Specifying detector components in configuration files

Gianluca Petrillo

SLAC National Accelerator Laboratory, U.S.A.

LArSoft Coordination Meeting, June 2, 2020

SLAC NATIONAL ACCELERATOR LABORATORY

Use case

Some times algorithms need to know which geometry element(s) to work with:

- TPC digitisation: *inject a test charge in this wire at this time*
- reconstruction: *operate only on hits from selected TPC*
- ...

The syntax and number of parameters to code is bulky:

```
{
  TestChargeCryostat: 0
  TestChargeTPC:      2
  TestChargePlane:    1
  TestChargeWire:  1542
  # ...
}
```

... and some code may “forget” there may be multiple TPC or cryostats.
To include more than one component is proportionally longer.

Configuration syntax

A simple implementation introduces a simplified syntax for the same purpose, mirroring the standard way of LArSoft to identify geometry components:

```
{
  TestCharge: { C:0 T:2 P:1 W:1542 }
  # ...
}
```

If a list needs to be specified, the standard FHiCL syntax is:

```
{
  OnlyOnTPC: [ { C:0 T:0 }, { C:0 T:1 } ]  # two TPC
  # ...
}
```

Notes:

- `TestCharge` in the example is just a regular FHiCL table which by chance matches the standard LArSoft ID printout
- elements *must not* be in quotes

Reading the configuration

The algorithm reading that configuration should look like:

```
#include "larcoreobj/SimpleTypesAndConstants/geo_types_fhicl.h" // geo::fhicl

class MyAlgorithm {
  std::optional<geo::WireID> const fTestChargeWire;
  std::vector<geo::TPCID> const fTPCs;
    public:
  struct Config {
    geo::fhicl::OptionalWireID TestCharge
      { fhicl::Name("TestCharge"), fhicl::Comment("wire to inject charge into") };
    geo::fhicl::TPCIDsequence TPCs
      { fhicl::Name("TPCs"), fhicl::Comment("selected TPCs") };
  };
}; // class MyAlgorithm

MyAlgorithm::MyAlgorithm(fhicl::Table<Config> const& config)
  : fTestChargeWire(readParameter(config().TestCharge))
  , fTPCs           (readParameter(config().TPCs))
  {}
```

Reading the configuration — commentary

- yep, this syntax is only available with FHiCL validation!
- instead of the usual `fhicl::Sequence<geo::TPCID>`, the special class `geo::fhicl::TPCIDsequence` is used
- several types are available to read optional or mandatory, atoms or sequences, with or without default values, for all geometry IDs
  → all of them can be read with `readParameter()`
- optional configuration is read as `std::optional` types
- currently there is no (easy) way to specify a default value for a sequence; the workaround is to use an optional sequence, and specify a default value in `readParameter()`
- the compiler will find `readParameter()` in namespace `geo::fhicl` because that's where the type of its argument is defined (ADL)

Ah the future!

With implementation of feature #23669, the syntax will adhere back to standard FHiCL.

Summary

- LArSoft offers geometry and readout element ID types (`geo::PlaneID`, `geo::ROPID`, ...)
- these IDs help algorithms be detector-independent and they are supported by LArSoft interface
- a new syntax is offered to conveniently specify geometry and readout elements in the configuration
- code will become even simpler with the addition of new FHiCL features

Some examples can be found in the documentation of `geo::fhicl` and `readout::fhicl` namespaces.

A possible equivalent in the old way: configuration

These are two possible ways to implement the same example as before.

```
{
  TestChargeCryo:    0
  TestChargeTPC:     2
  TestChargePlane:   1
  TestChargeWire: 1542

  OnlyOnTPC: [ [ 0, 0 ], [ 0, 1 ] ]  # C:0 T:0 and C:0 T:1
}
```

A possible equivalent in the old way (non-validated)

And this is how the code reading it may look like:

```
class MyAlgorithm {
  std::optional<geo::WireID> const fTestChargeWire;
  std::vector<geo::TPCID> fTPCs; // can't be const
    public:
  MyAlgorithm(fhicl::ParameterSet const& pset);
}; // class MyAlgorithm

MyAlgorithm::MyAlgorithm(fhicl::ParameterSet const& pset)
  : fTestChargeWire{
    pset.is_key_to_atom("TestChargeCryo")
    ? std::optional(geo::WireID{
      pset.get<unsigned int>("TestChargeCryo"),  pset.get<unsigned int>("TestChargeTPC"),
      pset.get<unsigned int>("TestChargePlane"), pset.get<unsigned int>("TestChargeWire")
      })
    : std::nullopt
    }
{
  for (auto const& idNum: pset.get<std::vector<std::vector<unsigned int>>>("OnlyOnTPC"))
    fTPCs.emplace_back(idNum[0], idNum[1]);
}
```