Proposed ParticleID Update

Larsoft Coordination Meeting
Aug. 24, 2021

H. Greenlee

Outline

- Historical overview.
- Proposed updates to ParticleID.
- Merge branches.
- Non-backward-compatibility issue, and possible alternatives and solutions.

Historical Overview

- This talk is a follow up to a proposal that was originally presented in the Jan. 15, 2019 larsoft coordintion meeting by Kirsty Duffy and Adam Lister.
  - Link to talk.
- The proposal involved changing data product class anab::ParticleID in lardataobj.
  - Current version of ParticleID support two particle id. algorithms, namely, PIDA and chisquare.
  - Updated version of ParticleID would support an arbitrary collection of particle id algorithms.

Historical Overview (cont.)

- My recollection from that time is that the proposal was agreed to in principle, with the caveat that an i/o rule should be developed to allow reading the old version of the data product by the new code.
  - At the time, and even up to now, no such o/o rule has been able to be developed, possibly due to root bugs.
  - MicroBooNE decided to go ahead and update our MCC9 branch anyway, knowing the code would not be able to read old data.
  - The plan to update ParticleID data product on the develop branch was shelved because of the non-backward-compatibility issue.
- Now MicroBooNE wants to merge our MCC9 updates into develop branch, which requires facing the ParticleID issue.

Proposed Changes

- The next 13 slides are copied from a talk by Kirsty Duffy and Adam Lister in a MicroBooNE reconstruction meeting from Feb. 20, 2019.
  - Slides closely follow what was presented in the Jan. 15, 2019 larsoft coordination meeting.

Kirsty Duffy
Adam Lister

Introduction

Public Member Functions

	ParticleID ()
	ParticleID (int Pdg, int Ndf, double MinChi2, double DeltaChi2, double Chi2Proton, doubl
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
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!

In the process of the recent PID work on MicroBooNE, 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.

Presented at LArSoft co-ordination meeting on 15th January 2019: breaking change approved.

Kirsty Duffy
Adam Lister

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 par
ticle
   double fMissingEavg;     ///< missing energy from dead wires using average dEd
x
   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

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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 mask for the algorithm result: Default "0000000"

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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"

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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"

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

)

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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"

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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"

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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"

Kirsty Duffy
Adam Lister

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<8>  fPlaneMask;
}
```

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"

Kirsty Duffy
Adam Lister

New Struct: fPlaneID

fPlaneID: Many algorithms make use of charge information from a single plane. This allows you to know which! 8-part bitset allows any combination of up to 8 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<8>  fPlaneMask;
}
```

Bitset: 00000000

Collection plane
First induction plane
Second induction plane
…

e.g. 10000000 = collection plane only
11000000 = collection plane and first induction plane

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- …

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

enum kVariableType {

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 mask for the algorithm result: Default "00000000"

Kirsty Duffy
Adam Lister

Breaking Change

Moving to this structure for anab::ParticleID is a breaking change (but approved by LArSoft)

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

Kirsty Duffy
Adam Lister

13

Integration status

- Feature branches with changes for uboone/LArSoft 
  - Larana
  - Lardataobj
  - Larreco
  - Ubana
  - Lareventdisplay
- Don’t break anyone else’s code: feature branches in other repositories 
  - Argoneutcode
  - Dunetpc
  - Icaruscode
  - Lariatsoft
  - Sbndcode
- Don’t break old files: ioread rule 

Kirsty Duffy
Adam Lister

Ioread rule

Root has the ability to let you tell it how to interpret data products and read them out in a different format: ioread rules

Defined in classes_def.xml file inside lardataobj/AnalysisBase/

Means that we can read old files (generated with old anab::ParticleID data product) even when we have loaded the new software

But…

Seems to be a problem with reading in the geo::PlaneID object. Output is the default PlaneID every time, even when we can see that there are non-default planes being set

Phillippe Canal from SCD is helping us look into this, but it is a **roadblock for integrating into larsoft develop** - can't make it so other experiments are unable to use their data/MC files!

However, this doesn't stop us integrating into MCC9 as long as we only read files that we generated in MCC9: **that is the plan for now**

End of Kirsty's and Adam's slides

Recent Work

- Following three slides are Kirsty's summary of the current situation, copied from an email, after taking another look recently.
  - Situation hasn't changed, but Kirsty's explanation contains more details.

Kirsty's Explanation of ParticleID Issue – Background Information

- We made a number of changes to the anab::ParticleID data class for MicroBooNE — changes summarized here: docdb 21274
- The intent is to make the class usable for new/different particle ID algorithms (the old class was coded only to support chi2 and PIDA algorithms). It also now supports multi-plane algorithms, by moving to a bitset for the plane ID (instead of the geo::PlaneID, which can only ever identify one single plane).
- This was presented at the LArSoft coordination meeting in July 2018 (https://indico.fnal.gov/event/17640//), although note that the MicroBooNE DocDB slides are more up-to-date so should be used as the reference — some changes were made since the LArSoft coordination meeting based on advice we were given there)
- My impression of the LArSoft coordination meeting outcome was that breaking changes are undesirable (but not necessarily a dealbreaker if they can't be avoided). They also asked us to make sure that the change wouldn't impact other experiments, or at least to minimize the impact. In this case all experiments' codes will have to change (we provided feature branches in which we changed the code for them where it was available in the repo, but of course most analysers' code is not committed and would have to be changed). That was accepted, and the code we changed was tested by the experiments to confirm that it gives the same results as previous code.
- We also want to make sure that experiments can still read files that were produced before the change. That can be done using I/O read rules in root: in the classes_def.xml file we can define rules for reading in the old file and converting it into the new format. I've attached the I/O read rule we wrote below as an example. In theory, that should make old files readable with the new code.

Kirsty's Explanation of ParticleID Issue – The Problem

- The problem is that the I/O read rule does not seem to be able to convert the geo::PlaneID to a bitset. The resulting file has an invalid planeID bitset.
- I was put in touch with Philippe Canal about this, and in 2019 this is what he said “The only problem is with fPlaneID and the fact that it is split. This is a known problem whose solution will take sometime. One work-around is to keep in the new code a fPlaneID data member that is used “solely” as a staging area for the incoming data”.
- I interpreted Philippe’s comment to mean that for some reason we can’t open and read the geo::PlaneID during the I/O read rule. So I took his advice (in an updated version of the code, in November 2020) and created a new geo::PlaneID data member in the new anab::ParticleID class. The idea is that the I/O read rule would no longer try to read the geo::PlaneID, just copy it to the new class (that’s what happens in the I/O rule I copied below). Then I also wrote some internal code to convert it to a bitset at a different stage.
- Unfortunately, this fix didn’t work. The copied PlaneID I get is still invalid. I can’t confirm whether that’s because an invalid PlaneID was copied in, or because the PlaneID was never read by the I/O rule (because both cases would give the same invalid PlaneID that we see)
- So the ultimate problem is that, when reading old files, we can’t get the information about the plane used for the PID algorithm. Although note that all other PID information is copied correctly and accessible.

Kirsty's Explanation of ParticleID Issue – Possible Solutions

- I think the easiest solution would be to discuss with LArSoft and the other experiments whether this is actually a problem. If the other experiments only ever use one plane for PID, then they don't actually need the PlaneID information and so this might not be a dealbreaker. They would have to rewrite their code to not check whether the PlaneID is valid, but that's about it.
- While trying to find the LArSoft coordination meeting to link above, I stumbled upon this presentation: https://indico.fnal.gov/event/20287/contributions/56915/attachments/35612/43437/larsoft-coordination-meeting-2019-03-26.pdf
  It gives a plan to update to root v6.16/00 and art v3.02 "to fix an iorule problem observed by MicroBooNE that breaks anab::ParticleID". I don't *think* this fixes it (because when I did my second round of attempts in November 2020 I based it on larsoft v09_09_01, which uses root v6_18_04d), but we shouldn't discount me making a mistake so it might be worth trying again.

Merge Branches

- Kirsty's latest larsoft merge branches (based off larsoft v09_09_01).
  - lardataobj: feature/kduffy_merge_uB_PID
  - lareventdisplay: feature/kduffy_merge_uB_PID
  - larana: feature/kduffy_merge_uB_PID
  - larreco: feature/kduffy_merge_uB_PID
  - ubana: feature/kduffy_merge_uB_PID
- Kirsty's experiment merge branches (based off larsoft v08_00_00).
  - argoneutcode: feature/kduffy_updatePIDdataprod
  - dunetpc: feature/kduffy_updatePIDdataprod
  - icaruscode: feature/kduffy_updatePIDdataprod
  - lariatsoft: feature/kduffy_updatePIDdataprod
  - sbndcode: feature/kduffy_updatePIDdataprod

Alternative Merge Branches

- Herb's MCC10 larsoft merge branches.
  - lardataobj: greenlee_mcc9_pid
  - larreco: greenlee_mcc9_pid
  - larana: greenlee_mcc9_pid
  - lareventdisplay: greenlee_mcc9_pid

What We Are Asking of Larsoft

- We would like larsoft experiments to consider whether they would consider accepting this ParticleID update even though it is not backward compatible with old format data.
  - Or only backward compatible for one readout plane.
- If above bullet is not acceptable, does anyone have any ideas, or is willing to help, developing a working ioread rule?
- Final fallback position is to fork ParticleID (following slides).

Forking Data Product ParticleID Overview

- Forking data product ParticleID would involve adding a new data product class in package lardataobj, which would be a copy of the MicroBooNE MCC9 version of ParticleID, but with a different name (suggestion: ParticleMID).
- It will then be necessary to assess what files in larsoft depend on ParticleID, and which ones need updating or forking (details see next slide).
- Additionally, it will be necessary to assess find all files in uboone suite and update all of them to depend on the new ParticleID class.
- Experiments would not be obligated to update their code unless they wanted to take advantage of forked version of ParticleID class.

Forking Data Product ParticleID Larsoft Dependencies

- Alternative version of ParticleID (lardataobj).
- Algorithm class Chi2PIDAlg (larana).
  - Factory class for ParticleID. Would need to be taught now to make either version of ParticleID (non-breaking interface expansion).
  - MicroBooNE uses this class (different in MCC9 vs. develop).
- Art module Chi2ParticleID.
  - Companion of ParticleID algorithm class.
  - MicroBooNE doesn't use it (we have a MicroBooNE-specific module).
- Larsoft event disiplay.
- Kalman filter track fitter module + tool (larreco).
  - KalmanFilterFinalTrackFitter module.
  - KalmanFilterFitTrackMaker tool.
  - MicroBooNE uses, but configured without ParticleID.

Summary

- We (MicroBooNE) are asking experiments whether they are interested in adopting MicroBooNE's MCC9 version of ParticleID, which supports multiple particle id algorithms.
  - Either as non-backward-compatible breaking change.
  - Or forking ParticleID and adding a second ParticleID data product.