

Restructuring anab::ParticleID

Kirsty Duffy and Adam Lister

Introduction

Public Member Functions

ParticleID ()	
ParticleID (int Pdg, int Ndf, double MinChi2, double DeltaChi2, double Chi2Proton, double Chi2Kaon, double Chi2Pion, double Chi2Muon, double MissingE, double MissingEavg, double PIDA)	
const int & Pdg () const	
const int & Ndf () const	
const double & MinChi2 () const	The current anab::ParticleID class is currently very restrictive.
const double & DeltaChi2 () const	
const double & Chi2Proton () const	
const double & Chi2Kaon () const	
const double & Chi2Pion () const	
const double & Chi2Muon () const	
const double & MissingE () const	
const double & MissingEavg () const	
const double & PIDA () const	
const geo::PlaneID & PlaneID () const	

The current **anab::ParticleID** class is currently **very** restrictive.

There are currently methods for the **Chi2** algorithm and **PIDA** but nothing else.

If you want to add a PID algorithm, this requires changing LArSoft each time!

We've recently been doing some PID work on MicroBooNE. In the process, we have developed a new organisation of the **anab::ParticleID** class which is **easily extendable**, and should be able to hold results for any potential algorithm we could think of.

Previously presented at LArSoft co-ordination meeting on 17th July 2018: have implemented suggestions and now hoping to implement new structure in LArSoft

Overview of new structure

The change comes down to replacing all variables with a new vector of **sParticleIDAlgScores** structs.

```
class ParticleID{
public:
    ParticleID();
    int fPdg;           ///< determined particle ID
    int fNdf;           ///< ndf for chi2 test
    double fMinChi2;    ///< Minimum reduced chi2
    double fDeltaChi2;  ///< difference between two lowest reduced chi2's
    double fChi2Proton; ///< reduced chi2 using proton template
    double fChi2Kaon;   ///< reduced chi2 using kaon template
    double fChi2Pion;   ///< reduced chi2 using pion template
    double fChi2Muon;   ///< reduced chi2 using muon template
    double fMissingE;   ///< missing energy from dead wires for contained particle
    double fMissingEavg; // < missing energy from dead wires using average dEdx
    double fPIDA;        ///< PID developed by Bruce Baller
    geo::PlaneID fPlaneID;

public:
    ParticleID(int Pdg,
               int Ndf,
               double MinChi2,
               double DeltaChi2,
               double Chi2Proton,
               double Chi2Kaon,
               double Chi2Pion,
               double Chi2Muon,
               double MissingE,
               double MissingEavg,
               double PIDA,
               geo::PlaneID planeID);

    friend std::ostream& operator << (std::ostream &o, ParticleID const& a);

    const int& Pdg() const;
    const int& Ndf() const;
    const double& MinChi2() const;
    const double& DeltaChi2() const;
    const double& Chi2Proton() const;
    const double& Chi2Kaon() const;
    const double& Chi2Pion() const;
    const double& Chi2Muon() const;
```



Current

```
class ParticleID{
public:
    ParticleID();
    std::vector<sParticleIDAlgScores> fParticleIDAlgScores; // < Vector of structs to hold outputs from generic PID algorithms
#ifndef __GCCXML__
public:
    ParticleID(std::vector<anab::sParticleIDAlgScores> &ParticleIDAlgScores);
    friend std::ostream& operator << (std::ostream &o, ParticleID const& a);
    const std::vector<anab::sParticleIDAlgScores> ParticleIDAlgScores() const;
#endif
};
```

Proposed

New Struct

The change comes down to replacing all variables with a new vector of **sParticleIDAlgScores** structs.

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...

Algorithm name, defined by experiment developers. Default: "AlgNameNotSet"

```
enum kVariableType {
    kGOF, // Goodness of Fit
    kLikelihood, // Likelihood
    kLogL, // Log-Likelihood
    kScore, // Generic Particle ID score
    kPIDA, // PIDA value
    kdEdxtruncmean, // dE/dx versus truncated mean
    kdQdxtruncmean, // dQ/dx versus truncated mean
    kTrackLength, // Track Length
    kEdeposited, // Deposited energy
    kEbyRange, // Energy by range
    kNotSet // Not set
};
```

Default: kNotSet

```
enum kTrackDir {
    kForward, // Direction track is reconstructed
    kBackward, // Opposite to reconstruction
    kNoDirection
};
```

Default: kNoDirection

Assumed PDG (for likelihoods, GOFs, etc.). Default: 0

Number of degrees of freedom, if applicable. Default: -9999

Value produced by algorithm. Default: -9999

Plane ID for the algorithm result: Default "00000"

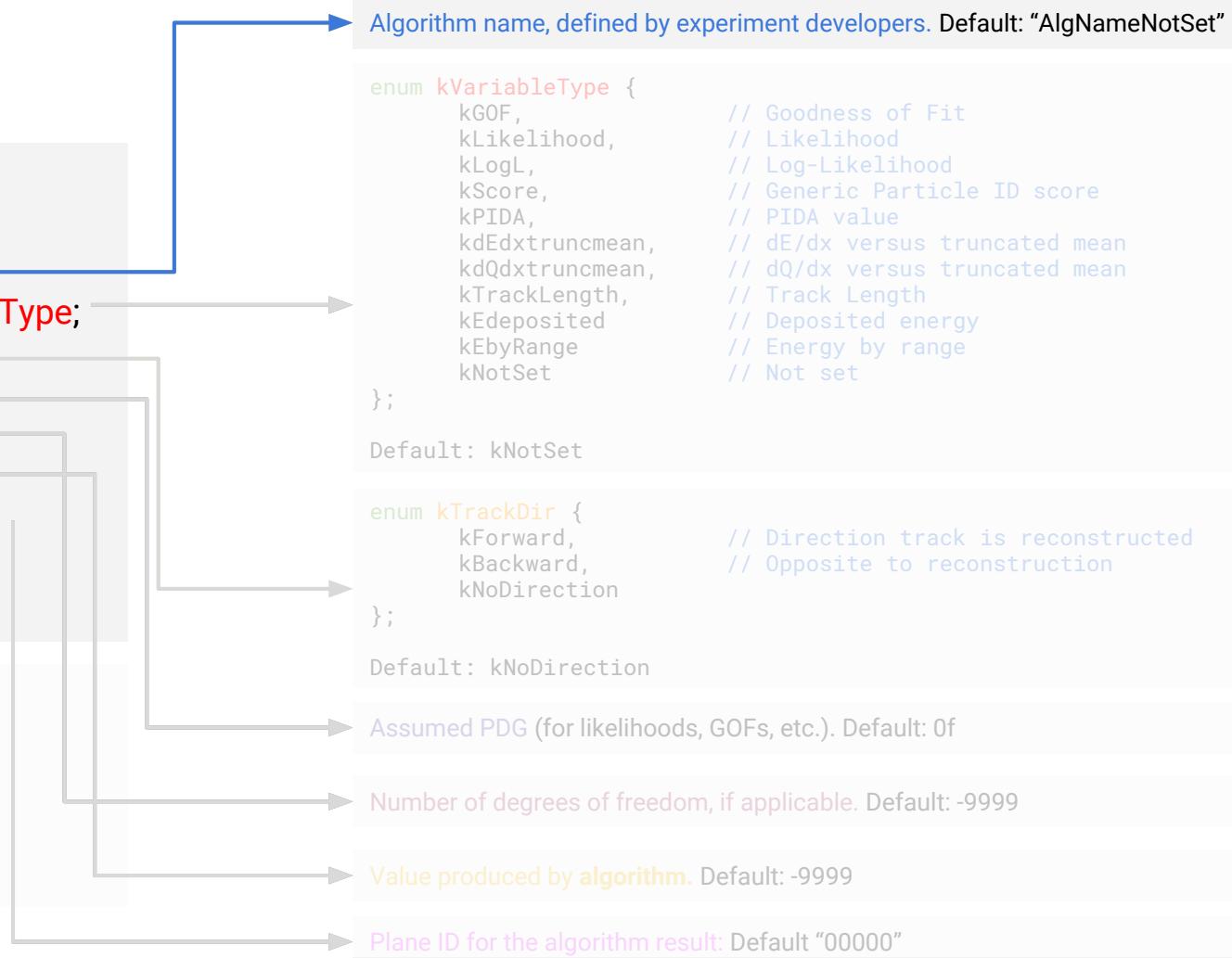
New Struct: fAlgName

fAlgName: this is just a string which can be used to identify an algorithm in the absence of anything else ("Chi2", "PIDA_mean", etc.). To be defined within experiments.

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...



New Struct: fVariableType

kVariableType: an enum which can be used to easily get at the type of variable you want.

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType; —————→
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...

```
Algorithm name, defined by experiment developers. Default: "AlgNameNotSet"
```

```
enum kVariableType {
    kGOF, // Goodness of Fit
    kLikelihood, // Likelihood
    kLogL, // Log-Likelihood
    kScore, // Generic Particle ID score
    kPIDA, // PIDA value
    kdEdxtruncmean, // dE/dx versus truncated mean
    kdQdxtruncmean, // dQ/dx versus truncated mean
    kTrackLength, // Track Length
    kEdeposited // Deposited energy
    kEbyRange // Energy by range
    kNotSet // Not set
};

Default: kNotSet
```

```
enum kTrackDir {
    kForward, // Direction track is reconstructed
    kBackward, // Opposite to reconstruction
    kNoDirection
};

Default: kNoDirection
```

```
Assumed PDG (for likelihoods, GOFs, etc.). Default: 0f
```

```
Number of degrees of freedom, if applicable. Default: -9999
```

```
Value produced by algorithm. Default: -9999
```

```
Plane ID for the algorithm result: Default "00000"
```

New Struct: fTrackDir

kTrackDir: enum to represent whether ParticleID score is calculated assuming track goes forwards or backwards (with respect to reconstructed direction)

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...

```
Algorithm name, defined by experiment developers. Default: "AlgNameNotSet"
```

```
enum kVariableType {
    kGOF, // Goodness of Fit
    kLikelihood, // Likelihood
    kLogL, // Log-Likelihood
    kScore, // Generic Particle ID score
    kPIDA, // PIDA value
    kdEdxtruncmean, // dE/dx versus truncated mean
    kdQdxtruncmean, // dQ/dx versus truncated mean
    kTrackLength, // Track Length
    kEdeposited, // Deposited energy
    kEbyRange, // Energy by range
    kNotSet // Not set
};
```

Default: kNotSet

```
enum kTrackDir {
    kForward, // Direction track is reconstructed
    kBackward, // Opposite to reconstruction
    kNoDirection
};
```

Default: kNoDirection

```
Assumed PDG (for likelihoods, GOFs, etc.). Default: 0f
```

```
Number of degrees of freedom, if applicable. Default: -9999
```

```
Value produced by algorithm. Default: -9999
```

```
Plane ID for the algorithm result: Default "00000"
```

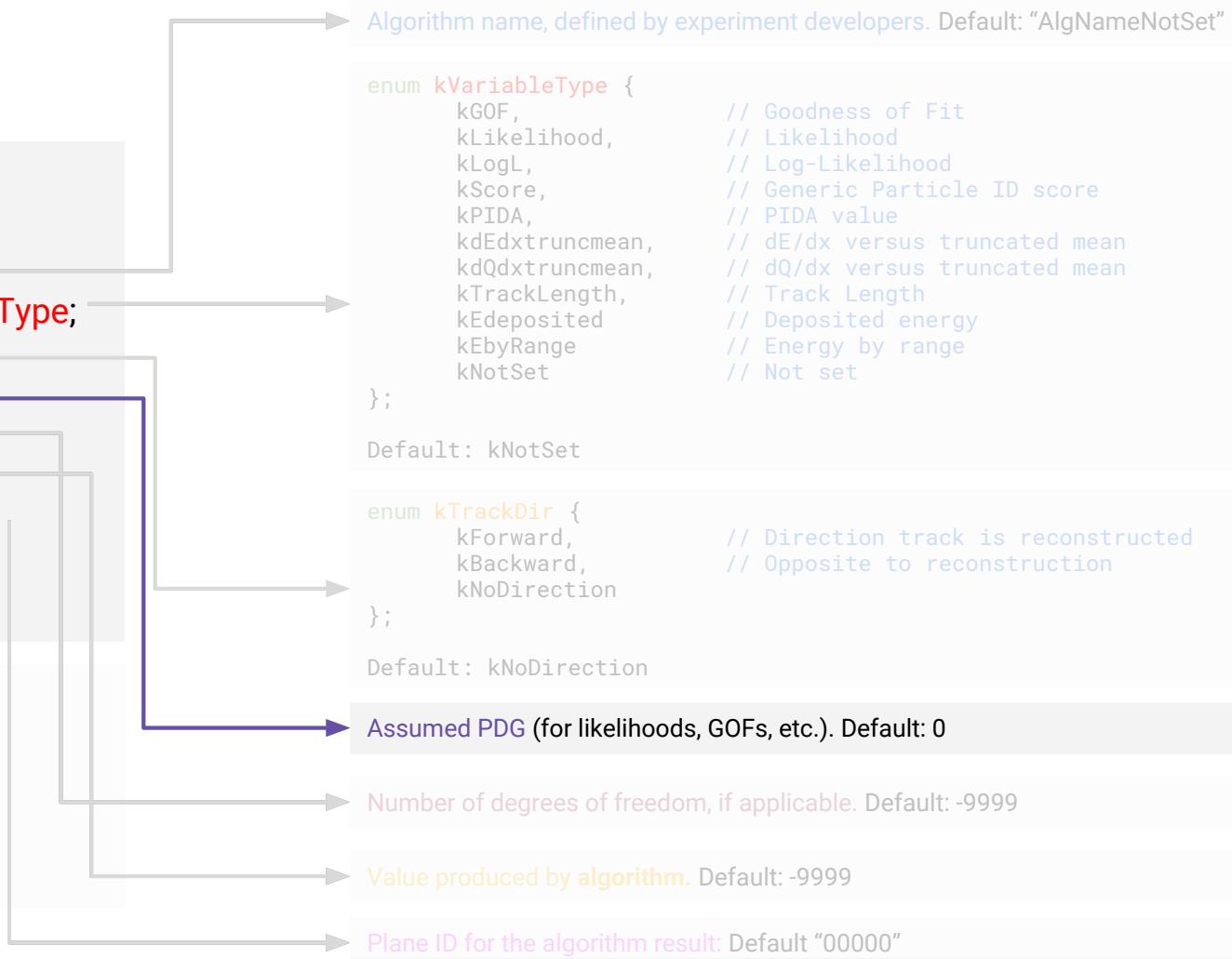
New Struct: fAssumedPdg

fAssumedPdg: This is used for algorithms where an assumption about the particle species is made (e.g. Chi2 with respect to the Muon hypothesis).

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...



New Struct: fNdf

fNdf: number of degrees of freedom assumed by an algorithm (e.g. Chi2)

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...

Algorithm name, defined by experiment developers. Default: "AlgNameNotSet"

```
enum kVariableType {
    kGOF, // Goodness of Fit
    kLikelihood, // Likelihood
    kLogL, // Log-Likelihood
    kScore, // Generic Particle ID score
    kPIDA, // PIDA value
    kdEdxtruncmean, // dE/dx versus truncated mean
    kdQdxtruncmean, // dQ/dx versus truncated mean
    kTrackLength, // Track Length
    kEdeposited // Deposited energy
    kEbyRange // Energy by range
    kNotSet // Not set
};
```

Default: kNotSet

```
enum kTrackDir {
    kForward, // Direction track is reconstructed
    kBackward, // Opposite to reconstruction
    kNoDirection
};
```

Default: kNoDirection

Assumed PDG (for likelihoods, GOFs, etc.). Default: 0f

Number of degrees of freedom, if applicable. Default: -9999

Value produced by algorithm. Default: -9999

Plane ID for the algorithm result: Default "00000"

New Struct: fValue

fValue: This contains the value or score from a list of algorithms which feed the ParticleID producer module. These algorithms can be general use or experiment specific!

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```

This is fed by c++ algorithms:

- Chi2
- PIDA
- ...

Algorithm name, defined by experiment developers. Default: "AlgNameNotSet"

```
enum kVariableType {
    kGOF, // Goodness of Fit
    kLikelihood, // Likelihood
    kLogL, // Log-Likelihood
    kScore, // Generic Particle ID score
    kPIDA, // PIDA value
    kdEdxtruncmean, // dE/dx versus truncated mean
    kdQdxtruncmean, // dQ/dx versus truncated mean
    kTrackLength, // Track Length
    kEdeposited, // Deposited energy
    kEbyRange, // Energy by range
    kNotSet // Not set
};
```

Default: kNotSet

```
enum kTrackDir {
    kForward, // Direction track is reconstructed
    kBackward, // Opposite to reconstruction
    kNoDirection
};
```

Default: kNoDirection

Assumed PDG (for likelihoods, GOFs, etc.). Default: 0f

Number of degrees of freedom, if applicable. Default: -9999

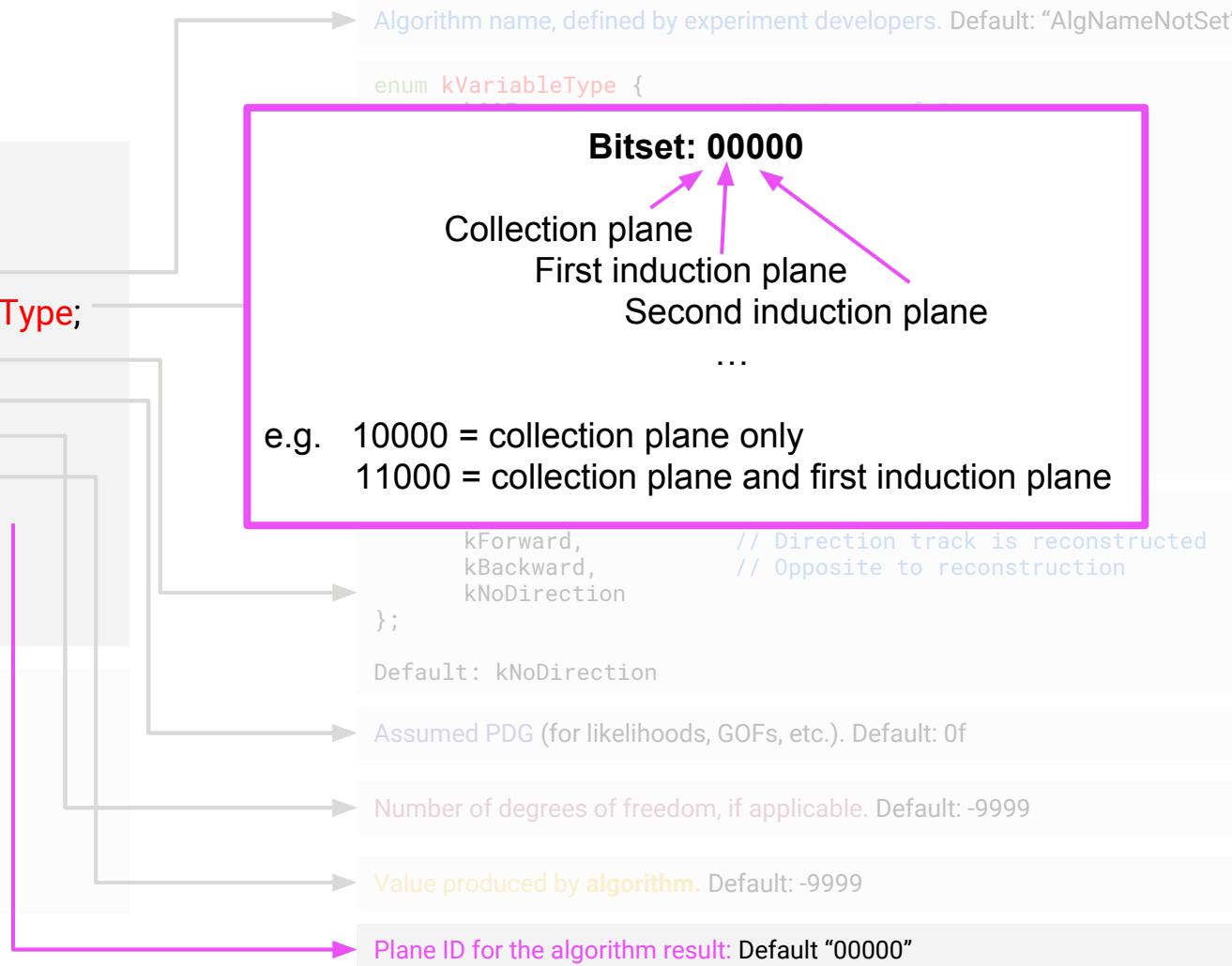
Value produced by algorithm. Default: -9999

Plane ID for the algorithm result: Default "00000"

New Struct: fPlaneID

fPlaneID: Many algorithms make use of charge information from a single plane. This allows you to know which! 5-part bitset allows any combination of up to 5 planes (allows for combinations and larger TPCs in the future)

```
struct sParticleIDAlgScores {
    std::string fAlgName;
    kVariableType fVariableType;
    kTrackDir fTrackDir;
    int fAssumedPdg;
    Int fNdf;
    float fValue;
    std::bitset<5> fPlaneID;
}
```



This is fed by c++ algorithms:

- Chi2
- PIDA
- ...

Breaking Change

Moving to this structure for `anab::ParticleID` is a breaking change

Affects the following modules/algorithms in the repository:

- Larana:
 - Chi2ParticleID_module.cc
 - Chi2PIDAlg.h
 - Chi2PIDAlg.cxx
- Lardataobj:
 - ParticleID.h
 - ParticleID.cxx
 - ParticleID_VariableTypeEnums.h
- Larreco:
 - KalmanFilterFitTrackMaker_tool.cc
 - KalmanFilterFinalTrackFitter_module.cc
- Ubana:
 - AnalysisTree_module.cc
 - XYZvalidatoin_module.cc
 - Diffusion_module.cc
 - MuonCandidateFinder.cxx
 - MuonCandidateFinder.h
 - UBXSec_module.cc
 - [Added folder] ParticleID
- Argoneutcode
 - AnalysisTreeT962_module.cc
- Dunetpc
 - AnalysisTree_module.cc
 - ProtoDUNEAnalCosmicTree_module.cc
 - ProtoDUNEAnalTree_module.cc
 - ProtoDUNETrackUtils.cxx
 - AnaRootParser_module.cc
- Icaruscode:
 - AnalysisTree_module.cc
- Lariatsoft:
 - AnaTreeT1034_module.cc
 - MCAnalysis_module.cc
 - MichelWfmReco_module.cc
 - AnaTree_module.cc
- Sbndcode:
 - AnalysisTree_module.cc
- Lareventdisplay

Breaking Change

Moving to this structure for `anab::ParticleID` is a breaking change

Affects the following modules/algorithms in the repository:

- **Larana:** [feature/alister1_chi2_pidclassupgrade](#)
 - `Chi2ParticleID_module.cc`
 - `Chi2PIDAlg.h`
 - `Chi2PIDAlg.cxx`
- **Lardataobj:** [feature/alister1_pid_ioread](#)
 - `ParticleID.h`
 - `ParticleID.cxx`
 - `ParticleID_VariableTypeEnums.h`
- **Larreco:** [feature/alister1_pidfix](#)
 - `KalmanFilterFitTrackMaker_tool.cc`
 - `KalmanFilterFinalTrackFitter_module.cc`
- **Ubana:** [feature/alister1_chi2_pidclassupgrade](#)
 - `AnalysisTree_module.cc`
 - `XYZvalidatoin_module.cc`
 - `Diffusion_module.cc`
 - `MuonCandidateFinder.cxx`
 - `MuonCandidateFinder.h`
 - `UBXSec_module.cc`
 - [Added folder] `ParticleID`

Changes already made on feature branches in

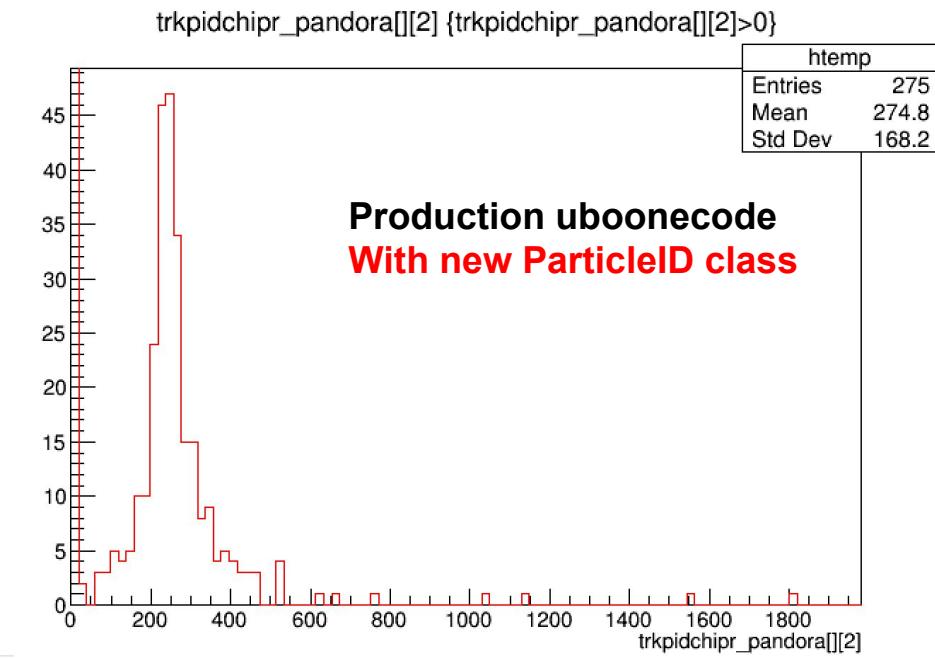
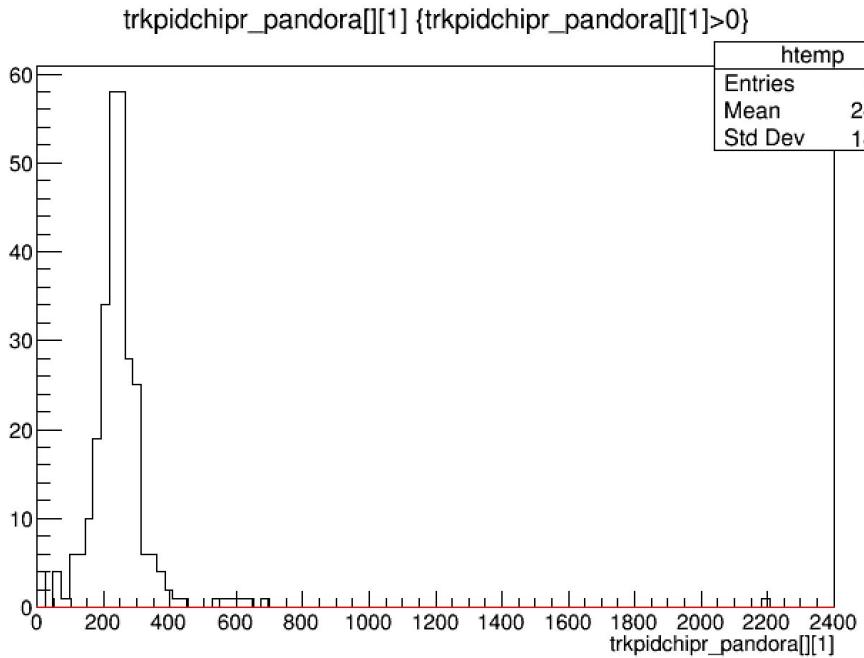
- **Larana**
- **Lardataobj**
- **Larreco** (remove PID from outdated Kalman fitting code)
- **Lareventdisplay** (remove PID from event display)
- **Ubana**

- **Lareventdisplay**
[feature/alister1_removepidfromevd](#)

Breaking Change

Added ioread rule in lardataobj/AnalysisBase/classes_def.xml to ensure that old files can still be read by new class

Validation work still ongoing but initial tests using uboonecode AnalysisTree_module.cc seem to be working correctly for collection plane (remaining issues with induction planes being investigated)



Breaking Change

A change to the ParticleID class will probably require changes to a lot of analysis code

Instructions for how to adapt code provided in backup slides for this presentation

Summary

We think that this reorganisation of the code is much more flexible: it allows for new PID algorithms and can be used for shower PID in addition to track PID.

The main downside is that this relies on the analyser knowing what's in the struct, and so it requires **good experiment-specific documentation**.

Thank you all for feedback at previous meetings about implementation!

Are there more problems that we haven't thought about? Is this something that we can implement now?

How to change your code

Because the PlanID is stored as a five-part bitset (to allow for multi-plane PID algorithms in the future), some PlanID bitset helper functions have been added to uboonecode to convert the plane ID to an int (following the standard convention: plane 2 is the collection Y plane, plane 1 is the induction V plane, and plane 0 is the induction U plane).

These helper functions are only valid for algorithms that use a single plane; they are not yet extended to multi-plane algorithms.

The MicroBooNE helper functions are contained in the following header file:
`#include "ubana/ParticleID/Algorithms/uB_PlaneIDBitsetHelperFunctions.h"`

The helper function used in this example code is `UBPID::uB_getSinglePlane` -- note that **this will not work out of the box for other experiments!** The helper functions need to be experiment-specific, so an equivalent header file will need to be added for each experiment in the experiment's own repository

How to change your code

First, get the anab::ParticleID objects by associations with tracks:

```
art::FindManyP trackPIDAssn(trackHandle, evt, fPIDLabel);
if (!trackPIDAssn.isValid()){
    std::cout << "[ParticleIDValidation] trackPIDAssn.isValid() == false. Skipping
track." << std::endl;
    continue;
}
```

```
std::vector<art::Ptr<anab::ParticleID>> trackPID = trackPIDAssn.at(track->ID());
if (trackPID.size() == 0){
    std::cout << "[ParticleID] No track-PID association found for trackID " <<
track->ID() << ". Skipping track." << std::endl;
    continue;
```

How to change your code

Next, get the vector of `anab::sParticleIDAlgScores` (the vector of Particle ID scores from all the various algorithms) for your track:

```
std::vector AlgScoresVec = trackPID.at(0)->ParticleIDAlgScores();
```

How to change your code

Now loop through this vector, and check fAlgName, fVariableType, fTrackDir, fAssumedPdg, and fPlaneID to find the result of the algorithm you want, on the plane you want. The algorithm score will be stored in fValue. In this example, we look for the result of the Chi2 algorithm assuming a forward-going muon, and store it in the variable chi2_mu.

```
double chi2_mu = -999;

// Loop through AlgScoresVec and find the variables we want
for (size_t i_algscore=0; i_algscore<AlgScoresVec.size(); i_algscore++){
    anab::sParticleIDAlgScores AlgScore = AlgScoresVec.at(i_algscore);
    int planeid = UBPID::uB_getSinglePlane(AlgScore.fPlaneID);

    if (planeid != 2){
        std::cout << "[ParticleID] Not using information for plane " << planeid <<
        "(using plane 2 calorimetry only)" << std::endl;
        continue;
    }
```

How to change your code

...

```
if (AlgScore.fAlgName == "Chi2"){
    if (anab::kVariableType(AlgScore.fVariableType) == anab::kGOF &&
anab::kTrackDir(AlgScore.fTrackDir) == anab::kForward){
        if (TMath::Abs(AlgScore.fAssumedPdg) == 13)
            chi2_mu = AlgScore.fValue;
    }
}
```

Another example: uboonecode AnalysisTree_module.cc

What the
code looks
like now

```
// find particle ID info
art::FindMany<anab::ParticleID> fmpid(trackListHandle[iTracker], evt, fParticleIDModuleLabel[iTracker]);
if(fmpid.isValid()) {
    std::vector<const anab::ParticleID*> pids = fmpid.at(iTrk);
    //if(pids.size() > 1) {
        //mf::LogError("AnalysisTree:limits")
        //<< "the " << fTrackModuleLabel[iTracker] << " track #" << iTrk
        //<< " has " << pids.size()
        //<< " set of ParticleID variables. Only one stored in the tree";
    //}
    for (size_t ipid = 0; ipid < pids.size(); ++ipid){
        if (!pids[ipid]->PlaneID().isValid) continue;
        int planenum = pids[ipid]->PlaneID().Plane;
        if (planenum<0||planenum>2) continue;
        TrackerData.trkpidpdg[iTrk][planenum] = pids[ipid]->Pdg();
        TrackerData.trkpidchi[iTrk][planenum] = pids[ipid]->MinChi2();
        TrackerData.trkpidchipr[iTrk][planenum] = pids[ipid]->Chi2Proton();
        TrackerData.trkpidchika[iTrk][planenum] = pids[ipid]->Chi2Kaon();
        TrackerData.trkpidchipi[iTrk][planenum] = pids[ipid]->Chi2Pion();
        TrackerData.trkpidchimu[iTrk][planenum] = pids[ipid]->Chi2Muon();
        TrackerData.trkpidpida[iTrk][planenum] = pids[ipid]->PIDA();
    }
} // fmpid.isValid()
```

Another example: uboonecode AnalysisTree_module.cc

Edits to
work with
new class

```
#include "lardataobj/AnalysisBase/PlaneIDBitsetHelperFunctions.h"

// find particle ID info
art::FindMany<anab::ParticleID> fmpid(trackListHandle[iTracker], evt, fParticleIDModuleLabel[iTracker]);
if(fmpid.isValid()) {
    std::vector<const anab::ParticleID*> pids = fmpid.at(iTrk);
    if (pids.size() == 0){
        mf::LogError("AnalysisTree:limits")
            << "No track-PID association found for " << fTrackModuleLabel[iTracker]
            << " track " << iTrk << ". Not saving particleID information.";
    }
    // Set dummy values
    double pidpdg[3] = {-1,-1,-1};
    double pidchi[3] = {99999.,99999.,99999.};
    for (size_t ipid=0; ipid<pids.size(); ipid++){
        std::vector<anab::sParticleIDAlgScores> AlgScoresVec = pids[ipid]->ParticleIDAlgScores();

        // Loop though AlgScoresVec and find the variables we want
        for (size_t i_algscore=0; i_algscore<AlgScoresVec.size(); i_algscore++){
            anab::sParticleIDAlgScores AlgScore = AlgScoresVec.at(i_algscore);
            ....
        }
    }
}
```

Another example: uboonecode AnalysisTree_module.cc

.....

Edits to work with new class

```
/* std::cout << "\n ParticleIDAlg " << AlgScore.fAlgName
<< "\n -- Variable type: " << AlgScore.fVariableType
<< "\n -- Track direction: " << AlgScore.fTrackDir
<< "\n -- Assuming PDG: " << AlgScore.fAssumedPdg
<< "\n -- Number of degrees of freedom: " << AlgScore.fNdf
<< "\n -- Value: " << AlgScore.fValue
<< "\n -- Using planeID: " << BitsetHelper::getSinglePlane(AlgScore.fPlaneID) << std::endl;*/

int planenum = UBPID::uB_getSinglePlane(AlgScore.fPlaneID);
if (planenum<0 || planenum>2) continue;

if (AlgScore.fAlgName == "Chi2"){
if (TMath::Abs(AlgScore.fAssumedPdg) == 13){ // chi2mu
TrackerData.trkpidchimu[iTrk][planenum] = AlgScore.fValue;
if (AlgScore.fValue<pidchi[planenum]){
pidchi[planenum] = AlgScore.fValue;
pidpdg[planenum] = TMath::Abs(AlgScore.fAssumedPdg);
}
}
else if (TMath::Abs(AlgScore.fAssumedPdg) == 2212){ // chi2pr
TrackerData.trkpidchipr[iTrk][planenum] = AlgScore.fValue;
if (AlgScore.fValue<pidchi[planenum]){
pidchi[planenum] = AlgScore.fValue;
pidpdg[planenum] = TMath::Abs(AlgScore.fAssumedPdg);
}
}
}
....
```

Another example: uboonecode AnalysisTree_module.cc

```
.....
else if (TMath::Abs(AlgScore.fAssumedPdg) == 211){ // chi2pi
    TrackerData.trkpidchipi[iTrk][planenum] = AlgScore.fValue;
    if (AlgScore.fValue<pidchi[planenum]){
        pidchi[planenum] = AlgScore.fValue;
        pidpdg[planenum] = TMath::Abs(AlgScore.fAssumedPdg);
    }
}
else if (TMath::Abs(AlgScore.fAssumedPdg) == 321){ // chi2ka
    TrackerData.trkpidchika[iTrk][planenum] = AlgScore.fValue;
    if (AlgScore.fValue<pidchi[planenum]){
        pidchi[planenum] = AlgScore.fValue;
        pidpdg[planenum] = TMath::Abs(AlgScore.fAssumedPdg);
    }
}
else if (AlgScore.fVariableType==anab::kPIDA){
    TrackerData.trkpidpida[iTrk][planenum] = AlgScore.fValue;
}
} // end loop though AlgScoresVec
} // end loop over pid[ipid]

// Finally, set min chi2
for (size_t planenum=0; planenum<3; planenum++){
    TrackerData.trkpidchi[iTrk][planenum] = pidchi[planenum];
    TrackerData.trkpidpdg[iTrk][planenum] = pidpdg[planenum];
}
} // fmpid.isValid()
```

Edits to work with new class