Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

Session 8: Details of Module Configuration

Rob Kutschke

art and LArSoft Course

August 4, 2015

Run-time Configuration

- Sometimes shortened to just “configuration”
- Fermilab Hierarchical Configuration Language
  - FHiCL
  - Convention: files end in .fcl
- This session will answer the question:
  - How do you put something into your .fcl file so that it changes what your module does?
- This exercise teaches ideas and tools
  - Consult with your experiment to learn their standards and practices for using the tools wisely.

Recap: `OptionalMethods/optional.fcl`

```
physics : {
  analyzers: {
    opt : {
      module_type : Optional
      // In this exercise you will provide
      // additional definitions that
      // modify what your module does
    }
  }
}
```

ParameterSets/pset01.fcl and PSet01_module.cc

```
physics : { analyzers: {
    psetTester : {
      module_type : PSet01
      // Additional Definitions
    } }}
```

```
namespace tex {
  class PSet01 : public art::EDAnalyzer {
  public:
    explicit PSet01(fhicl::ParameterSet const& pset );
  }; }
```

- The argument `pset` is a copy of the .fcl fragment in red
  - It's the same information, but in a different format, called an fhicl::ParameterSet

Accessing the Information

```
psetTester : {
  module_type : PSet01
  p1          : "this is quoted string"
}
```

```
tex::PSet01::PSet01(fhicl::ParameterSet const& pset ):
  art::EDAnalyzer(pset){
  std::string p1 = pset.get<std::string>("p1");
}
```

- The identifiers in red must match exactly
- Type is provided to FHiCL as a template argument
- Good practice: green name similar to the red name

Prefer Initializer Lists for Member Data

```
class PSet02 : public art::EDAnalyzer {
  public:
    explicit PSet02(fhicl::ParameterSet const& );
    void analyze(art::Event const& event) override;
  private:
    int b_;
};
```

```
tex::PSet02::PSet02(fhicl::ParameterSet const& pset ):
  art::EDAnalyzer(pset),
  b_(pset.get<int>("b")){
}
```

- Order of members in the initializer list should match the order of members in the class declaration (sometimes it must!)

Types - 1

- The FHiCL parser and `fhicl::ParameterSet` both store all values as strings
  - Conversion to your requested type is done at the time that you call `pset.get<type>("name")`
- FHiCL knows about:
  - primitive types: `int, unsigned, double, float` etc
  - `std::string, std::tuple`
  - `fhicl::ParameterSet`
  - `std::vector<any of the above types type>`
    - Uses the FHiCL sequence notation:

```
fileNames : [ "file1.art" , "file2.art", "file3.art" ]
```

Types - 2

- *art* extends FHiCL:

```
offset : [ -3904., 0., 10200. ]
```

- Can be read as:

```
CLHEP::Hep3Vector offset =
pset.get<CLHEP::Hep3Vector>("offset")
```

- Or, if you like auto:

```
auto offset = pset.get<CLHEP::Hep3Vector>("offset")
```

- Similarly for CLHEP::HepLorentzVector
  - The order convention is: [px, py, pz, e]
- You may also define your own supported types.
  - Contact the *art* team.

Errors on Type Conversion

```
p1 : "ack thpppt"
```

```
int p1 = pset.get<int>("p1");
```

- The conversion will throw an exception
- *art* will catch the exception and attempt a graceful shutdown
  - Stop processing the current event
  - Call `endSubRun` for all modules
  - Call `endRun` for all modules
  - Call `endJob` for all modules
  - Call the destructor for all modules
  - Properly flush and close all output files
  - The goal is that you get complete output up to the exception.

Check Parameters for Validity!

```
class PSet08 : public art::EDAnalyzer {
  public:
    explicit PSet08(fhicl::ParameterSet const& );
    void analyze(art::Event const& event) override;
  private:
    double weight_;// Must be between 0 and 1
};
```

```
tex::PSet08::PSet08(fhicl::ParameterSet const& pset ):
  art::EDAnalyzer(pset),
  weight(pset.get<int>("weight")){
}
```

- It is your job to verify that all parameters have valid values
- If a parameter has an invalid value, throw an exception
  - See `PSet08_module.cc` and `pset08.fcl`

Missing Parameters

```
int p1 = pset.get<int>("p1");
```

- If `p1` is not present in the parameter set, the code in pset.get<int> will throw an exception
- *art* will catch the exception and attempt a graceful shutdown

Default Values

```
int p1 = pset.get<int>("p1",42);
```

- If `p1` is present in the parameter set, the second argument is ignored and the value from the parameter set is returned.
- `If p1` is not present in the parameter set, the parameter set code will return the value given by the second argument
- Consult your experiment for standards and practices about using default values
  - Usually OK to use default values for parameters that control things like the verbosity of diagnostics.
  - We recommend no default values for parameters that affect physics.

Canonical Form of Numbers

- When FHiCL recognizes a value as a number, the string that it stores may not be exactly what it found in the file.
  - All numbers are converted to a canonical form
- Suppose that two files are identical except one file has:

  n : 0.0
  - And the other has:

    n : 0
- FHiCL will store both numbers as: 0
- When numbers are stored in canonical form, the two files can be recognized, programmatically, as logically identical.
- This is discussed in detail in the write-up for this exercise.

FHiCL Recognizes these Special Values

- `true, false`
- `infinity, +infinity, -infinity`
- `@nil`
  - A special value that cannot convert to any type
  - An attempt to get this value as any type, even as a string, will throw an exception.
- These must never be quoted.

Quoting

- A string must be quoted if it contains any white space or any special characters
- A non-quoted string may only contain letters, numbers and underscore ("_")
- It is always safe to quote strings and numbers
- Things that must NOT be quoted:
  - FHiCL names:

```
"name" : value  //  Error
```

  - The special values:

```
name : "@nil"  // Is just a string, not a special value
```

Module Hygiene

- Did you remember to use override?
- When a compiler supplied function will do the right thing, let the compiler write it for you.
  - Design your classes so that the compiler written d'tor will work.
    - One part of this is using safe pointers.
- Does your c'tor initialize all data members to a valid state?
- If a member is a pointer, use an appropriate safe pointer:
  - Often the right safe pointer is: `std::unique_ptr<T>`
    - Avoid `new;` prefer `std::make_unique_ptr<T>();`
  - If you are not sure, ask your experiment or the *art* team.

Looking Forward

- This exercise will cover the basics of parameter sets
  - Enough for most exercises in this school
- You will learn more about parameter sets in Session 22
- There are many ways to use parameter sets
  - Consult with your experiment to learn the standards and practices that they recommend
- In a release of *art*, coming soon, there will be a new facility to let you describe the contents of a legal parameter set.
  - The facility can is able to check the parameter set for validity
  - It also provides a user help facility.

Questions so Far?

Get Started

- Start to work on Chapter 14 (Exercise 4) in the *art* workbook writeup:
  - https://web.fnal.gov/project/ArtDoc/Shared%20Documents/art-documentation.pdf
- There is a lot to do; come back to it later if you do not finish!

- My Powerpoint is flakey.
- If the above link fails or if it display pdf as text, try:
  - https://web.fnal.gov/project/ArtDoc/SitePages/documentation.aspx
  - Under latest releases, click on the document with the highest version number.
- If both links fail, mouse in the url.

Backup Slides:

Hints on Navigating the Giant PDF file

- Title page
- Blank page
- List of Chapters (3 pages long)
- Detailed Table of Contents (16 pages long)
- Everything is internally hyperlinked:
  - Page numbers in the TOC, and index
  - Table, Listing, Figure and Section cross-references
  - Configure your browser to highlight hyperlinks.
- Many PDF browsers have previous and next buttons
  - MAC Safari
    - Back: Apple-[
    - Forward: Apple-]

Aside: About a fragment appearing in all .fcl files to date

```
#include "fcl/minimalMessageService.fcl"
// … lines deleted …
services : {
  message : @local::default_message
}
```

- Reduces the verbosity of *art*'s informational messages
- `@local::default_message` says
  - Find the definition of a parameter named `default_message`
  - Use that definition to initialize the parameter message
  - FHiCL will find the definition in the included file
  - It's a long, ugly definition, beyond the scope of this course
- More in Sesssion 22.