Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Using ResultsProducers - when storing in Runs, SubRuns or Events just won’t do

Brian Rebel

Art Users Meeting

17 June 2016

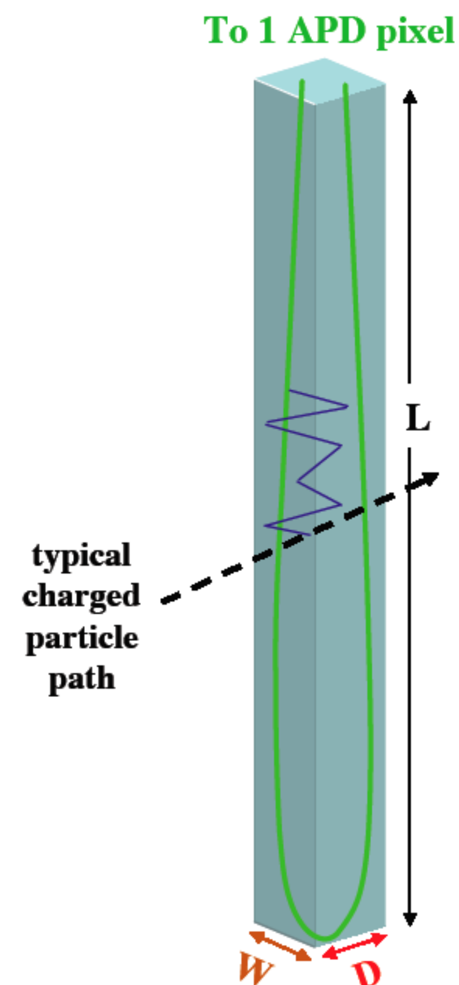
What do you mean Runs/Subruns/Events aren't sufficient?

- The traditional levels of storage in art files have been the art::Run, art::SubRun, and art::Event options
- These are great for any object that is contained within a single one of those containers
- But what about results that have to concatenate information from multiple runs?
  - You could store the concatenation in a single art::Run container, but the user would have to specify which run in their module configuration…and know ahead of time the last run to be seen by the job (difficult if SAM is serving up files)
  - You could also use the endJob part of a job to do the work and write the results into a ROOT file made by the art::TFileService…but then you lose the provenance tracking of art

Fermilab

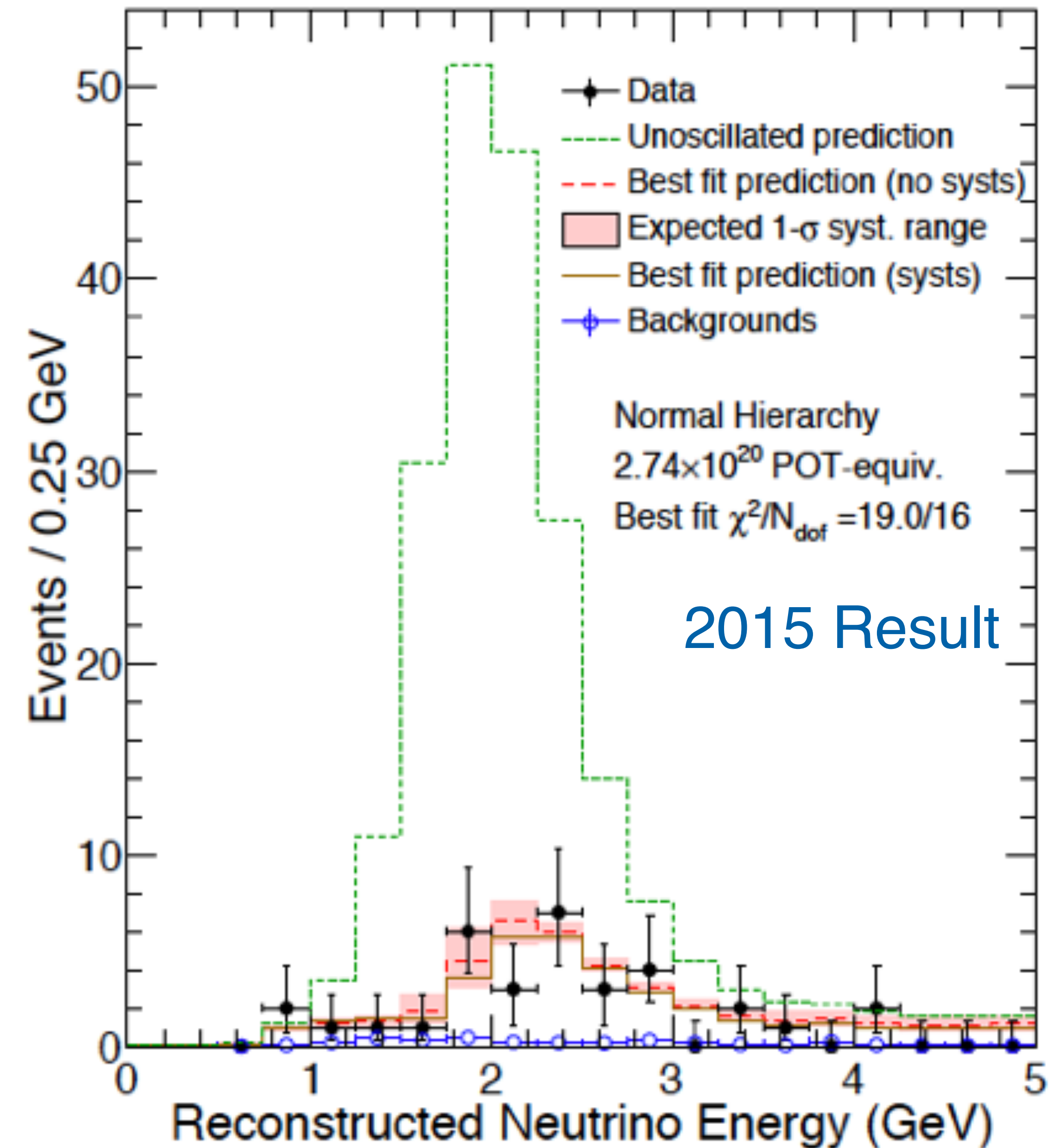
What use cases are you considering?

- The first use case we found for this type of storage/product was the calibration of the NOvA detectors
- In NOvA we have wavelength shifting fibers inserted into cells filled with scintillator to bring the light to APDs
- The scintillation light is attenuated in the fibers and we have to correct for the attenuation
- We do a cell-by-cell (344k cells!) calibration using cosmic ray muons
- No single run has enough muons to provide sufficient statistics in each cell for the calibration
- Also want to have a provenance for how we calibrated the detector



What use cases are you considering?

- The next use case was for NOvA analysis
- The thing we ultimately care about is the full spectrum of neutrino energies at the near and far detectors, not just the spectrum per run
- We need to concatenate the very few neutrinos we observe in the far detector spread across thousands of data files
- Would like to also define a data product that contains the results of the analysis and whose provenance can be tracked
- Also need a way to store the resulting spectra from different detectors in a single file (still on the wish list)



Fermilab

What is an art::Result and how do I use it?

- A recent upgrade to art provided the solution in the form of art::Result
- The art::Result is a container to hold the output of a concatenation across multiple art::Run products, because if you wanted to concatenate art::Events you could do that in art::SubRun, and art::SubRuns can be concatenated in art::Runs.
- That doesn't mean it only sees art::Runs - there are hooks to get at art::SubRuns and art::Events too.

What is an art::Result and how do I use it?

- To use the art::Result you need to write an art::ResultsProducer plugin
  - Notice it is not called a module because it is not a module
  - It has hooks that are not available in modules: clear(), readResults() and writeResults()
- The readResults() is there to read in any data products you created with a previous ResultsProducer
- The writeResults() writes out your new data products

art/Framework/core/ResultsProducer.h

```
// * Subclass implementations *must* invoke:
//
//   DEFINE_ART_RESULTS_PLUGIN(class)
//
// * Subclasses *must* define:
//
//   Constructor(fhicl::ParameterSet const &);
//
//     Declare data products to be put into the Results (via
//     art::Results::put() in writeResults() below) using produces<>(),
//     in a similar fashion to producers and filters.
//
//   void writeResults(art::Results &);
//
//     Called immediately prior to output file closure. Users should
//     put() their declared products into the Results object. Using
//     getLabel() here will access only those results products that were
//     put() by plugins executed earlier for the same output module. In
//     order to access results products from input files, use
//     readResults(), below. Note that for the purposes of product
//     retrieval, the "module label" of a results product is
//     <output-module-label>#<results-producer-label> eg for results
//     producer label rp1 defined for output module o1, any product
//     produced by rp1 will have the label, o1#rp1.
//
//   void clear();
//
//     In this function (called after writeResults()), the user should
//     reset any accumulated information in preparation for (possibly)
//     accumulating data for the next output file.
//
// * Subclasses *may* define:
//
//   void beginJob();
//
//   void endJob();
//
//   void beginSubRun(SubRun const &);
//
//   void endSubRun(SubRun const &);
//
//   void beginRun(Run const &);
//
//   void endRun(Run const &);
//
//   void event(Event const &);
//
//   void readResults(art::Results const &);
//
//     Access any results-level products in input files here. The user
//     is entirely responsible for combining information from possibly
//     multiple input files into a possible output product, and for
```

What is an art::Result and how do I use it?

```
namespace fnex {
  class FitPoint : public art::ResultsProducer {
  public:
    explicit FitPoint(fhicl::ParameterSet const& pset);
    virtual ~FitPoint();

    void readResults (art::Results const& r);
    void writeResults(art::Results      & r);
    void reconfigure(const fhicl::ParameterSet& p);
    void clear();
    void beginJob();

  private:

    art::ServiceHandle<art::TFileService>      fTFS;                     ///< TFileService
    fhicl::ParameterSet                        fFitterParameters;        ///< parameter set for all experiments to run
    std::unique_ptr<fnex::InputPoint>          fInitialGuessInputPoint; ///< Initial point passed to fitter
    std::unique_ptr<fnex::FitAlgorithm>        fFit;                     ///< algorithm for doing the fit
    fnex::EventListMap                         fEventLists;              ///< the lists of events we are going to use, data and MC
    std::unique_ptr<fnex::EventListManipulator> fManipulator;            ///< helper object to work with event lists from TTrees
    bool                                       fDrawDebugPlots;          ///< turn on/off drawing of debug plots
  };
```

- This is an example plugin from the NOvA code
- It is designed to fit the data with the simulation and store the output of that fit

What is an art::Result and how do I use it?

- The constructor calls produces<T> just like any EDFilter or EDProducer

```
//......................................................................
FitPoint::FitPoint(fhicl::ParameterSet const& pset)
{
  this->reconfigure(pset);
  produces<fnex::InputPoint >();
  produces<fnex::PointResult>();
  return;
}
```

- The clear() method allows the user to clear out any previously read in collections if desired. It is called after the writeResults() method

```
//......................................................................
// Method to clear out the collections of data products after the
// writeResults method is called.
void FitPoint::clear()
{
  fEventLists.clear();

  return;
}
```

- The readResults() method in this case does not read in any previous data products from the art::Results (we could have done the work in the beginJob() method)

```
//......................................................................
void FitPoint::readResults(art::Results const& r)
{
  fEventLists = fManipulator->Deserialize();

  return;
}
```

Fermilab

What is an art::Result and how do I use it?

- The writeResults() method here calls a function to do the fit
- It then makes some plots to store in the file produced by the TFileService
- Finally it puts the initial guess of the fitter and the final result into the art::Results object
- The required macro to register it with art is at the bottom

```
 //.......................................................................
 void FitPoint::writeResults(art::Results & r)
 {

   // need to setup the fit algorithm object here, pass it the appropriate
   // parameter sets and the appropriate fnex::EventLists
   fnex::PointResult pointResult;

   fFit->Fit(fEventLists, *fInitialGuessInputPoint, pointResult);
   //if(fDrawDebugPlots){
     art::TFileDirectory fitDir = fTFS->mkdir( "Fit" );
     fFit->DrawPlots(&fitDir);
     LOG_VERBATIM("FitPoint")
     << "INPUT POINT WAS: \n"
     << *fInitialGuessInputPoint
     << "\n"
     << "FIT POINT WAS: \n"
     << pointResult;
   //}

   std::unique_ptr<fnex::PointResult> pr(new fnex::PointResult(pointResult));

   r.put(std::move(fInitialGuessInputPoint));
   r.put(std::move(pr));
 } // end writeResults

} // end fnex namespace

namespace fnex{

 DEFINE_ART_RESULTS_PLUGIN(FitPoint);

}
```

What is an art::Result and how do I use it?

- The configuration is a bit different from what you are used to
- It is configured in association with an output, “fit” in the example to the left
- You have a results block for collecting all the producers, which are a sub-block
- You have to put the producers into a path, “rpPath” in the example
- Put as many producers in the results block as you need

```
outputs:
{
 fit:
 {
  module_type: RootOutput
  fileName:    "output.root"
  results:
  {
   producers:
   {
    oscfit: @local::test_fitpoint
   }
   rpPath: [ oscfit ]
  }
 }
}
```


Fermilab

What is an art::Result and how do I use it?

- Getting it to compile is easy with the art CMake functions
- Just use the simple_plugin option
- Specify it as a “plugin” and not a “module”
- Add in the required libraries to the link list
- After a brief wait, you are ready to test it out

```
simple_plugin(FitPoint "plugin"
               FNEXcore
               FNEXcustom
               FNEXfitter
               FNEXmodules
               FNEXvars
               FNEXutilities
               FNEXdataProducts
               FNEXutilities
               ${ART_FRAMEWORK_PRINCIPAL}
               ${ART_PERSISTENCY_COMMON}
               ${MF_MESSAGELOGGER}
               ${MF_UTILITIES}
               ${CETLIB}
               ${FHICLCPP}
               BASENAME_ONLY
              )
```

Ready to Try It?

- Results and ResultsProducers fill a need for a way to store data products outside of the Run/SubRun/Event categories that have always been part of art
- They have been tested in NOvA and filled the desired role well
- It would be great if we could get them to read results from multiple types of input files (data and MC, different detectors, etc).