The new `RawData`, `Wire` and `Hit` objects

or, “what have you done with my data”

Gianluca Petrillo, Erica Snider

University of Rochester/Fermilab

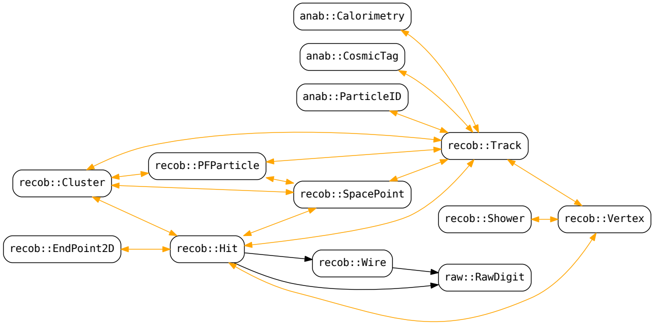
MicroBooNE IIT Workshop, January 7th , 2015

Outline

Data products review

`LArSoft` comprises about 40 classes that can be serialized into ROOT files (“data products”).

Here the largest block of interconnected products is shown:



Class relationships are expressed through *data members* (as `art` pointers) and *associations* (a separate data product).

The review process

These products are currently being reconsidered:

- do they still reflect useful concepts?
- do they still meet experiments' current needs?
- do they look flexible enough to accommodate future needs?
- can they read outside an `art` environment?

The review process is such organized:

1. there is a lot of discussions with code authors
2. we collect the ideas and synthesize proposals
3. proposals are evaluated by a specific meeting *with experiments and framework representatives*
4. often, it's `GOTO 1`
5. finally, it's time to implement them

Planned changes

These data products are identified as revision candidate:

`raw::RawDigit` removing pedestal information

`recob::Wire` removing `art` dependency

`recob::Hit` removing `art` dependency, changing some data members

`recob::Cluster` adding a few data members, removing a few others

`recob::EndPoint2D` splitting into a geometric 2D point class and a reconstructed class

`recob::Track` redesigning the concept, splitting in different classes

`raw::RawDigit`

- `RawData::fADC` became private
- `RawData::fPedestal` is candidate for removal: its information should ideally come from a database
- documentation shows some usage example and precise definition of data members

`fADC`	replaced by `ADCs()` (read-only)
`fSigmaPedestal`	removed

Memento!

`raw::RawDigit` data is often compressed: uncompress it with

```
std::vector<short> uncompressed_digits(rawdigit->Samples());
raw::Uncompress
  (rawdigit->ADCs(), uncompressed_digits, rawdigit->Compression());
```

(`raw::Uncompress()` is defined in `RawData/raw.h`)

`recob::Wire`

- `Wire::fRawDigit` was removed
- `Wire::Channel()` was added
- `Wire::fSignalType` was removed
- documentation updated

`recob::Wire` class is now `art`-independent. Consequences:

1. no direct way to connect to the original `raw::RawDigit`
2. constructors are changed (no `raw::RawDigit` pointer any more)

`recob::Wire`: pointer removal

- `art` pointer to `raw::RawDigit` has been removed from `recob::Wire`
- **prescription**: a producer creating `recob::Wire` from `raw::RawDigit` should also create their 1-to-1 association
- `CalWire`-like modules in `LArSoft`, uboonecode and lbnecode have been updated to produce this association
- the pointer was used (almost?) exclusively to get channel number; now can be accessed directly as `recob::Wire::Channel()`
- channel number has now its own data type, `raw::ChannelID_t` (defined in `SimpleTypesAndConstants/RawTypes.h`); a special constant `raw::InvalidChannelID` represents... an invalid channel ID
- do you still need the original `raw::RawDigit`?
  I am working to write some generic (meaning also somehow inefficient) code to get blindly from a `art::Ptr<recob::Wire>` to its associated `raw::RawDigit`...

recob::Wire: raw::RawDigit access

In the meanwhile, an efficient wire loop:

```
// if you have the wire handle from the event...
art::ValidHandle<recob::Wire> Wires
  = event.getValidHandle(CalWireModuleLabel);
// ... query for the Wire-RawDigit association from the same module
art::FindOneP<raw::RawDigit> RawDigits(Wires, event, CalWireModuleLabel);

for (size_t iWire = 0; iWire < Wires->size(); ++iWire) {
  // this is the wire...
  recob::Wire const& wire = Wires->at(iWire);
  // ... and this is the associated RawDigit
  art::Ptr<raw::RawDigit> const& pDigit = RawDigits->at(iWire);
  // ...
} // for
```

Requires:

```
#include <art/Framework/Core/FindOneP.h>
#include "art/Framework/Principal/Handle.h"
#include "RecoBase/Wire.h>
#include "RawData/RawDigit.h>
```

Warning: slide code! not compiled, might be just wrong

recob::Wire: construction

- constructor has become simple, taking a list of all data members
- for convenience, a *wire creation helper* is also provided

From uboonecode/uboone/CalData/CalWireROI_module.cc:

```
art::Ptr<raw::RawDigit> digitVec; // pointer to the original RawDigit
std::unique_ptr<std::vector<recob::Wire>> wirecol; // future product
std::unique_ptr<art::Assns<raw::RawDigit,recob::Wire>> WireDigitAssn;
recob::Wire::RegionsOfInterest_t ROIVec; // signal regions of interest

// create the new wire directly in wirecol
wirecol->push_back
  (recob::WireCreator(std::move(ROIVec),*digitVec).move());
// add an association between the last object in wirecol
// (that we just inserted) and digitVec
if (!util::CreateAssn
  (*this, evt, *wirecol, digitVec, *WireDigitAssn, fSpillName))
{
  throw art::Exception(art::errors::InsertFailure) /* ... */;
} // if failed to add association
```

The helper recob::WireCreator is defined in:

```
#include "RecoBaseArt/WireCreator.h"
```

recob::Wire: “but I want...”

RawDigit()	gone; see previous slides
RawDigit()->Channel()	use Channel()
SignalType()	really?? anyway see below

To get the signal type you need the geometry information:

```
recob::Wire const& wire; // ... filled with a real wire
// geo::SigType sigType = wire.SignalType(); // removed!!
art::ServiceHandle<geo::Geometry> geom;
geo::SigType sigType = geom->SignalType(wire.Channel());
```

Requires:

```
#include "SimpleTypesAndConstants/geo_types.h"
#include "Geometry/Geometry.h"
#include "RecoBase/Wire.h"
```

Same Warning: slide code! not compiled, might be just wrong

`recob::Hit`

- removed some ambiguously defined data members, added some replacements for them
- added some new ones: `RMS()`, `DegreesOfFreedom()`, ...
- `Hit::fRawDigit` and `Hit::fWire` were removed
- documentation updated

`recob::Hit` class is now `art`-independent. Consequences:

1. no direct way to connect to the original `raw::RawDigit` and `recob::Wire`
2. constructors are changed (no `raw::RawDigit` nor `recob::Wire` pointer any more)

`recob::Hit`: times

- `StartTime()` and `EndTime()` have an ambiguous definition
- the code usually seems to expect them to be $t = t_{\text{peak}} \pm \sigma_{\text{hit}}$
- most common use: `(EndTime()-StartTime())/2.)`: peak half-width
- second common use: tick time range for time matching

New approach:

- `StartTime()` and `EndTime()` ambiguous: removed!
- enter `RMS()`, related to hit width (σ for Gaussian hits)
- also, and *mostly unrelated*: added `Signal()`: the waveform portion the hit was extracted from
- also add signal range time extremes, `StartTick()` and `EndTick()`, *in TDC count units*

recob::Hit: pointer removal and construction

- same situation as for recob::Wire: simple constructors...
- for convenience, a *hit creation helper* is also provided

Code from larreco/HitFinder/GausHitFinder_module.cc:

```
produces<std::vector<recob::Hit>>();
```

Listing 1: In module constructor: equivalent to produces<>() (old code)

```
std::unique_ptr<std::vector<recob::Hit>> hcol; // data product
art::Ptr<recob::Wire> wire; // original wire (with pointer to raw digit)


// construct a hit (including pointers to wire and rawdigit)
recob::Hit hit(
   wire, wid, StartTime[dd], EndTime[dd], MeanPosition[dd],
  MeanPosError[dd], Charge[dd], ChargeError[dd], Amp[dd], AmpError[dd],
              NumOfHits[dd],     FitGoodness[dd]
  )
  );
// move the hit into the product
hcol.emplace_back(std::move(hit));

evt.put(std::move(hcol)); // put the data product into the event
```

Listing 2: In produce() (old code)

recob::Hit: pointer removal and construction

- same situation as for recob::Wire: simple constructors...
- for convenience, a *hit creation helper* is also provided

Code from larreco/HitFinder/GausHitFinder_module.cc:

```
recob::HitCollectionCreator::declare_products(*this);
```

Listing 3: In module constructor: equivalent to produces<>()

```
recob::HitCollectionCreator hcol(*this, evt); // data product proxy
art::Ptr<recob::Wire> wire; // original wire
art::Ptr<raw::RawDigits> rawdigits; // original rawdigit

// create a hit with HitCreator
recob::HitCreator hit(
  *wire, wid, startT, endT, RMS[dd],      MeanPosition[dd],
  MeanPosError[dd], Amp[dd], AmpError[dd], Charge[dd], ChargeError[dd],
  SumADC[dd], NumOfHits[dd], dd, FitGoodness[dd], FitNDF[dd],
  std::vector<float>(signal.begin() + startT, signal.begin() + endT)
  );
// move the hit into the product and associate it
hcol.emplace_back(hit.move(), wire, rawdigits);

hcol.put_into(evt); // put the data product into the event
```

Listing 4: In produce()

`recob::Hit`: construction (commentary)

- `recob::HitCreator` helper class makes creation of new `recob::Hit` roughly similar to the old one
- pointers to `raw::RawDigit` and `recob::Wire` have been removed, therefore...
- **prescription**: a producer creating `recob::Hit` should also produce associations to its `recob::Wire` and `raw::RawDigit`, as proper
- a class, called `recob::HitCollectionCreator`, simplifies the creation of a `recob::Hit` data product together with its (optional) associations
- both declared in `lardata/RecoBaseArt/HitCreator.h`
- `HitFinder`-like modules in `LArSoft`, uboonecode and lbnecode have been updated (checks needed!)

`recob::Hit`: other additions

Additions:

- there may be multiple hits in the same time range: we allow each hit to know
  - how many hits are reconstructed in that range (`Multiplicity()`)
  - which of them this one is (`LocalIndex()`)
- `Channel()`: ID of the channel the hit is on
- `DegreesOfFreedom()` of the hit finding process (pairs with the existing `GoodnessOfFit()`)
- `SummedADC()`, plain sum of the ADC counts covered by the hit (in contrast with `Integral()`, that is typically a direct fit parameter)

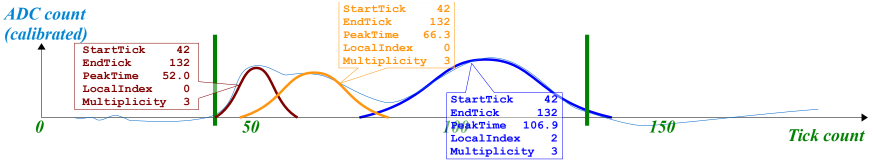


Illustration of a train of hits (multiplicity: 3) in the same region of interest

`recob::Hit`: “but I want...”

`RawDigit(),Wire()`	both gone (see above)
`RawDigit()->Channel()` `Wire()->RawDigit()->Channel()`	use `Channel()`
`Charge(),Charge(false)`	⇒ `Integral()`
`Charge(true)`	⇒ `PeakAmplitude()`
`StartTime(),EndTime()` `StartTime()` `EndTime()`	see above, but also: maybe `PeakTimeMinusRMS()` maybe `PeakTimePlusRMS()`
`SigmaStartTime()` `SigmaEndTime()`	removed, not replaced

I ask the authors of hit finder algorithms to verify and correct:

- `fLocalIndex` (most often I just plugged in a -1)
- `fStartTick`, `fEndTick`
- `fSignal`: did I get the right signal vector?
- `fNDF`
- `fSummedADC` (usually $\sum_{k=\texttt{fStartTick}}^{\texttt{fEndTick}} \texttt{fSignal}_k$)

Wait a moment... won't this break the existing data?... yep.

Time line:

- this revolution will render the existing data unreadable, a textbook example of breaking change
- testing of the new classes should start immediately after this workshop; *you are welcome to start it today!*
- changes to `recob::Cluster` will be discussed and summarized during this workshop
- we are preparing a draft implementation of a proposal for `recob::Track`
- we want MicroBooNE to be able to use the new classes in your next Monte Carlo Challenge (*end of January 2015*)

This is still a *very ambitious* goal.

Status

- changes to `raw::RawDigit`, `recob::Wire` and `recob::Hit` are in a published branch: `feature/DataProductRevision`
- that code compiles, but checks by the respective code authors and testing are due
- a itemized status is available on the web: `https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/DataProductsArchitectureActionItems`
- there is still plenty of room for changes
- Eric Church will help with testing; but checks from the authors are still needed!
- err... did I mention that authors of the algorithms should verify the changes make sense?

  No, really: there are a lot of placeholders in the code, that you will want to replace with real information

Outlook

- we are reviewing the main reconstruction data products
- we have a partial implementation in place; working fast on the rest
- as you may have overheard by now, we ask the help of algorithm authors to complete the work
- we also need the experiments to actively scrutiny the changes under discussion

Additional material

The other example: `recob::Track`

The track object (`recob::Track`) has some problems:

- lacks a definition about density of trajectory points
- contains elements that are algorithm-dependent
- is able to host Bézier tracks only by a (ugly) hack

A proposal, under discussion, consists in:

- leaving in `recob::Track` only general track properties: start and end point, intermediate trajectory points, track quality
- prescribing by policy the trajectory point density
- delegating the continuous representation to another object (`recob::Trajectory`)
- moving the momentum estimation into a new object (`recob::Momentum`)
- binding all these objects with associations

Example of direct use of data in another framework

Say that your favourite framework has a data model where all data classes inherit from `TDataObject` (may be even ROOT's `TObject`). You can define a base class:

```
#include "ArtlessFW/TDataObject.h"
#include "RecoBase/Hit.h"

template <typename Data>
class TDataWrapper: public Data, public TDataObject { /* ... */ };

namespace wrapped {
  class Hit: public TDataWrapper<recob::Hit> { /* ... */ };
}
```

- `wrapped::Hit` includes the very interface of `recob::Hit`
- ... plus additional methods that you care to define specifically for it
- you are using the original `lardata/RecoBase/Hit.h`[`.cxx`]
- you might even read directly from the art tree into `wrapped::Hit`

* I haven't tried the limits of this model, but it might even work
(if I were sure, this slide would not be in the appendix!)

The current proposal: raw digits and wires

The review has so far focused on six data products.

`raw::RawData`: straight readout from DAQ

- removal of pedestal information, *still under discussion*

`recob::Wire`: calibrated and filtered signal on a channel

- removal of `art` pointers
- addition of channel ID member
- removal of `SignalType()`

The current proposal: hits and clusters

`recob::Hit`: deposition from a particle on a wire

- removal of `art` pointers
- removal of unused information
- addition of hit width (RMS)
- more precise definition of start and end times (now defined in TDC tick units), peaks and integrals

`recob::Cluster`: set of hits from the same particle

- removal of unused information (dQ/dx)
- addition of shower/track discriminating variables
- more things being discussed

The current proposal: tracks and 2D end-points

`recob::Track`: reconstructed path of a ionizing particle

A lot of discussion ongoing... implementing multiple classes:

- a discrete track, result of the tracking fit
- a trajectory, continuous representation of the particle path
- a momentum describing the initial impulse of the particle
- additional classes will hold results specific to the algorithms

`recob::EndPoint2D`: a point on a wire plane

Still under discussion. Split into:

- a geometric object: just two coordinates
- an object result of a reconstruction algorithm (with goodness of fit etc.)