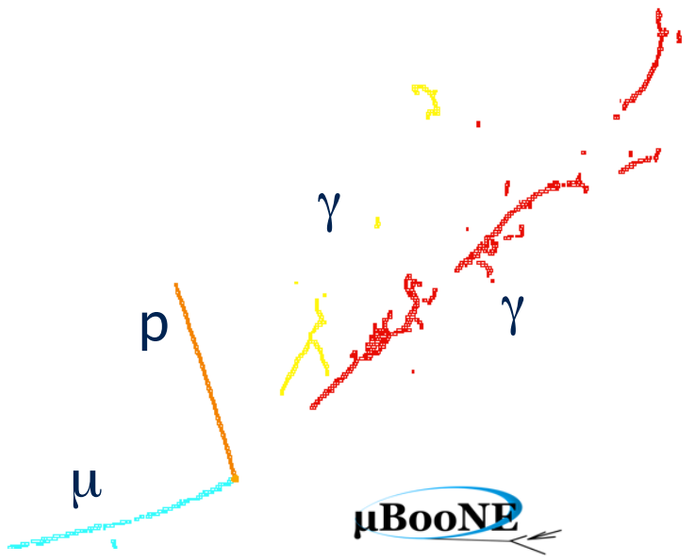


Pandora Talk 2: ClientApp

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

larpandora

- The client application for Pandora in LArSoft is “larpandora”.
- Its default git remote repository is hosted on Fermilab Redmine.

https://cdcvs.fnal.gov/redmine/projects/larpandora/repository

MicroBooNE Pandora Workshop and Working Meeting (11-14 July 2016) | / - Repository - LArPandora - Fermilab Redmine

Home Projects Help Sign in

LArSoft » LArPandora

Search:

Overview Activity Gantt Documents Wiki Files Repository

root @ develop

Statistics | Branch: develop | Tag: | Revision:

Name	Size	Revision	Age	Author	Comment
larpandora		6c044827	about 1 month	Lynn Garren	using larpandoracontent v02_07_03
ups		0c0b60ba	5 days	Lynn Garren	larpandora v05_09_06 for larsoft v05_12_01
CMakeLists.txt	2.55 KB	6c044827	about 1 month	Lynn Garren	using larpandoracontent v02_07_03

Latest revisions

#	Date	Author	Comment
0c0b60ba	06/08/2016 02:46 PM	Lynn Garren	larpandora v05_09_06 for larsoft v05_12_01
f6255577	05/25/2016 07:24 PM	Lynn Garren	larpandora v05_09_05 for larsoft v05_12_00
137138a1	05/11/2016 02:34 PM	Lynn Garren	larpandora v05_09_04 for larsoft v05_11_01
6c044827	05/11/2016 01:26 PM	Lynn Garren	using larpandoracontent v02_07_03
6772841e	05/04/2016 10:32 AM	Lynn Garren	fix the logic
e408297c	05/04/2016 09:33 AM	Lynn Garren	larpandoracontent qualifiers
bfe563f2	05/03/2016 04:42 PM	Lynn Garren	larpandora v05_09_03 for larsoft v05_11_00
6fb39681	05/03/2016 03:55 PM	Lynn Garren	allow for building larpandoracontent with mrb
d8de6fa0	04/27/2016 06:15 PM	Lynn Garren	larpandora v05_09_02 for larsoft v05_10_00
cdc573bd	04/19/2016 06:18 PM	Lynn Garren	larpandora v05_09_01 for larsoft v05_09_01

View differences

View all revisions | View revisions

larpandora

- In principle, the client application should be rather simple: create Pandora instance(s), register algorithms, provide Pandora Settings XML file and handle event input/output.
- In reality, tends to get rather complicated, but this just vindicates decision to separate algorithm implementation from steps needed to access/format the inputs as desired.

```
/**
 *  @brief  ILArPandora class
 */
class ILArPandora : public art::EDProducer
```

Implement as an art::EDProducer

Relevant callbacks:

```
void beginJob();
```

```
void produce(art::Event &evt);
```

Algorithm Pseudocode description of a client application for LAr TPC event reconstruction in a single drift volume

```
 1: procedure MAIN
 2:     Create a Pandora instance
 3:     Register Algorithms and Plugins
 4:     Ask Pandora to parse XML settings file
 5:     for all Events do
 6:         Create CaloHit instances
 7:         Create MCParticle instances
 8:         Specify MCParticle-CaloHit relationships
 9:         Ask Pandora to process the event
10:         Get output PFOs and write to file
11:         Reset Pandora before next event
```

Create Pandora Instance

Pandora.h

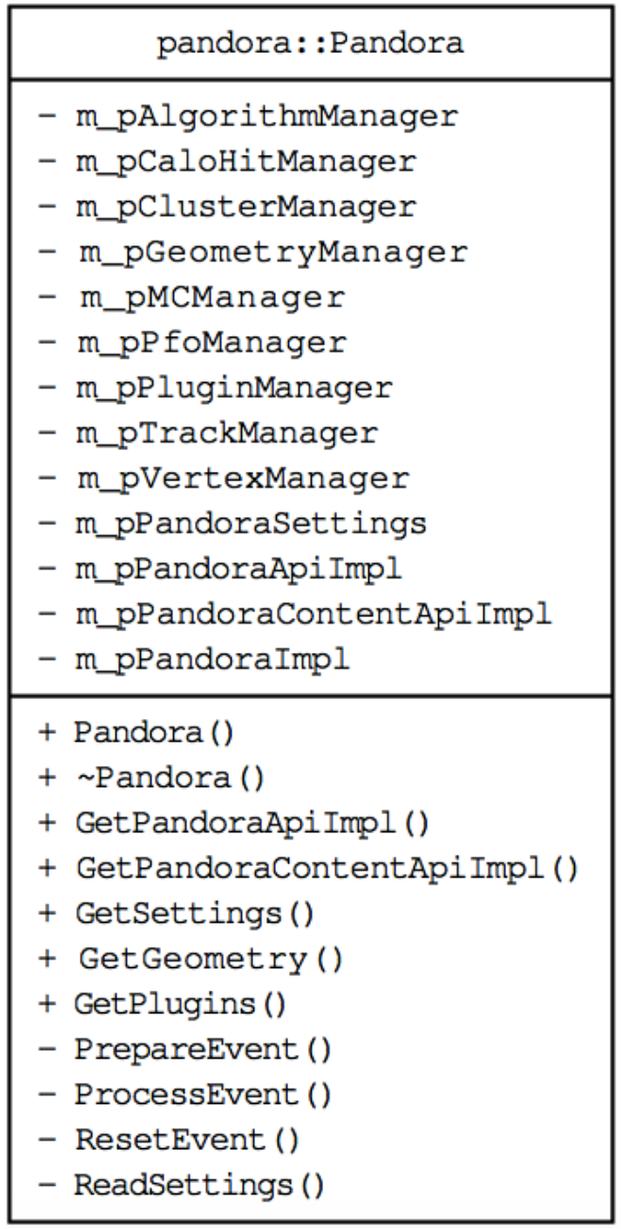
```
/**
 *    @brief Pandora class
 */
class Pandora
{
public:
    /**
     *  @brief  Default constructor
     */
    Pandora();

    ...
};
```

Only one instance needed for MicroBooNE, with its single 'drift volume'

In client app:

```
const pandora::Pandora *const pPandora = new pandora::Pandora();
```

- **Simple to create a Pandora instance (on stack or heap) via public default constructor.**
- **Will then find that its functionality is only available via its APIs, which are divided into:**
  i. **PandoraAPIs for use by a client app.**
  ii. **PandoraContentAPIs for use by algorithms.**



Member variables are addresses of Manager instances, API implementation instances and a Settings instance. Services are typically accessed via APIs.

Register LAr TPC Content

PandoraApi.h

```
/**
 *  @brief  Register an algorithm factory with pandora
 *
 *  @param  pandora the pandora instance to register the algorithm factory with
 *  @param  algorithmType the type of algorithm that the factory will create
 *  @param  pAlgorithmFactory the address of an algorithm factory instance
 */
static pandora::StatusCode RegisterAlgorithmFactory(const pandora::Pandora &pandora, const std::string &algorithmType,
    pandora::AlgorithmFactory *const pAlgorithmFactory);
```

API to register an algorithm factory, giving Pandora instance ability to instantiate a specific algorithm type

LArContent.h

Quickly register all 80+ algorithm factories in the LAr TPC 'content' library

```
/**
 *  @brief  Register all the lar content algorithms and tools with pandora
 *
 *  @param  pandora the pandora instance with which to register content
 */
static pandora::StatusCode RegisterAlgorithms(const pandora::Pandora &pandora);

/**
 *  @brief  Register lar coordinate transformation plugin with pandora
 *
 *  @param  pandora the pandora instance with which to register content
 *  @param  pLArTransformationPlugin the address of the lar transformation plugin
 */
static pandora::StatusCode SetLArTransformationPlugin(const pandora::Pandora &pandora,
    lar_content::LArTransformationPlugin *const pLArTransformationPlugin);
```

LArTransformationPlugin interface

```
/**
 *  @brief  Transform from (U,V) to W position
 *
 *  @param  U the U position
 *  @param  V the V position
 */
virtual double UVtoW(const double u, const double v) const = 0;


/**
 *  @brief  Transform from (U,V) to world volume Y coordinate
 *
 *  @param  U the U position
 *  @param  V the V position
 */
virtual double UVtoY(const double u, const double v)  const = 0;

...
```

Note preprocessor macro checking API return values

In client app:

```
const pandora::Pandora *const pPandora = new pandora::Pandora();
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::RegisterAlgorithms(*pPandora));
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::SetLArTransformationPlugin(*pPandora, new MicroBooNETransformationPlugin));
```

Read Pandora Settings

PandoraApi.h

```
/**
 *  @brief  Read pandora settings
 *
 *  @param  pandora the pandora instance to run the algorithms initialize
 *  @param  xmlFileName the name of the xml file containing the settings
 */
static pandora::StatusCode ReadSettings(const pandora::Pandora &pandora, const std::string &xmlFileName);
```

path to file describing Pandora reconstruction config.

In client app:

```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ReadSettings(*m_pPandora, configFileName));
```

- Specify which algorithms to instantiate and the order of algorithm execution.
- XML-configured multi-algorithm chains currently available for LAr TPC reco:
  - Dedicated reco for **cosmic ray** muons
  - Dedicated reco for **neutrino** events
  - **Cheated** reco (development use only!)
- Much more on this later and in exercises.

Example XML snippet - 3D track reco →

```
<!-- 3D track reconstruction -->
<algorithm type = "LArThreeDTransverseTracks">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputPfoListName>TrackParticles3D</OutputPfoListName>
    <TrackTools>
        <tool type = "LArClearTracks"/>
        <tool type = "LArLongTracks"/>
        <tool type = "LArOvershootTracks">
            <SplitMode>true</SplitMode>
        </tool>
        <tool type = "LArUndershootTracks">
            <SplitMode>true</SplitMode>
        </tool>
        <tool type = "LArOvershootTracks">
            <SplitMode>false</SplitMode>
        </tool>
        <tool type = "LArUndershootTracks">
            <SplitMode>false</SplitMode>
        </tool>
        <tool type = "LArMissingTrackSegment"/>
        <tool type = "LArTrackSplitting"/>
        <tool type = "LArLongTracks">
            <MinMatchedFraction>0.75</MinMatchedFraction>
            <MinXOverlapFraction>0.75</MinXOverlapFraction>
        </tool>
        <tool type = "LArMissingTrack"/>
    </TrackTools>
</algorithm>
```

Create Pandora Input

PandoraApi.h

```
/**
 *  @brief  Object creation helper class
 *
 *  @param  PARAMETERS the type of object parameters
 *  @param  OBJECT the type of object
 */
template <typename PARAMETERS, typename OBJECT>
class ObjectCreationHelper
{
public:
    typedef PARAMETERS Parameters;
    typedef OBJECT Object;

    /**
     *  @brief  Create a new object from a user factory
     *
     *  @param  pandora the pandora instance to create the new object
     *  @param  parameters the object parameters
     *  @param  factory the factory that performs the object allocation
     */
    static pandora::StatusCode Create(const pandora::Pandora &pandora, const Parameters &parameters,
        const pandora::ObjectFactory<Parameters, Object> &factory = pandora::PandoraObjectFactory<Parameters, Object>());
};

typedef ObjectCreationHelper<CaloHitParameters, pandora::CaloHit> CaloHit;
typedef ObjectCreationHelper<MCParticleParameters, pandora::MCParticle> MCParticle;
```

Advanced functionality: Can provide custom object instantiation factory to ‘decorate’ base objects in Pandora Event Data Model

Provides clean, simple interface to create any/all Pandora objects:

i. Construct parameters, e.g. PandoraApi::CaloHit::Parameters
ii. Assign properties to parameters public member variables
iii. Request object creation, e.g. PandoraApi::CaloHit::Create(…)
iv. Failure to assign to all properties will raise an exception

In client app:

```
PandoraApi::CaloHit::Parameters caloHitParameters;
caloHitParameters.m_positionVector = ...
caloHitParameters.m_expectedDirection = ...
...

PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::CaloHit::Create(*pPandora, caloHitParameters));
```

Create Pandora (Calo)Hits

PandoraApi.h

```
/**
 *  @brief  CaloHitParameters class
 */
class CaloHitParameters : public pandora::ObjectParameters
{
public:
    pandora::InputCartesianVector   m_positionVector;           ///< Position vector of center of calorimeter cell, units mm
    pandora::InputCartesianVector   m_expectedDirection;        ///< Unit vector in direction of expected hit propagation
    pandora::InputCartesianVector   m_cellNormalVector;         ///< Unit normal to sampling layer, pointing outwards from the origin
    pandora::InputCellGeometry      m_cellGeometry;             ///< The cell geometry type, pointing or rectangular
    pandora::InputFloat             m_cellSize0;                ///< Cell size 0 [pointing: eta, rect: up in ENDCAP, along beam in BARREL, units mm]
    pandora::InputFloat             m_cellSize1;                ///< Cell size 1 [pointing: phi, rect: perp. to size 0 and thickness, units mm]
    pandora::InputFloat             m_cellThickness;            ///< Cell thickness, units mm
    pandora::InputFloat             m_nCellRadiationLengths;    ///< Absorber material in front of cell, units radiation lengths
    pandora::InputFloat             m_nCellInteractionLengths;  ///< Absorber material in front of cell, units interaction lengths
    pandora::InputFloat             m_time;                     ///< Time of (earliest) energy deposition in this cell, units ns
    pandora::InputFloat             m_inputEnergy;              ///< Corrected energy of calorimeter cell in user framework, units GeV
    pandora::InputFloat             m_mipEquivalentEnergy;      ///< The calibrated mip equivalent energy, units mip
    pandora::InputFloat             m_electromagneticEnergy;    ///< The calibrated electromagnetic energy measure, units GeV
    pandora::InputFloat             m_hadronicEnergy;           ///< The calibrated hadronic energy measure, units GeV
    pandora::InputBool              m_isDigital;                ///< Whether cell should be treated as digital
    pandora::InputHitType           m_hitType;                  ///< The type of calorimeter hit
    pandora::InputHitRegion         m_hitRegion;                ///< Region of the detector in which the calo hit is located
    pandora::InputUInt              m_layer;                    ///< The subdetector readout layer number
    pandora::InputBool              m_isInOuterSamplingLayer;   ///< Whether cell is in one of the outermost detector sampling layers
    pandora::InputAddress           m_pParentAddress;           ///< Address of the parent calo hit in the user framework
};
```

InputTypes template checks assignment operator is used, plus vetoes NaN and INF assignments

- **List of variables to which client app must assign values before requesting Hit creation.**
- Still oriented towards collider experiments: plan to prune and ‘decorate’ with LAr-specific properties. Information available to algs, but doesn’t mean any/all properties *need* to be used.
- Algorithms can access information stored in Hits, but do not need to know how properties were obtained: client application isolates algorithms from input software framework (LArSoft).

Create Pandora MCParticles

PandoraApi.h

```
/**
 *  @brief  MCParticleParameters class
 */
class MCParticleParameters : public pandora::ObjectParameters
{
public:
    pandora::InputFloat             m_energy;                ///< The energy of the MC particle, units GeV
    pandora::InputCartesianVector   m_momentum;              ///< The momentum of the MC particle, units GeV
    pandora::InputCartesianVector   m_vertex;                ///< The production vertex of the MC particle, units mm
    pandora::InputCartesianVector   m_endpoint;              ///< The endpoint of the MC particle, units mm
    pandora::InputInt               m_particleId;            ///< The MC particle's ID (PDG code)
    pandora::InputMCParticleType    m_mcParticleType;        ///< The type of mc particle, e.g. vertex, 2D-projection, etc.
    pandora::InputAddress           m_pParentAddress;        ///< Address of the parent MC particle in the user framework
};
```

Properties that must be provided before MCParticle creation can be requested

```
/**
 *  @brief  Set parent-daughter mc particle relationship
 *
 *  @param  pandora the pandora instance to register the relationship with
 *  @param  pParentAddress address of parent mc particle in the user framework
 *  @param  pDaughterAddress address of daughter mc particle in the user framework
 */
static pandora::StatusCode SetMCParentDaughterRelationship(const pandora::Pandora &pandora, const void *const pParentAddress,
    const void *const pDaughterAddress);
```

Set parent-daughter relationships to full describe MCParticle hierarchy in Pandora

```
/**
 *  @brief  Set calo hit to mc particle relationship
 *
 *  @param  pandora the pandora instance to register the relationship with
 *  @param  pCaloHitParentAddress address of calo hit in the user framework
 *  @param  pMCParticleParentAddress address of mc particle in the user framework
 *  @param  mcParticleWeight weighting to assign to the mc particle
 */
static pandora::StatusCode SetCaloHitToMCParticleRelationship(const pandora::Pandora &pandora, const void *const pCaloHitParentAddress,
    const void *const pMCParticleParentAddress, const float mcParticleWeight = 1);
```

Set (custom/energy-weighted) relationships between Hits and MCParticles in Pandora

Run Pandora Algorithms

PandoraApi.h

```
/**
 *  @brief  Process an event
 *
 *  @param  pandora the pandora instance to process event
 */
static pandora::StatusCode ProcessEvent(const pandora::Pandora &pandora);
```

In client app:

```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ProcessEvent(*pPandora));
```

- **Pass thread to the Pandora instance, which will process the event by running the algorithms as specified in the Pandora Settings XML file.**
- **Algorithms will form Clusters, Vertices and Particles to represent the pattern-recognition solution. The thread will then be returned for output to be persisted.**

Extract Pandora Output

PandoraApi.h

```
/**
 *  @brief  Get the current pfo list
 *
 *  @param  pandora the pandora instance to get the objects from
 *  @param  pPfoList to receive the address of the particle flow objects
 */
static pandora::StatusCode GetCurrentPfoList(const pandora::Pandora &pandora, const pandora::PfoList *&pPfoList);
```

In client app:

```
const pandora::PfoList *pPfoList(nullptr);
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::GetCurrentPfoList(*pPandora, pPfoList));
```

- **Access list of Particles as specified/selected by final algorithm, and designated to be the 'current' list. Particles may have hierarchical list of daughters.**
- **From Particles, can navigate to constituent Clusters, Vertices and Hits. Can use ParentAddresses in Pandora objects to identify relevant input LArSoft objects.**

More on this topic in a later talk and exercise

Reset Pandora

PandoraApi.h

```
/**
 *  @brief  Reset pandora to process another event
 *
 *  @param  pandora the pandora instance to reset
 */
static pandora::StatusCode Reset(const pandora::Pandora &pandora);
```

In client app:

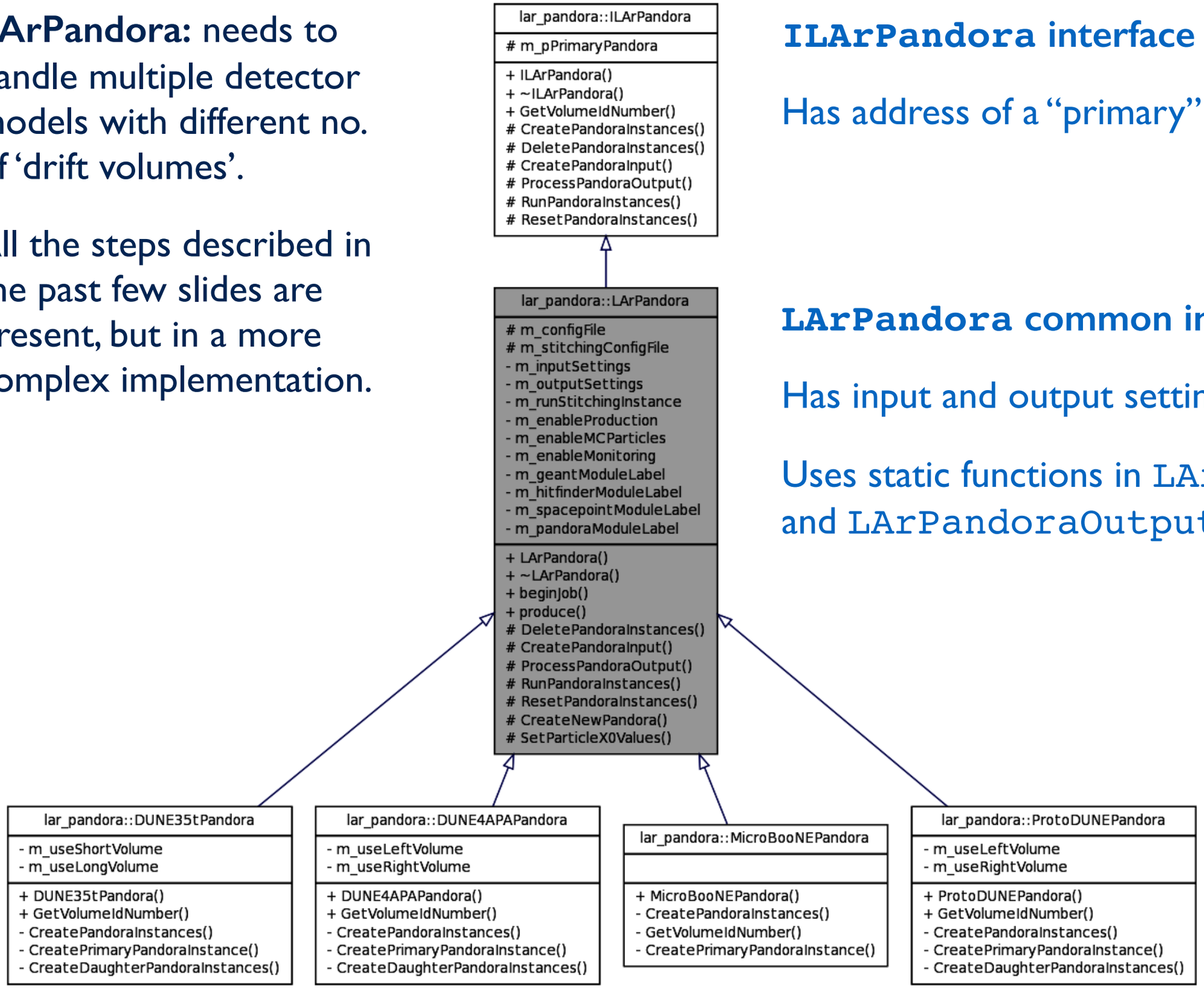
```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::Reset(*pPandora));
```

- **Ask to reset the Pandora instance, deleting all objects and lists made by algorithms and all input building-blocks provided by the client application.**
- **Pandora instance is then ready to begin receiving new Hits and MCParticles to describe the next input event.**

larpandora

LArPandora: needs to handle multiple detector models with different no. of 'drift volumes'.

All the steps described in the past few slides are present, but in a more complex implementation.



`ILArPandora` interface class

Has address of a "primary" Pandora instance

`LArPandora` common implementation

Has input and output settings instances

Uses static functions in `LArPandoraInput` and `LArPandoraOutput`

Create Pandora instance(s)

Specify wire pitches, angles to vertical, etc.

LArPandoraInput

larpandoracontent/LArPandoraInterface/LArPandoraInput.h

```
/**
 *  @brief  LArPandoraInput class
 */
class LArPandoraInput
{
public:
    /**
     *  @brief  Create the Pandora 2D hits from the ART hits
     *
     *  @param  settings the settings
     *  @param  hits the input list of ART hits for this event
     *  @param  idToHitMap to receive the mapping from Pandora hit ID to ART hit
     */
    static void CreatePandoraHits2D(const Settings &settings, const HitVector &hitVector, IdToHitMap &idToHitMap);

    /**
     *  @brief  Create pandora line gaps to cover any (continuous regions of) bad channels
     *
     *  @param  settings the settings
     */
    static void CreatePandoraLineGaps(const Settings &settings);

    /**
     *  @brief  Create the Pandora MC particles from the MC particles
     *
     *  @param  settings the settings
     *  @param  truthToParticles  mapping from MC truth to MC particles
     *  @param  particlesToTruth  mapping from MC particles to MC truth
     */
    static void CreatePandoraMCParticles(const Settings &settings, const MCTruthToMCParticles &truthToParticles,
        const MCParticlesToMCTruth &particlesToTruth);

    ...
```

CreatePandoraHits2D

1. Loop over recob::Hits; collect required information for self-describing Pandora Hits

```
void LArPandoraInput::CreatePandoraHits2D(const Settings &settings, const HitVector &hitVector, IdToHitMap &idToHitMap)
{
    if (!settings.m_pPrimaryPandora)
        throw pandora::StatusCodeException(pandora::STATUS_CODE_INVALID_PARAMETER);

    art::ServiceHandle<geo::Geometry> theGeometry;
    auto const* theDetector = lar::providerFrom<detinfo::DetectorPropertiesService>();

    int hitCounter(0);

    for (HitVector::const_iterator iter = hitVector.begin(), iterEnd = hitVector.end(); iter != iterEnd; ++iter)
    {
        const art::Ptr<recob::Hit> hit = *iter;
        const geo::WireID hit_WireID(hit->WireID());
        const pandora::Pandora *const pPandora(settings.m_pPrimaryPandora);

        const geo::View_t hit_View(hit->View());
        const double hit_Time(hit->PeakTime());
        const double hit_Charge(hit->Integral());
        const double hit_TimeStart(hit->PeakTimeMinusRMS());
        const double hit_TimeEnd(hit->PeakTimePlusRMS());

        double xyz[3];
        theGeometry->Cryostat(hit_WireID.Cryostat).TPC(hit_WireID.TPC).Plane(hit_WireID.Plane).Wire(hit_WireID.Wire).GetCenter(xyz);
        const double y0_cm(xyz[1]);
        const double z0_cm(xyz[2]);

        const double wire_pitch_cm(theGeometry->WirePitch(hit_View)); // cm

        const double xpos_cm(theDetector->ConvertTicksToX(hit_Time, hit_WireID.Plane, hit_WireID.TPC, hit_WireID.Cryostat));
        const double dxpos_cm(std::fabs(theDetector->ConvertTicksToX(hit_TimeEnd, hit_WireID.Plane, hit_WireID.TPC, hit_WireID.Cryostat) -
            theDetector->ConvertTicksToX(hit_TimeStart, hit_WireID.Plane, hit_WireID.TPC, hit_WireID.Cryostat)));

        const double mips(LArPandoraInput::GetMips(settings, hit_Charge, hit_View));

        // Continued on next slide
        ...
    }
}
```

CreatePandoraHits2D

```
PandoraApi::CaloHit::Parameters caloHitParameters;
caloHitParameters.m_expectedDirection = pandora::CartesianVector(0., 0., 1.);
caloHitParameters.m_cellNormalVector = pandora::CartesianVector(0., 0., 1.);
caloHitParameters.m_cellSize0 = settings.m_dx_cm;
caloHitParameters.m_cellSize1 = (settings.m_useHitWidths ? dxpos_cm : settings.m_dx_cm);
caloHitParameters.m_cellThickness = wire_pitch_cm;
caloHitParameters.m_cellGeometry = pandora::RECTANGULAR;
caloHitParameters.m_time = 0.;
caloHitParameters.m_nCellRadiationLengths = settings.m_dx_cm / settings.m_rad_cm;
caloHitParameters.m_nCellInteractionLengths = settings.m_dx_cm / settings.m_int_cm;
caloHitParameters.m_isDigital = false;
caloHitParameters.m_hitRegion = pandora::SINGLE_REGION;
caloHitParameters.m_layer = 0;
caloHitParameters.m_isInOuterSamplingLayer = false;
caloHitParameters.m_inputEnergy = hit_Charge;
caloHitParameters.m_mipEquivalentEnergy = mips;
caloHitParameters.m_electromagneticEnergy = mips * settings.m_mips_to_gev;
caloHitParameters.m_hadronicEnergy = mips * settings.m_mips_to_gev;
caloHitParameters.m_pParentAddress = (void*)((intptr_t)(++hitCounter));

if (hit_View == geo::kW)
{
    caloHitParameters.m_hitType = pandora::TPC_VIEW_W;
    const double wpos_cm(z0_cm);
    caloHitParameters.m_positionVector = pandora::CartesianVector(xpos_cm, 0., wpos_cm);
}
else if(hit_View == geo::kU)
{
    caloHitParameters.m_hitType = pandora::TPC_VIEW_U;
    const double upos_cm(lar_content::LArGeometryHelper::GetLArTransformationPlugin(*pPandora)->YZtoU(y0_cm, z0_cm));
    caloHitParameters.m_positionVector = pandora::CartesianVector(xpos_cm, 0., upos_cm);
}
else if(hit_View == geo::kV)
{
    caloHitParameters.m_hitType = pandora::TPC_VIEW_V;
    const double vpos_cm(lar_content::LArGeometryHelper::GetLArTransformationPlugin(*pPandora)->YZtoV(y0_cm, z0_cm));
    caloHitParameters.m_positionVector = pandora::CartesianVector(xpos_cm, 0., vpos_cm);
}
else
{
    mf::LogError("LArPandora") << " --- WARNING: UNKNOWN VIEW !!!  (View=" << hit_View << ")" << std::endl;
    throw pandora::StatusCodeException(pandora::STATUS_CODE_FAILURE);
}

idToHitMap[hitCounter] = hit;
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::CaloHit::Create(*pPandora, caloHitParameters));
```

2. Assign information to members in **PandoraApi::CaloHit::Parameters** instance, then call **PandoraApi::CaloHit::Create**

LArPandoraOutput

larpandoracontent/LArPandoraInterface/LArPandoraOutput.h

```
class LArPandoraOutput
{
public:
    /**
     *  @brief  Convert the Pandora PFOs into ART clusters and write into ART event
     *
     *  @param  settings the settings
     *  @param  idToHitMap the mapping from Pandora hit ID to ART hit
     *  @param  evt the ART event
     */
    static void ProduceArtOutput(const Settings &settings, const IdToHitMap &idToHitMap, art::Event &evt);

    /**
     *  @brief Build a recob::Cluster object from an input vector of recob::Hit objects
     *
     *  @param id the id code for the cluster
     *  @param hitVector the input vector of hits
     *  @param isolatedHits the input list of isolated hits, not to be fed to the cluster parameter algorithms
     *  @param algo Algorithm set to fill cluster members, if unsure StandardClusterParamsAlg is a good default
     */
    static recob::Cluster BuildCluster(const int id, const HitVector &hitVector, const HitList &isolatedHits,
       cluster::ClusterParamsAlgBase &algo);

    /**
     *  @brief Lookup ART hit from an input Pandora hit
     *
     *  @param idToHitMap the mapping between Pandora and ART hits
     *  @param pCaloHit the input Pandora hit (2D)
     */
    static art::Ptr<recob::Hit> GetHit(const IdToHitMap &idToHitMap, const pandora::CaloHit *const pCaloHit);

    ...
```

pandoramodules_microboone.fcl

Pre-defined producer types

```
BEGIN_PROLOG

microboone_pandora:
{
  module_type:            "MicroBooNEPandora"
  ConfigFile:             "PandoraSettings_MicroBooNE_Neutrino.xml"
  GeantModuleLabel:       "largeant"
  HitFinderModuleLabel:   "gaushit"
  EnableMCParticles:      false
  EnableProduction:       true
  EnableMonitoring:       false
  EnableLineGaps:         true
  UseHitWidths:           true
  BuildTracks:            true
  BuildShowers:           false
}

microboone_pandoracosmic:                       @local::microboone_pandora
microboone_pandoracosmic.ConfigFile:            "PandoraSettings_MicroBooNE_Cosmic.xml"

microboone_pandoraneutrino:                     @local::microboone_pandora
microboone_pandoraneutrino.ConfigFile:          "PandoraSettings_MicroBooNE_Neutrino.xml"

microboone_pandorawriter:                       @local::microboone_pandora
microboone_pandorawriter.ConfigFile:            "PandoraSettings_Write.xml"
microboone_pandorawriter.EnableMCParticles:     true
microboone_pandorawriter.EnableProduction:      false

END_PROLOG
```

Cosmic-ray reco

Neutrino reco

Write LArSoft inputs to Pandora formats

Only major difference between producer types is Pandora Settings files, which dictates alternative algorithm selection/configuration

LArSoft↔Pandora translation typically remains unchanged whatever the pattern-recognition configuration (may choose to alter e.g. input Hit collection)

In principle simple; in reality tends to be rather technical.
Haven't done any pattern recognition yet!

But, have got all input processing 'out of the way' and can now present problem in (hopefully) simple and well-defined manner

Questions?