track> fmTrack(slice, evt, fTrackLabel);
for (i=0; i<slice->size(); ++i){
 std::vector<art::Ptr<rb::Track> > tracks = fmTrack.at(i);
}
```

Get tracks for a specific slice

- For more on how to produce associations read the links and then ask me questions.
- Most NOvA code accesses data products in this way.

Where to find data products

- Data product class definitions exist in different parts of the NOvA code, convention is to group similar things together
- Key packages containing data products:
  - RecoBase: Holds class definitions for base reconstruction objects
  - RecoInt: Defines class for neutrino interactions
  - CalibrationDataProducts: Light weight reco objects used in calibration analysis
  - RawData: Raw daq objects
  - Simulation: Defines particle FLS hit. MCNeutrino defined in the base class
  - SummaryData: POT info and run summary objects
  - StandardRecord: CAF class definitions
- When making new products try to keep them in central locations.

Defining data products

- Every package defining data products must contain two files
  - classes.h: Define templates for data product classes
  - classes_def.xml: Declare class names, and most importantly versions
- If you change the data members in a data product class you MUST increase the class version number for backwards compatibility

```
<class name="rb::CellHit"   ClassVersion="13" />
<class name="rb::Cluster"  ClassVersion="13" />
```

- More advanced information on evolving data products to ensure backwards compatibility can be found here.

RecoBase data products

- Summary of design philosophy for the RecoBase package can be found in docdb-6119
- rb::CellHit: Inherits from rawdata::RawDigit , adds calibration independent charge (PE) and time (ns) fields
- rb::RecoHit: Inherits from CellHit. Calibration has been applied to produce attenuation corrected charge (PECorr) and reconstructed hit time. This object requires and estimate of the hit position in the opposite view.
- rb::WeightedHit: A structure containing a CellHit with a weight associated. This weight is useful if one hit is being shared by several clusters
- rb::HitMap: Provides efficient way to store/look-up hits by plane and cell. A tool used in several algorithms
- rb::Energy: Base class with only one data member, Energy. Energy estimator classes should inherit from this and add algorithm specific information
- rb::PID: Base class holding PID value. PID algorithms define data products that inherit from this with additional information.

More RecoBase data products

- rb::Cluster: A collection of CellHits. Hits can be added/subtracted/extracted. Average position/energy can be calculated, this is the basic reconstruction object. Can be 2D or 3D.
- rb::Prong: Inherits from Cluster. This object adds a start point and direction. 2D or 3D.
- rb::Track: Inherits from Prong. Defines and endpoint and intermediate trajectory points/path length. This is a refined final track. Can be 2D or 3D.
- rb::Shower: Inherits from Prong. Defines length and endpoint but no trajectory. Other shower objects inherit from this. 2D or 3D.
- rb::Vertex: Defines X,Y,Z,T for an interaction point. Clusters,prongs or tracks can be attached to this vertex and there are functions to retrieve them.
- rb::HoughResult: Collection of hough lines with information on rho,theta and strength of the hough line.

CalHit (CalHit_module.cc)

- Primary contact: Chris Backhouse, Kanika Sachdev
- Relevant documents: n/a
- S14-03-06 production performance: docdb-10909
- Input: daq raw digits (rawdata::RawDigit)
- Output: cell hits (rb::CellHit)
- Description: This module takes the raw digits produced in the DAQ and turns them into cell hits containing a fitted ADC value and Time. This module is required for all other offline reconstruction.
- Standard module label: calhit
- In Production? YES

Slicer (Slicer4D_module.cc)

- Primary contact: Michael Baird
- Relevant documents: docdb-9195, docdb-10042
- S14-03-06 production performance: docdb-10915
- Input: rb::CellHit from the CalHit module
- Output: rb::Cluster
- Description: This module groups cell hits that are "neighbors" in 4D space-time into a collection. A perfect slice would group the hits from one physics interaction (either neutrino or cosmic ray) into a slice with no contamination. Almost all other reconstruction is performed on these slices. This version of the slicer is used in all production now and replaces older versions seen in the package.
- Standard module label: slicer
- In Production? YES

HoughTransform (MultiHoughT_module.cc)

- Primary contact: Michael Baird
- Relevant documents: docdb-8241
- S14-03-06 production performance: docdb-10934
- Input: rb::Cluster_ from Slicer4D
- Output: rb::HoughResult
- Description: This algorithm applies multiple hough transforms to a slice in order to find the dominant straight line features or “guidelines” which are output. These lines do not have start/endpoints or hits associated with them.
- Standard module label: multihough
- In Production? YES

ElasticArms (ElasticArmsHS_module.cc)

- Primary contact: Mark Messier
- Relevant documents: docdb-7530
- S14-03-06 production performance: docdb-10910
- Input: rb::HoughResult from MultiHough and rb::Cluster from Slicer4D
- Output: rb::Vertex and rb::Prong
- Description: This algorithm takes the slice and uses the hough lines as seeds to fit a 3D global vertex. Algorithm assumes that all activity in the slice comes from one point. Output prongs were used in the fit, but are not refined and meant for analysis.
- Standard module label: elasticarmshs
- In Production? YES

FuzzyKVertex (FuzzyKVertex_module.cc)

- Primary contact: Evan Niner
- Relevant documents: docdb-7648, docdb-9864
- S14-03-06 production performance: docdb-10910
- Input: rb::Cluster from Slicer4D and rb::Vertex from ElasticArms
- Output: rb::Prong (both 2D and 3D prongs are produced, under separate instance labels)
- Description: This algorithm takes the global vertex and fits hits in the slice to prongs based on a k-means clustering algorithm. Optimized toward electron/photon showers but also does tracks. Energy profiles of 2D prongs matched by ks test to make 3D output.
- Standard module label: fuzzykvertex
- In Production? YES

TrackFit (CosmicTrack_module.cc)

- Primary contact: Brian Rebel and Gavin Davies
- Relevant documents: docdb-6890
- S14-03-06 production performance: n/a
- Input: rb::Cluster from Slicer4D
- Output: rb::Track both 2D and 3D
- Description: This algorithm applies a straight line fit to hits in a slice, dropping outliers. It is meant to be very fast and is used primarily for cosmic background rejection and calibration. NOTE current version will not follow curve in muon tracks.
- Standard module label: cosmictrack
- In Production? YES

TrackFit (KalmanTrack_module.cc)

- Primary contact: Nick Raddatz
- Relevant documents: docdb-6828
- S14-03-06 production performance: docdb-10965
- Input: rb::Cluster from Slicer4D
- Output: rb::Track 2D tracks only
- Description: Applies a kalman filter algorithm to stitch hits in a slice together into track segments. Optimized for muon reconstruction, can track through multiple scattering. This module only outputs 2D track candidates.
- Standard module label: kalmantrack
- In Production? YES

TrackFit (KalmanTrackMerge_module.cc)

- Primary contact: Nick Raddatz
- Relevant documents: docdb-6828
- S14-03-06 production performance: docdb-10965
- Input: rb::Cluster from Slicer4D
- Output: rb::Track both 2D and 3D
- Description: Takes the 2D kalman tracks and produces 3D tracks by matching start/end points, unmatched tracks are also output.
- Standard module label: kalmantrackmerge
- In Production? YES

MichelEFilter (SliceMEF_module.cc)

- Primary contact: Daniel Pershey
- Relevant documents: docdb-10256, docdb-10588
- S14-03-06 production performance: n/a
- Input: rb::Cluster from Slicer4D
- Output: me::MichelECluster
- Description: Forms clusters of hits in the noise slice that are near the physics slice in space/time and tags these as Michel electron candidates.
- Standard module label: slicemef
- In Production? YES

MichelEFilter (MichelEFilter_module.cc)

- Primary contact: Ji Liu, Kanika Sachdev
- Relevant documents: docdb-7642
- S14-03-06 production performance: docdb-10908
- Input: rb::CellHit from CalHit and rb::Track from specified tracker
- Output: me::MichelECluster
- Description: Algorithm that tags clusters near the ends of tracks as Michel electron candidates.
- Standard module label: michelecosmictrack, michelekalmantrack
- In Production? YES

Final thoughts

- Particle ID and energy estimation algorithms not covered here. If you have questions I can answer or get you in touch with the right people.
- Many other reconstruction packages exist either in active development or that or obsolete. A job file that runs all the reco algorithms just discussed can be found here.
- A lot of reconstruction is in place, but improvement of existing algorithms or new development is encouraged. Happy Coding!