Reviewing data products

Part I: `recob::Hit` and `recob::Track`

Gianluca Petrillo

LArSoft architecture committee, October 13th , 2014

Outline

1. Introduction

2. `recob::Hit`

3. `recob::Track`

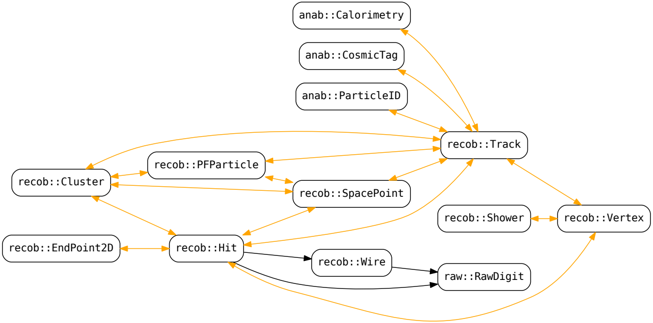
4. Summary

Data products review

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

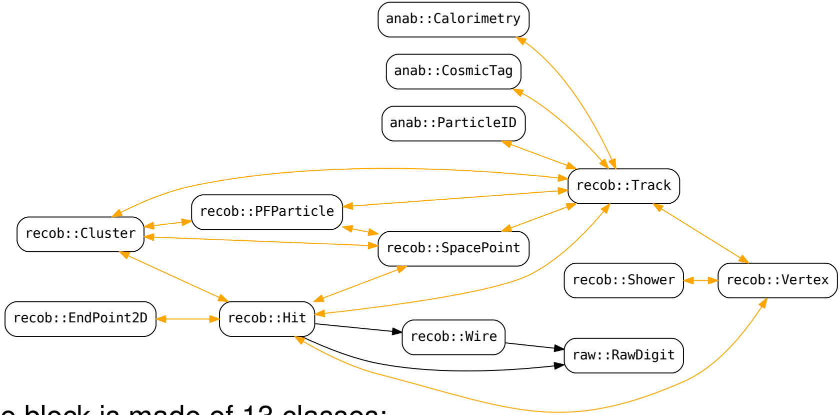
Class relationships are expressed through data members (as pointers) and associations (a separate data product).

"Blocks" of related data products can be thus identified; e.g.



I start the review with the largest block in RecoBase *(shown above)*.

The “RecoBase block”



The block is made of 13 classes:

- some work as additional attributes (e.g. `anab` classes to `Track`)
- some work as bridges (e.g. `PFParticle`)
- some are data-member-like (e.g. `SpacePoint`)

What is what

Each data product should describe a concept:

`raw::RawDigit` sequence of ADC counts on one channel

`recob::Wire` signal on one channel as function of time (TDC)

`recob::Hit` observed charge from a particle on a single wire

`recob::Cluster` sequence of hits showing geometrical correlation

`recob::EndPoint2D` two-dimensional coordinate on a plane

`recob::SpacePoint` three-dimensional coordinate

`recob::Shower` observed information from a showering particle

`recob::Track` observed information from a single charged particle

`anab::Calorimetry` calorimetric information of a particle

`anab::CosmicTag` cosmic-ray-like attributes of a particle

`anab::ParticleID` hypothesis on the nature of a particle

`recob::Vertex` point in the detector origin of particles

`recob::PFParticle` the evolution of particles (flow) in the event

Known issues

We are aware of some issues in this design:

- `raw::RawDigit`, `recob::Wire` and `recob::Hit` interdepend
- `recob::Cluster` might need additional information
- `recob::Track` includes two different concepts
- `recob::Shower` is being redefined
- `recob::PFParticle` role is not completely clear yet

We give higher priority to changes that:

- are central in the current reconstruction and analysis flow
- require a deeper rethinking of the classes
- are likely not to be backward-compatible

In general, additions can be handled on demand.

Outline

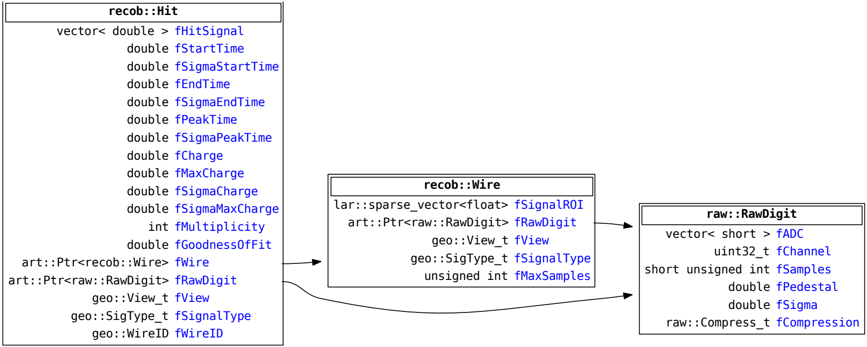
1 Introduction

2 `recob::Hit`

3 `recob::Track`

4 Summary

recob::Hit



```
recob::Hit
  vector< double > fHitSignal
            double fStartTime
            double fSigmaStartTime
            double fEndTime
            double fSigmaEndTime
            double fPeakTime
            double fSigmaPeakTime
            double fCharge
            double fMaxCharge
            double fSigmaCharge
            double fSigmaMaxCharge
               int fMultiplicity
            double fGoodnessOfFit
art::Ptr<recob::Wire> fWire
art::Ptr<raw::RawDigit> fRawDigit
       geo::View_t fView
    geo::SigType_t fSignalType
       geo::WireID fWireID
```

```
recob::Wire
lar::sparse_vector<float> fSignalROI
  art::Ptr<raw::RawDigit> fRawDigit
              geo::View_t fView
           geo::SigType_t fSignalType
             unsigned int fMaxSamples
```

```
raw::RawDigit
    vector< short > fADC
           uint32_t fChannel
 short unsigned int fSamples
             double fPedestal
             double fSigma
  raw::Compress_t fCompression
```

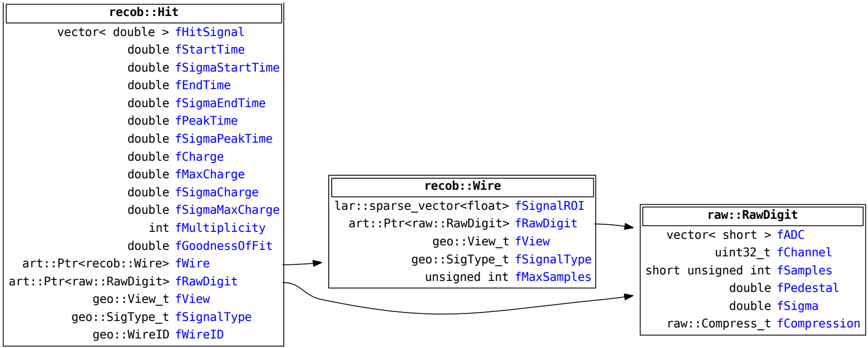
Three classes are interdependent:

raw::RawDigit sequence of ADC counts on one channel

recob::Wire signal on one channel as function of time (TDC)

recob::Hit observed charge from a particle on a single wire

recob::Hit changes: channel ID



```
recob::Hit
      vector< double > fHitSignal
                double fStartTime
                double fSigmaStartTime
                double fEndTime
                double fSigmaEndTime
                double fPeakTime
                double fSigmaPeakTime
                double fCharge
                double fMaxCharge
                double fSigmaCharge
                double fSigmaMaxCharge
                   int fMultiplicity
                double fGoodnessOfFit
  art::Ptr<recob::Wire> fWire
 art::Ptr<raw::RawDigit> fRawDigit
           geo::View_t fView
        geo::SigType_t fSignalType
           geo::WireID fWireID
```

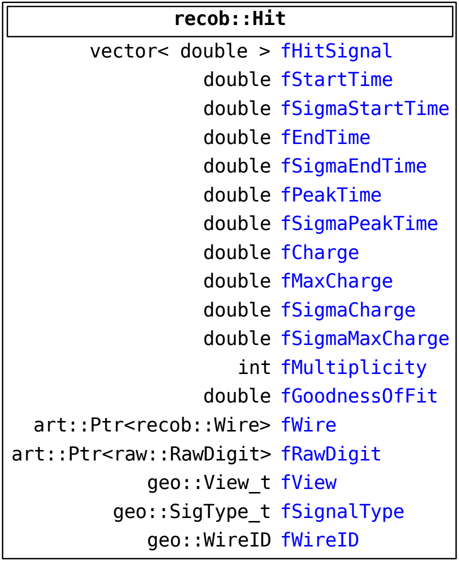
```
recob::Wire
lar::sparse_vector<float> fSignalROI
  art::Ptr<raw::RawDigit> fRawDigit
              geo::View_t fView
           geo::SigType_t fSignalType
             unsigned int fMaxSamples
```

```
raw::RawDigit
     vector< short > fADC
            uint32_t fChannel
short unsigned int fSamples
              double fPedestal
              double fSigma
   raw::Compress_t fCompression
```

Proposed changes:

1. recob::Hit: add channel number
2. recob::Wire: add channel number
3. recob::Hit and recob::Wire: remove pointers
   we may consider writing associations instead
4. recob::Hit: change the channel ID retrieval logic
   use the added channel field

recob::Hit changes: precision



```
                   recob::Hit
       vector< double > fHitSignal
                 double fStartTime
                 double fSigmaStartTime
                 double fEndTime
                 double fSigmaEndTime
                 double fPeakTime
                 double fSigmaPeakTime
                 double fCharge
                 double fMaxCharge
                 double fSigmaCharge
                 double fSigmaMaxCharge
                    int fMultiplicity
                 double fGoodnessOfFit
  art::Ptr<recob::Wire> fWire
art::Ptr<raw::RawDigit> fRawDigit
            geo::View_t fView
         geo::SigType_t fSignalType
            geo::WireID fWireID
```

fHitSignal `double` ⇒ `float`

fXxxCharge `double` ⇒ `float`

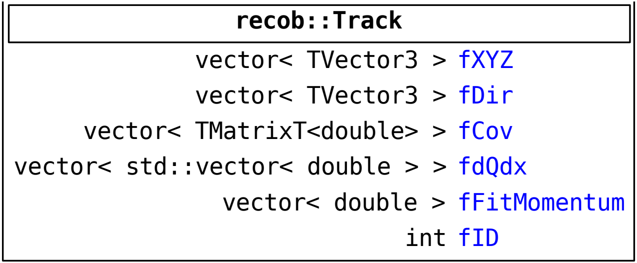
fGoodnessOfFit `double` ⇒ `float`

Anything else?

Outline

1. Introduction
2. `recob::Hit`
3. `recob::Track`
4. Summary

recob::Track



```
                   recob::Track
              vector< TVector3 > fXYZ
              vector< TVector3 > fDir
      vector< TMatrixT<double> > fCov
vector< std::vector< double > > fdQdx
                vector< double > fFitMomentum
                             int fID
```

Merges two concepts:

trajectory position, direction, uncertainty of a trajectory hypothesis based on hits (e.g. straight line, broken line, Bézier curve)

calorimetry dQ/dx and momentum fit

It also suffers from other issues:

- does not provide a continuous trajectory
- BezierTracker algorithm saves `recob::Track`, losing information

Proposal for `recob::Track`

We propose to reorganize the information with three classes:

`recob::Track` estimated waypoints of a single charged particle

`recob::Trajectory` continuous representation of the path of a particle

`recob::Momentum` point-by-point estimation of particle momentum

(challenge: find worse names)

There is one trajectory class for each functional form (e.g. `recob::BezierTrajectory`, `recob::DiscreteTrajectory`, `recob::CubicInterpolationTrajectory`, ...), all sharing the interface.

Scavenging `recob::Track`

We propose to carve out the momentum information as:

```
class Track {
   std::vector<TVector3> fXYZ;
   std::vector<TVector3> fDir;
   std::vector<TMatrixT<double>> fCov; // 6x6 matrix
   float fQuality; // NEW: a chi2-probability-like quantity
   int fID;
   //...
}; // class Track

class Momentum {
   std::vector<TVector3> fFitMomentum; // becomes a vector
   std::vector<TVector3> fSigma;
   //...
}; // class Momentum
```

Not clear (to me!) what `std::vector<std::vector<float>> fdQdX` is and where it belongs.

Issues with Kalman trackers

Issues with adaptation of Kalman trackers:

- track point uncertainty is incomplete: Kalman fits $\left|\vec{p}\right|$ together with the other points, but no $\sigma_{\left|\vec{p}\right|,x}$ covariance is present
- can the momentum estimation be expanded to 3D?
- how many covariances should be present in `recob::Momentum`?

```
class Track {
   std::vector<TVector3> fXYZ;
   std::vector<TVector3> fDir;
   std::vector<TMatrixT<double>> fCov; // 6x6 matrix
   float fQuality; // NEW: a chi2-probability-like quantity
   //...
}; // class Track

class Momentum {
   std::vector<TVector3> fFitMomentum; // becomes a vector
   std::vector<TMatrixT<double>> fCov; // px/py/pz covariances
   std::vector<std::vector<TVector3>> fCovXYZ; // cov. with position
   std::vector<std::vector<TVector3>> fCovDir; // cov. with direction
   // ... and more according to your generosity
}; // class Momentum
```

Implementations of `recob::Trajectory`

```
class Trajectory {
      public:
   TVector3 GetPositionAt(double s) = 0;
   TVector3 GetDirectionAt(double s) = 0;
   // and uncertainties, etc...
}; // class Trajectory

class BezierTrajectory: public Trajectory {
   BezierData params; ///< trajectory parameters
      public:
   // constructor, interface implementation ...
}; // class BezierTrajectory

class CubicInterpolationTrajectory: public Trajectory {
   CubicInterpolationData params; ///< trajectory data and parameters
      public:
   // constructor, interface implementation ...
}; // class CubicInterpolationTrajectory
```

What do we write into the event? Here two possible implementations:

1. the derived classes (e.g. `recob::BezierTrajectory`)
2. the enclosed plain old data structure (e.g. `recob::BezierData`)

Implementations of `recob::Trajectory` (II)

In both cases, we will `get()` from the event something,

- `std::vector<recob::BezierTrajectory>`
- `std::vector<recob::BezierData>`

that will need to be wrapped, for the interface to be transparently used:

- `void CoolAlg(std::vector<const recob::Trajectory*>)`

Modules still need to know in advance which trajectory or parameter class they are reading.

Outline

1. Introduction
2. `recob::Hit`
3. `recob::Track`
4. Summary

Summary

- we have a proposal for fixing `recob::Wire` and `recob::Hit`
- some further thinking is needed about disincorporating the momentum from `recob::Track`
- we have a proposal for a new `recob::Trajectory` class
- we need to learn to use better names
- we need to address technical points: how to replace `art::Ptr` and `art::Assns` with a framework-independent structure

Additional material

Simple references and associations

We need to replace `art::Ptr` and `art::Assns` with a framework-independent subsystem. The new simple classes:

- they should provide the same kind of functionality as the art structures currently do
- they have to rely to some “algorithm” replacing the services offered by the framework

The functionality provided by art classes is based on:

1. being able to locate a specific source (“data product”)
   art uses “product ID”; can the same product ID be used with generic, non-art-aware code?
2. being able to locate a specific object in that product
   products being simple objects or immutable collections, a index suffices