ProtoDUNE SP Analysis Utilities

Leigh Whitehead

ProtoDUNE DRA Meeting

22/08/18

Introduction

- I want to simplify “doing an analysis” as much as possible
- Analysers shouldn’t have to be experts in the LArSoft and art frameworks to perform what should be simple tasks
- I have written a number of utilities that live in dunetpc:
  - dune/Protodune/Analysis/
- These are by no means complete, and I give a few examples here
- If there is anything that “should be easy” or there “must be a simpler way” then we can try to write a utility function to help

List of Utilities

- I have written the following utilities:

- ProtoDUNEPFParticleUtils
  - Aim here it to simplify using the output from Pandora
  - Allow users to access important information using a single function
  - Encapsulate the Pandora metadata

- ProtoDUNETrackUtils
  - Provide simple access to anab::T0 and anab::CosmicTag information

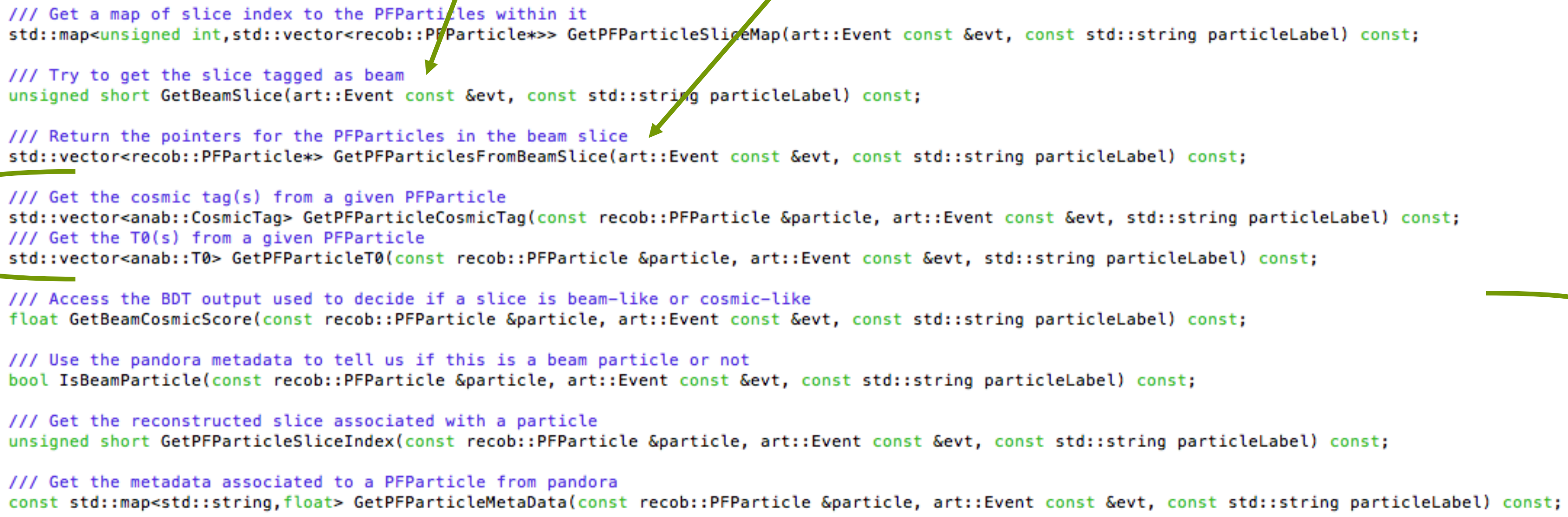
- ProtoDUNETruthUtils
  - Provide the matching between the GEANT and beam generator truth
  - Match true particles to reconstructed tracks

PFParticles

Group primary PFParticles by their Pandora slice

Identify the slice tagged as beam-like

Get the PFParticle(s) from the beam slice



```
/// Get a map of slice index to the PFParticles within it
std::map<unsigned int,std::vector<recob::PFParticle*>> GetPFParticleSliceMap(art::Event const &evt, const std::string particleLabel) const;

/// Try to get the slice tagged as beam
unsigned short GetBeamSlice(art::Event const &evt, const std::string particleLabel) const;

/// Return the pointers for the PFParticles in the beam slice
std::vector<recob::PFParticle*> GetPFParticlesFromBeamSlice(art::Event const &evt, const std::string particleLabel) const;

/// Get the cosmic tag(s) from a given PFParticle
std::vector<anab::CosmicTag> GetPFParticleCosmicTag(const recob::PFParticle &particle, art::Event const &evt, std::string particleLabel) const;
/// Get the T0(s) from a given PFParticle
std::vector<anab::T0> GetPFParticleT0(const recob::PFParticle &particle, art::Event const &evt, std::string particleLabel) const;

/// Access the BDT output used to decide if a slice is beam-like or cosmic-like
float GetBeamCosmicScore(const recob::PFParticle &particle, art::Event const &evt, const std::string particleLabel) const;

/// Use the pandora metadata to tell us if this is a beam particle or not
bool IsBeamParticle(const recob::PFParticle &particle, art::Event const &evt, const std::string particleLabel) const;

/// Get the reconstructed slice associated with a particle
unsigned short GetPFParticleSliceIndex(const recob::PFParticle &particle, art::Event const &evt, const std::string particleLabel) const;

/// Get the metadata associated to a PFParticle from pandora
const std::map<std::string,float> GetPFParticleMetaData(const recob::PFParticle &particle, art::Event const &evt, const std::string particleLabel) const;
```

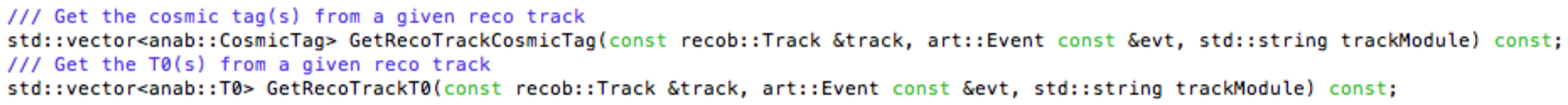
Get associated T0 and CosmicTag objects

NB: No CosmicTag objects from Pandora (yet)

Access Pandora metadata for a given track

Tracks

- Only a couple of functions so far…
    - Access associated anab::T0 and anab::CosmicTag objects

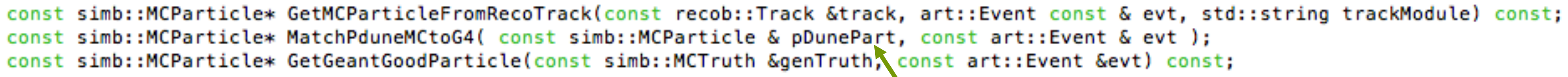


```
/// Get the cosmic tag(s) from a given reco track
std::vector<anab::CosmicTag> GetRecoTrackCosmicTag(const recob::Track &track, art::Event const &evt, std::string trackModule) const;
/// Get the T0(s) from a given reco track
std::vector<anab::T0> GetRecoTrackT0(const recob::Track &track, art::Event const &evt, std::string trackModule) const;
```

Truth

- Three functions currently implemented

Get the best-match MCParticle for a given reconstructed track



```
const simb::MCParticle* GetMCParticleFromRecoTrack(const recob::Track &track, art::Event const & evt, std::string trackModule) const;
const simb::MCParticle* MatchPduneMCtoG4( const simb::MCParticle & pDunePart, const art::Event & evt );
const simb::MCParticle* GetGeantGoodParticle(const simb::MCTruth &genTruth, const art::Event &evt) const;
```

Get the GEANT MCParticle corresponding to the good beam particle from the beam generator MCTruth object

Get the G4 MCParticle that matches a beam generator MCParticle (Thanks to James Pillow)

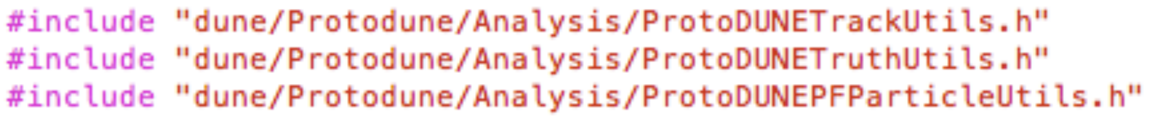
Usage Example

- I have included an example module on using these utilities
  - dune/ProtoDUNE/Analysis/UtilityExample_module.cc

- Run using the default .fcl file:
  - lar -c runProtoUtilExample.fcl <some_reco_file>.root –n <num_events>

- Tools built into a library ProtoDUNEAnaUtils that needs to be included in the CMakeLists.txt for your module

- Included the headers as usual

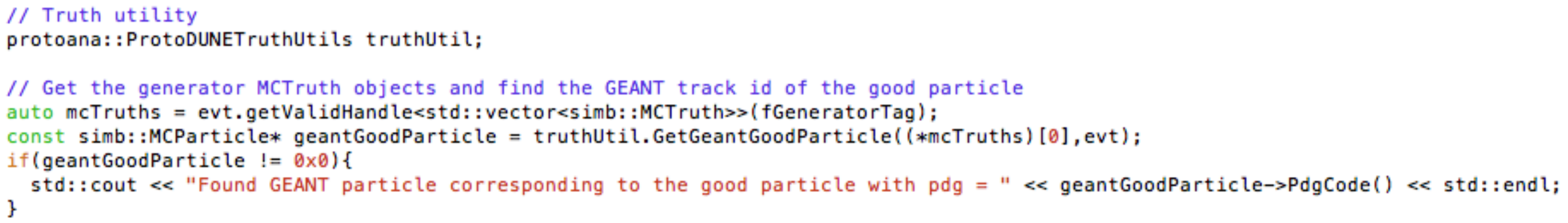


```
#include "dune/Protodune/Analysis/ProtoDUNETrackUtils.h"
#include "dune/Protodune/Analysis/ProtoDUNETruthUtils.h"
#include "dune/Protodune/Analysis/ProtoDUNEPFParticleUtils.h"
```

Usage Example

- The example first gets the G4 good particle

- The prerequisite for this is the list of generator MCTruth objects
  - There is only one so we can just use the zeroth element



```cpp
// Truth utility
protoana::ProtoDUNETruthUtils truthUtil;

// Get the generator MCTruth objects and find the GEANT track id of the good particle
auto mcTruths = evt.getValidHandle<std::vector<simb::MCTruth>>(fGeneratorTag);
const simb::MCParticle* geantGoodParticle = truthUtil.GetGeantGoodParticle((*mcTruths)[0],evt);
if(geantGoodParticle != 0x0){
  std::cout << "Found GEANT particle corresponding to the good particle with pdg = " << geantGoodParticle->PdgCode() << std::endl;
}
```

- If no match is found then a null pointer is returned, so be careful of that

Usage Example

- It then moves on to tracks

```
// Track utility
protoana::ProtoDUNETrackUtils trackUtil;

unsigned int nTracksWithTruth = 0;
unsigned int nTracksWithT0    = 0;
unsigned int nTracksWithTag   = 0;

// Get the reconstructed tracks
auto recoTracks = evt.getValidHandle<std::vector<recob::Track> >(fTrackerTag);

// Loop over the tracks
for(unsigned int t = 0; t < recoTracks->size(); ++t){

  // Match to truth
  const recob::Track thisTrack = (*recoTracks)[t];
  const simb::MCParticle *trueMatch = truthUtil.GetMCParticleFromRecoTrack(thisTrack,evt,fTrackerTag);
  bool hasTruth = (trueMatch != 0x0);

  // Check for T0
  std::vector<anab::T0> trackT0 = trackUtil.GetRecoTrackT0(thisTrack,evt,fTrackerTag);
  bool hasT0 = (trackT0.size() != 0);

  // Check for cosmic tag
  std::vector<anab::CosmicTag> trackCosmic = trackUtil.GetRecoTrackCosmicTag(thisTrack,evt,fTrackerTag);
  bool hasTag = (trackCosmic.size() != 0);

  if(hasTruth) ++nTracksWithTruth;
  if(hasT0)    ++nTracksWithT0;
  if(hasTag)   ++nTracksWithTag;

} // End loop over reconstructed tracks

std::cout << "Found " << recoTracks->size() << " reconstructed tracks:" << std::endl;
std::cout << " - " << nTracksWithTruth << " successfully associated to the truth information " << std::endl;
std::cout << " - " << nTracksWithT0    << " have a reconstructed T0" << std::endl;
std::cout << " - " << nTracksWithTag   << " have a cosmic tag" << std::endl;
```

Get all reconstructed tracks

Get the true particle that produced the track

Look for a T0 measurement for this track

Check to see if this track was tagged as a cosmic muon

Usage Example

- Finally, it looks at PFParticles

Get all reconstructed particles

```
// What about PFParticles?
protoana::ProtoDUNEPFParticleUtils pfpUtil;

auto recoParticles = evt.getValidHandle<std::vector<recob::PFParticle>>(fPFParticleTag);

unsigned int nParticlesPrimary   = 0;
unsigned int nParticlesWithT0    = 0;
unsigned int nParticlesWithTag   = 0;
for(unsigned int p = 0; p < recoParticles->size(); ++p){

  // Only consider primary particles here
  if(!(recoParticles->at(p).IsPrimary())) continue;

  ++nParticlesPrimary;

  // Do we have a T0?
  if(pfpUtil.GetPFParticleT0(recoParticles->at(p),evt,fPFParticleTag).size() != 0) ++nParticlesWithT0;
  if(pfpUtil.GetPFParticleCosmicTag(recoParticles->at(p),evt,fPFParticleTag).size() != 0) ++nParticlesWithTag;
}
std::cout << "Found " << nParticlesPrimary << " reconstructed primary particles:" << std::endl;
std::cout << " - " << nParticlesWithT0    << " have a reconstructed T0" << std::endl;
std::cout << " - " << nParticlesWithTag   << " have a cosmic tag" << std::endl;

// We can also build a slice map if we want to
std::map<unsigned int, std::vector<recob::PFParticle*>> sliceMap;
sliceMap = pfpUtil.GetPFParticleSliceMap(evt,fPFParticleTag);
unsigned int nMultiSlice = 0;
std::cout << "Found " << sliceMap.size() << " slices with PFParticles" << std::endl;
for(auto slice : sliceMap){
  if(slice.second.size() > 1) ++nMultiSlice;
}
std::cout << " - " << nMultiSlice << " have at least one primary PFParticle" << std::endl;

// Look for the beam particle slice and get the particles if we can
unsigned short beamSlice = pfpUtil.GetBeamSlice(evt,fPFParticleTag);
std::cout << "- Beam slice = " << beamSlice << std::endl;
if(beamSlice != 9999){
  std::vector<recob::PFParticle*> beamSlicePrimaries = pfpUtil.GetPFParticlesFromBeamSlice(evt,fPFParticleTag);
  std::cout << " - Found the beam slice! " << beamSlicePrimaries.size() << " beam particles in slice " << beamSlice << std::endl;
}
```

Look for a T0 measurement for this particle

Check to see if this particle was tagged as a cosmic muon

Build a map of primary particles and their reconstructed slice number in pandora

Get the particles in the beam slice

Summary

- In the process of producing analysis utilities to simplify “doing an analysis”

- Some functionality committed and working
  - New ideas and features welcome!
  - Requests welcome!

- Aim is to encapsulate some of the lengthier features of extracting useful information from art root files

- We also want to avoid each analyser having to write the same code to do what should be a “simple task”