Status of `Geometry` service changes to accommodate pixel readouts

Part 2 of N

Kyle J. Knoepfel

LArSoft coordination meeting

29 November 2022

Motivation

- **LArSoft intends to support pixel geometries**
  - To do this, some adjustments to the Geometry service/system are required.
    - *Will likely be separating readout-specific concepts from those of geometry.*
  - A few of us are meeting weekly to determine how best to proceed.
  - While analyzing geometry code, it became apparent that much of the interface serves as “legacy” code to support older coding patterns

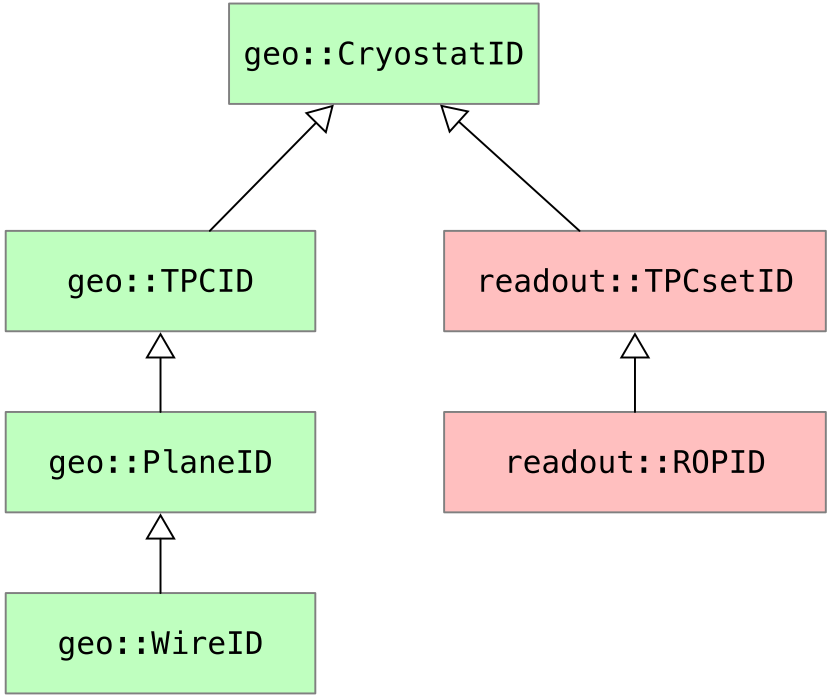
Motivation

- **LArSoft intends to support pixel geometries**
  - To do this, some adjustments to the Geometry service/system are required.
    - *Will likely be separating readout-specific concepts from those of geometry.*
  - A few of us are meeting weekly to determine how best to proceed.
  - While analyzing geometry code, it became apparent that much of the interface serves as "legacy" code to support older coding patterns

- **Maintenance issues**
  - We will need to rearrange some parts of the code to support pixel geometries—***it's less work to adjust only the code that's required***.
  - Recently we removed a lot of "deprecated" code.
  - **I'd now like to address the large number of overloads** (+ some missed deprecations), **and the geometry iteration patterns.**

Geometry IDs

- The `larcorealg` repository provides an inheritance-based system for identifying elements of a LArTPC geometry.
- Any ID object can access the ID properties of its base class.
- Provides degree of extensibility.
- Used consistently, this type of system lends itself to simple APIs.
- It is also the basis for smart iteration through geometry elements.



Geometry IDs

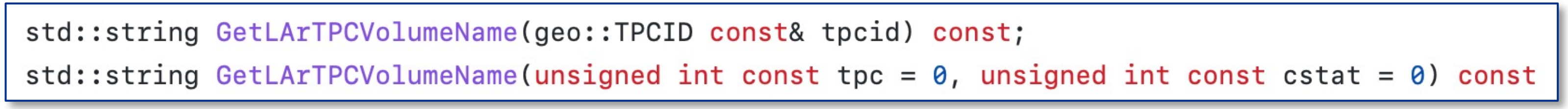
- Much of the geometry interface has supported both the geometry ID system and simple unsigned integer arguments.

  This can lead to confusion.

Geometry IDs

- Much of the geometry interface has supported both the geometry ID system and simple unsigned integer arguments.

This can lead to confusion.



```
std::string GetLArTPCVolumeName(geo::TPCID const& tpcid) const;
std::string GetLArTPCVolumeName(unsigned int const tpc = 0, unsigned int const cstat = 0) const
```

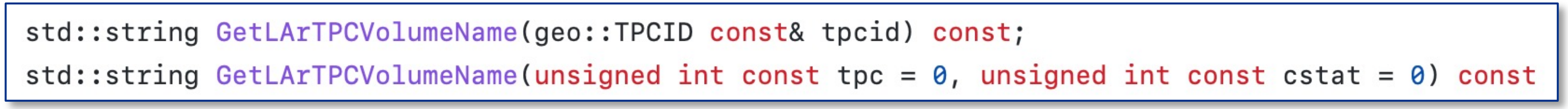
Use:

```
geom->GetLArTPCVolumeName(TPCID{1, 2}); // or
geom->GetLArTPCVolumeName(      2, 1 );
```

Geometry IDs

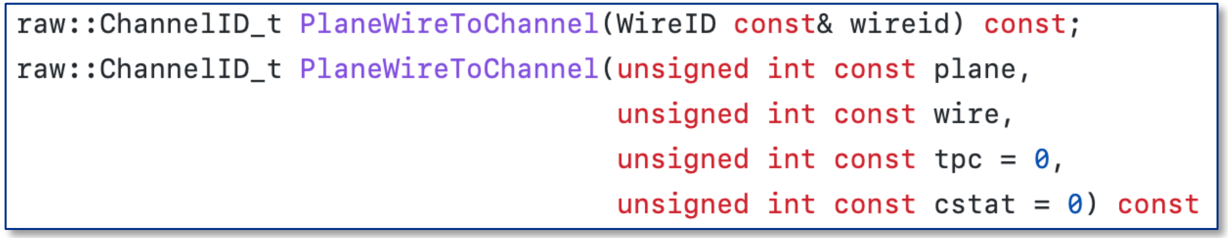
- Much of the geometry interface has supported both the geometry ID system and simple unsigned integer arguments.

This can lead to confusion.



```
std::string GetLArTPCVolumeName(geo::TPCID const& tpcid) const;
std::string GetLArTPCVolumeName(unsigned int const tpc = 0, unsigned int const cstat = 0) const
```

Use:
```
geom->GetLArTPCVolumeName(TPCID{1, 2}); // or
geom->GetLArTPCVolumeName(      2, 1 );
```



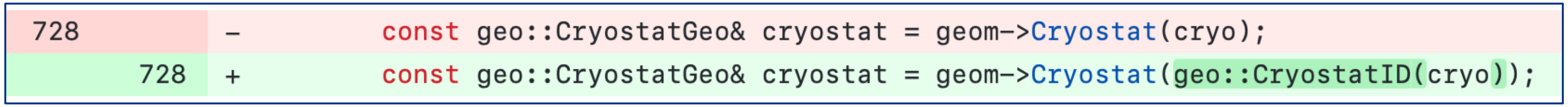
```
raw::ChannelID_t PlaneWireToChannel(WireID const& wireid) const;
raw::ChannelID_t PlaneWireToChannel(unsigned int const plane,
                                    unsigned int const wire,
                                    unsigned int const tpc = 0,
                                    unsigned int const cstat = 0) const
```

Use:
```
geom->PlaneWireToChannel(WireID{1, 2, 3, 4}); // or
geom->PlaneWireToChannel(      3, 4, 2, 1 );
```

Change 1: Only Geometry IDs will be used for geometry interface

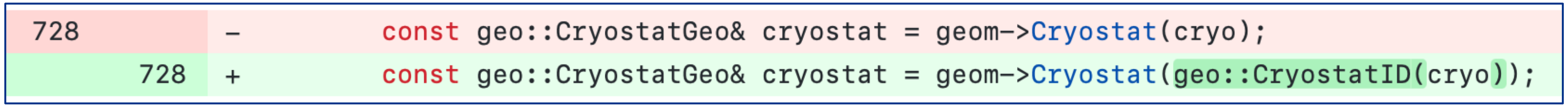
Fermilab

Change 1: Only Geometry IDs will be used for geometry interface

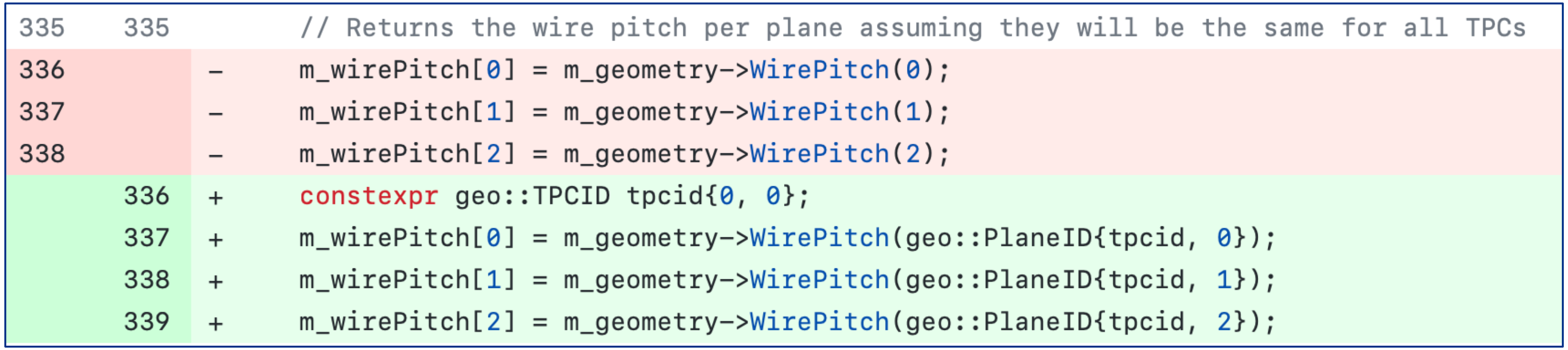


```diff
-       const geo::CryostatGeo& cryostat = geom->Cryostat(cryo);
+       const geo::CryostatGeo& cryostat = geom->Cryostat(geo::CryostatID(cryo));
```

Change 1: Only Geometry IDs will be used for geometry interface

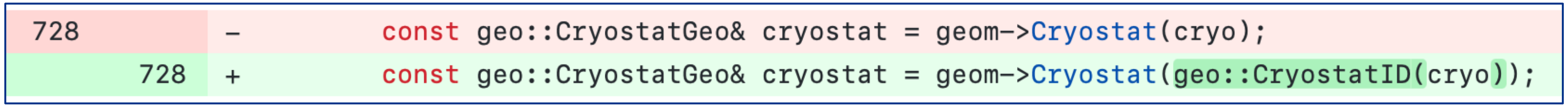


```
-        const geo::CryostatGeo& cryostat = geom->Cryostat(cryo);
+        const geo::CryostatGeo& cryostat = geom->Cryostat(geo::CryostatID(cryo));
```

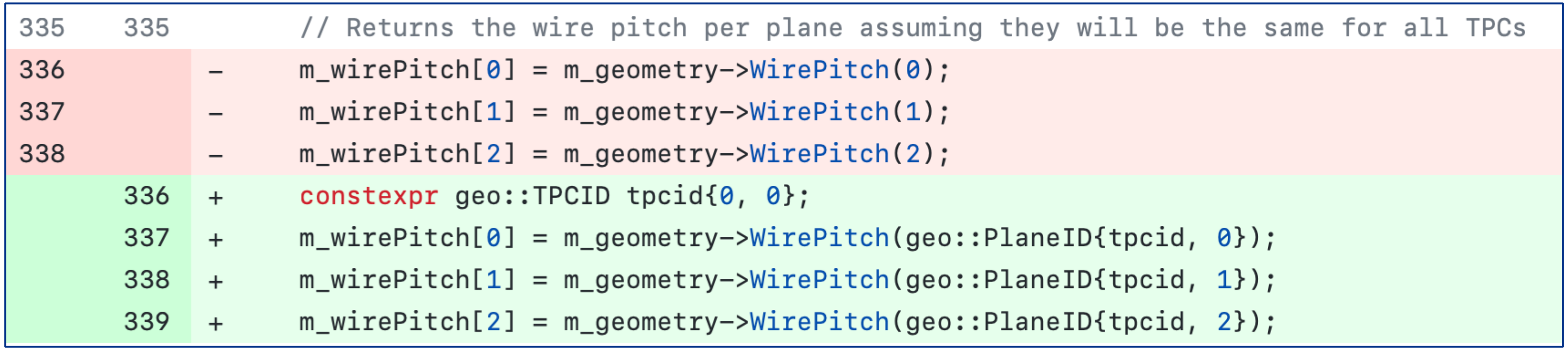


```
      // Returns the wire pitch per plane assuming they will be the same for all TPCs
-     m_wirePitch[0] = m_geometry->WirePitch(0);
-     m_wirePitch[1] = m_geometry->WirePitch(1);
-     m_wirePitch[2] = m_geometry->WirePitch(2);
+     constexpr geo::TPCID tpcid{0, 0};
+     m_wirePitch[0] = m_geometry->WirePitch(geo::PlaneID{tpcid, 0});
+     m_wirePitch[1] = m_geometry->WirePitch(geo::PlaneID{tpcid, 1});
+     m_wirePitch[2] = m_geometry->WirePitch(geo::PlaneID{tpcid, 2});
```

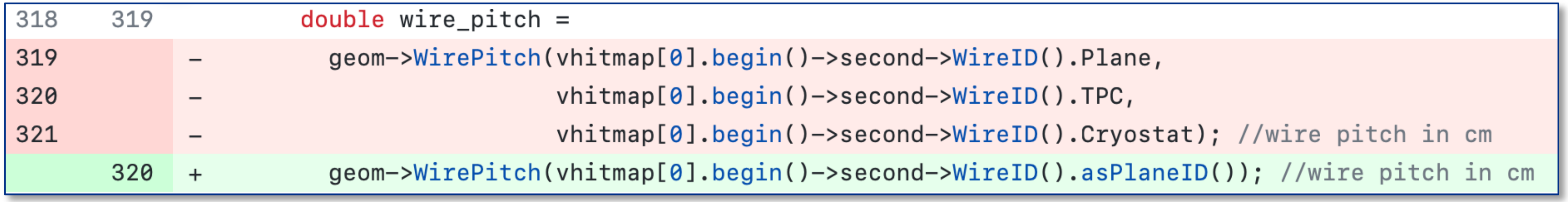
Change 1: Only Geometry IDs will be used for geometry interface



```diff
-    const geo::CryostatGeo& cryostat = geom->Cryostat(cryo);
+    const geo::CryostatGeo& cryostat = geom->Cryostat(geo::CryostatID(cryo));
```



```diff
   // Returns the wire pitch per plane assuming they will be the same for all TPCs
-  m_wirePitch[0] = m_geometry->WirePitch(0);
-  m_wirePitch[1] = m_geometry->WirePitch(1);
-  m_wirePitch[2] = m_geometry->WirePitch(2);
+  constexpr geo::TPCID tpcid{0, 0};
+  m_wirePitch[0] = m_geometry->WirePitch(geo::PlaneID{tpcid, 0});
+  m_wirePitch[1] = m_geometry->WirePitch(geo::PlaneID{tpcid, 1});
+  m_wirePitch[2] = m_geometry->WirePitch(geo::PlaneID{tpcid, 2});
```



```diff
     double wire_pitch =
-      geom->WirePitch(vhitmap[0].begin()->second->WireID().Plane,
-                      vhitmap[0].begin()->second->WireID().TPC,
-                      vhitmap[0].begin()->second->WireID().Cryostat); //wire pitch in cm
+      geom->WirePitch(vhitmap[0].begin()->second->WireID().asPlaneID()); //wire pitch in cm
```

Vector overloads

- There are many point or vector geometry functions with overloads that support:
    - `double const*/double[3]`
    - `TVector3 const&`
    - `geo::Point_t const&`
    - `geo::Vector_t const&`

- **Change 2:** Only `geo::Point_t` and `geo::Vector_t` function arguments will be supported.

Geometry iterators and iteration patterns

- The `GeometryCore.h` file contains many lines of code to support smart iteration through geometry elements (e.g.):

```
geometry->begin_wire_id(); // Get iterator to first wire ID
for (geo::TPCGeo const& tpc : geometry->IterateTPCs())
```

- Very useful, but it hard-codes geometry element names into the interface.

Geometry iterators and iteration patterns

- The `GeometryCore.h` file contains many lines of code to support smart iteration through geometry elements (e.g.):

```
geometry->begin_wire_id(); // Get iterator to first wire ID
for (geo::TPCGeo const& tpc : geometry->IterateTPCs())
```

- Very useful, but it hard-codes geometry element names into the interface.

- This code can be rearranged to use templates so that iterating through elements *does not* require a member function with element names hard-coded into the function name.

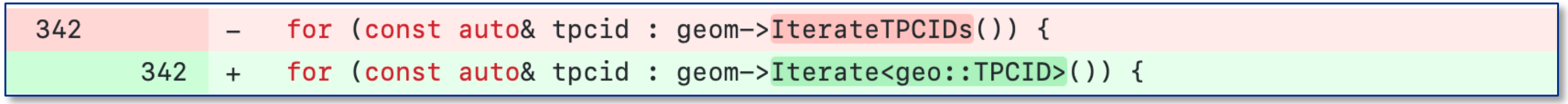
- This is a step toward factorizing geometry and readout constructs.

Geometry iterators and iteration patterns

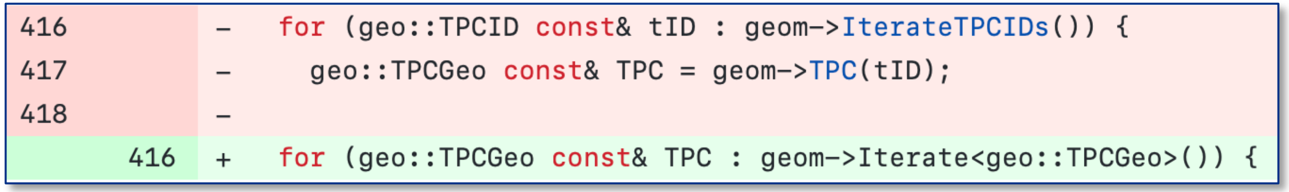
- **Change 3:** Geometry iterators will become largely internal and you will specify the type of object you want to iterate through via template argument:

Geometry iterators and iteration patterns

- **Change 3:** Geometry iterators will become largely internal and you will specify the type of object you want to iterate through via template argument:



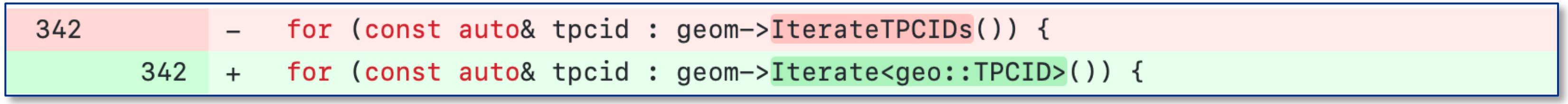
```
-   for (const auto& tpcid : geom->IterateTPCIDs()) {
+   for (const auto& tpcid : geom->Iterate<geo::TPCID>()) {
```



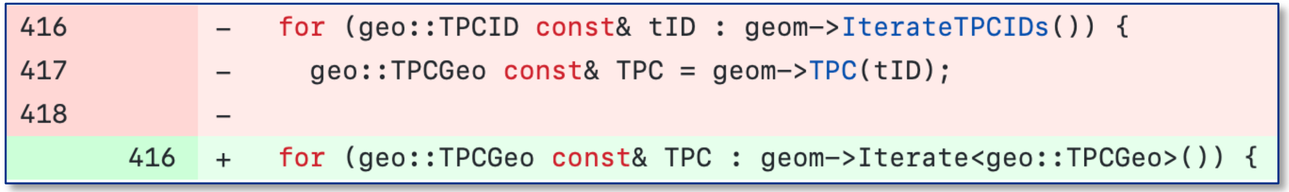
```
-   for (geo::TPCID const& tID : geom->IterateTPCIDs()) {
-     geo::TPCGeo const& TPC = geom->TPC(tID);
-
+   for (geo::TPCGeo const& TPC : geom->Iterate<geo::TPCGeo>()) {
```

Geometry iterators and iteration patterns

- **Change 3:** Geometry iterators will become largely internal and you will specify the type of object you want to iterate through via template argument:



```diff
-   for (const auto& tpcid : geom->IterateTPCIDs()) {
+   for (const auto& tpcid : geom->Iterate<geo::TPCID>()) {
```

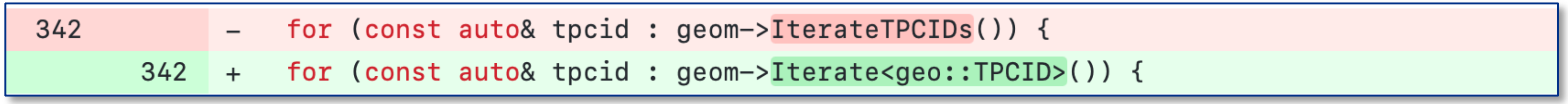


```diff
-   for (geo::TPCID const& tID : geom->IterateTPCIDs()) {
-     geo::TPCGeo const& TPC = geom->TPC(tID);
-
+   for (geo::TPCGeo const& TPC : geom->Iterate<geo::TPCGeo>()) {
```

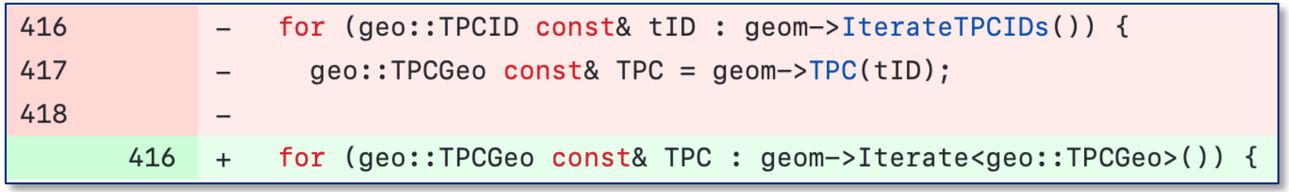
- The API for iterating through ID objects and Geo objects is the same.
- Specific iterator names (e.g. `geo::wire_iterator`) will be removed.

Geometry iterators and iteration patterns

- **Change 3:** Geometry iterators will become largely internal and you will specify the type of object you want to iterate through via template argument:



```
-   for (const auto& tpcid : geom->IterateTPCIDs()) {
+   for (const auto& tpcid : geom->Iterate<geo::TPCID>()) {
```



```
-   for (geo::TPCID const& tID : geom->IterateTPCIDs()) {
-     geo::TPCGeo const& TPC = geom->TPC(tID);
-
+   for (geo::TPCGeo const& TPC : geom->Iterate<geo::TPCGeo>()) {
```

- The API for iterating through ID objects and Geo objects is the same.
- Specific iterator names (e.g. `geo::wire_iterator`) will be removed.
- Removes 1-2k LOC from `larcorealg`.

Conclusion

- We are continuing the effort to clean up/pare down the geometry interface.
- The next raft of PRs will remove remaining obsolete interface and will introduce a slightly different iteration pattern that is more conducive to a factorized geometry/readout system.

- **Status**

  I have LArSoft feature branches ready, where almost all of the deprecated functionality has been removed and is no longer used.

  I have started feature branches for the experiments.