A FHiCL feature menagerie

Kyle J. Knoepfel

art Users Meeting

17 June 2016

Configuring your *art* job is ...

- Simple, in principle.
- In the context of an experiment's software infrastructure, it gets more complicated.

Configuring your *art* job is ...

- Simple, in principle.
- In the context of an experiment's software infrastructure, it gets more complicated.
- Tools exist that help you understand the details of an arbitrarily-complex configuration:
  - **`fhicl-dump`**
  - configuration description and validation suite

Configuring your *art* job is ...

- Simple, in principle.
- In the context of an experiment's software infrastructure, it gets more complicated.
- Tools exist that help you understand the details of an arbitrarily-complex configuration:
  - **`fhicl-dump`**
  - configuration description and validation suite
- More importantly, there are a set of guidelines to follow when developing configurations and configuration-aware C++ code, resulting in:
  - better maintainability
  - easier-to-understand configurations
  - ***less time debugging; more time on physics.***

Configuring your *art* job is ...

- Simple, in principle.
- In the context of an experiment's software infrastructure, it gets more complicated.
- Tools ex… …f an arbitrar…
  - `fhi…`
  - conf…
- More in… …llow when c… …aware C++ + code…
  - bette…
  - easier-to-understand configurations
  - ***less time debugging; more time on physics.***

Today:

- We will build a FHiCL file, listing and adopting best practices as we go.
- I will discuss **`fhicl-dump`**.
- I will introduce you to configuration validation and description.

The FHiCL file

- We will configure a producer: **`EventGenerator`**.
- We will configure an analyzer: **`ParticleViewer`**.

The FHiCL file

- We will configure a producer: **`EventGenerator`**.
- We will configure an analyzer: **`ParticleViewer`**.

```
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         ...
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         ...
      }
   }
}
```

The FHiCL file

- We will configure a producer: **`EventGenerator`**.
- We will configure an analyzer: **`ParticleViewer`**.

```
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         ...
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         ...
      }
   }
}
```

We won't worry about trigger paths and end path parameters right now.

The FHiCL file

- We will configure a producer: **`EventGenerator`**.
- We will configure an analyzer: **`ParticleViewer`**.

```
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         ...
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         ...
      }
   }
}
```

We won't worry about trigger paths and end path parameters right now.

Let's configure `generator` *and* `viewer` *to take a set of particle IDs.*

The FHiCL file

- Let's simulate e^-, μ^-, and τ^- leptons.
- Then let's see what they look like.

```
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         ...
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         ...
      }
   }
}
```

The FHiCL file

- Let's simulate e^-, μ^-, and τ^- leptons.
- Then let's see what they look like.

```
physics: {
    producers: {
        generator: {
            module_type: EventGenerator
            particles: ["e-","mu-","tau-"]
        }
    }

    analyzers: {
        viewer: {
            module_type: ParticleViewer
            particles: ["e-","mu-","tau-"]
        }
    }
}
```

The FHiCL file

- Let's simulate e^-, μ^-, and τ^- leptons.
- Then let's see what they look like.

```
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: ["e-","mu-","tau-"]
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: ["e-","mu-","tau-"]
      }
   }
}
```

Ugh. Multiple points of maintenance.

References

```
<COMMON PARAMETER>: ["e-","mu-","tau-"]

physics: {
    producers: {
        generator: {
            module_type: EventGenerator
            particles: <substitute value of COMMON PARAMETER>
        }
    }

    analyzers: {
        viewer: {
            module_type: ParticleViewer
            particles: <substitute value of COMMON PARAMETER>
        }
    }
}
```

References – substitutions (`@local`)

```
parameters: {
   particles: ["e-","mu-","tau-"]
}

physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: @local::parameters.particles
      }
   }
}
```

References – substitutions (`@local`)

```
parameters: {
   particles: ["e-","mu-","tau-"]
}

physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzer
      viewe
         mo
         particles: @local::parameters.particles
      }
   }
}
```

`@local` rules:

- referenced key must be previously defined
- referenced key must be fully qualified (e.g.):

  `@local::particles` would be a parse error

References – substitutions (`@local`)

```
parameters: {
   particles: ["e-","mu-","tau-"]
}

physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: @local::parameters.particles
      }
   }
}
```

To see fully processed configuration:

`fhicl-dump myConfig.fcl`

```
parameters: {
   particles: [
      "e-",
      "mu-",
      "tau-"
   ]
}
physics: {
   analyzers: {
      viewer: {
         module_type: "ParticleViewer"
         particles: [
            "e-",
            "mu-",
            "tau-"
         ]
      }
   }
   producers: {
      generator: {
         module_type: "EventGenerator"
         particles: [
            "e-",
            "mu-",
            "tau-"
         ]
      }
   }
}
```

References – substitutions (`@local`)

```
parameters: {
   particles: [
}

physics: {
   producers: {
      generator: {
         module_type: EventGenerato
         particles: @local::paramet
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewe
         particles: @local::paramet
      }
   }
}
```

Unwanted parameters:

```
parameters: {
   particles: [
      "e-",
      "mu-",
      "tau-"
   ]
}
physics: {
   analyzers: {
      viewer: {
         module_type: "ParticleViewer"
         particles: [
            "e-",
            "mu-",
            "tau-"
         ]
      }
   }
   producers: {
      generator: {
         module_type: "EventGenerator"
         particles: [
            "e-",
            "mu-",
            "tau-"
         ]
      }
   }
}
```

Prologs

```
parameters: {
   particles: ["e-","mu-","tau-"]
}

physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: @local::parameters.particles
      }
   }
}
```

Prologs

```
BEGIN_PROLOG
parameters: {
   particles: ["e-","mu-","tau-"]
}
END_PROLOG
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: @local::parameters.particles
      }
   }
}
```

Prolog parameters are not kept in the final configuration.

Prologs

```
BEGIN_PROLOG
parameters: {
   particles: ["e-","mu-","tau-"]
}
END_PROLOG
physics: {
   producers: {
      generator: {
                  @local::EventGenerato
                  @local::paramet
      }

   analyzers: {
      viewer: {
         module_type: ParticleViewe
         particles: @local::paramet
      }
   }
}
```

fhicl-dump myConfig.fcl

Prolog parameters are not kept in t

```
physics: {
   analyzers: {
      viewer: {
         module_type: "ParticleViewer"
         particles: [
            "e-",
            "mu-",
            "tau-"
         ]
      }
   }
   producers: {
      generator: {
         module_type: "EventGenerator"
         particles: [
            "e-",
            "mu-",
            "tau-"
         ]
      }
   }
}
```

Where we were

```
BEGIN_PROLOG
parameters: {
   particles: ["e-","mu-","tau-"]
}
END_PROLOG
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: @local::parameters.particles
      }
   }
}
```

#include facility

```
#include "parameters.fcl"
```

```
BEGIN_PROLOG
parameters: {
   particles: ["e-","mu-","tau-"]
}
END_PROLOG
```

```
physics: {
   producers: {
      generator: {
         module_type: EventGenerator
         particles: @local::parameters.particles
      }
   }

   analyzers: {
      viewer: {
         module_type: ParticleViewer
         particles: @local::parameters.particles
      }
   }
}
```

Using `#include`

- In order for a file to be included, its containing directory must be present on the **`FHICL_FILE_PATH`** environment variable.

Using `#include`

- In order for a file to be included, its containing directory must be present on the **`FHICL_FILE_PATH`** environment variable.
- The **`#include`** feature can be very convenient; but it can be easily abused.
- Guidelines:
  - Only **`#include`** files that contain only prologs.
    - Makes it much easier to understand the configuration-file structure.
  - Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would specify include directories for C++ headers.
    - For code you're developing it's better if you have to be explicit in your FHiCL file.

Extensible configuration

- We want to use our experiment's modules ***and*** our own. How do we do that?

Extensible configuration

- We want to use our experiment's modules ***and*** our own. How do we do that?

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
   producers: {
      p1: {...}
      p2: {...}
      p3: {...}
   }
   analyzers: {
      a1: {...}
      a2: {...}
      a3: {...}
   }
}
END_PROLOG
```

Extensible configuration

- We want to use our experiment's modules ***and*** our own. How do we do that?

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
   producers: {
      p1: {...}
      p2: {...}
      p3: {...}
   }
   analyzers: {
      a1: {...}
      a2: {...}
      a3: {...}
   }
}
END_PROLOG
```

```
#include "experiment.fcl"

physics: {
   producers: @local::experiment.producers
   analyzers: @local::experiment.analyzers
}
```

Cannot easily add own modules.

References – splicing

- **`@table`** and **`@sequence`** insert the *contents* of the value of the specified key.

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
   producers: {
      p1: {...}
      p2: {...}
      p3: {...}
   }
   analyzers: {
      a1: {...}
      a2: {...}
      a3: {...}
   }
}
END_PROLOG
```

References – splicing

- **`@table`** and **`@sequence`** insert the *contents* of the value of the specified key.

```
#include "experiment_modules.fcl"

physics: {
   producers: @local::experiment.producers
   analyzers: @local::experiment.analyzers
}
```

is equivalent to

```
#include "experiment_modules.fcl"

physics: {
   producers: { @table::experiment.producers }
   analyzers: { @table::experiment.analyzers }
}
```

References – splicing

- **`@table`** and **`@sequence`** insert the *contents* of the value of the specified key.

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
   producers: {
      p1: {...}
      p2: {...}
      p3: {...}
   }
   analyzers: {
      a1: {...}
      a2: {...}
      a3: {...}
   }
}
END_PROLOG
```

```
#include "experiment.fcl"

physics: {
   producers: {
      @table::experiment.producers
      p4: {...}
   }

   analyzers: {
      @table::experiment.analyzers
      a4: {...}
   }
}
```

References – splicing

- **`@table`** and **`@sequence`** insert the *contents* of the value of the specified key.

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
    ...
    producerPath: [p1, p2, p3]
    analyzerPath: [a1, a2, a3]
}
END_PROLOG
```

References – splicing

- **`@table`** and **`@sequence`** insert the *contents* of the value of the specified key.

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
    ...
    producerPath: [p1, p2, p3]
    analyzerPath: [a1, a2, a3]
}
END_PROLOG
```

```
#include "experiment.fcl"

physics: {
    producers: {
        @table::experiment.producers
        p4: {...}
    }

    analyzers: {
        @table::experiment.analyzers
        a4: {...}
    }
    producerPath: [@sequence::experiment.producerPath, p4]
    analyzerPath: [@sequence::experiment.analyzerPath, a4]
}
```

References – splicing

- **`@table`** and **`@sequence`** insert the *contents* of the value of the specified key.

```
# experiment.fcl
BEGIN_PROLOG
experiment: {
   ...
   producerPath: [p1, p2, p3]
   analyzerPath: [a1, a2, a3]
}
END_PROLOG
```

```
#include "experiment.fcl"

physics: {
   producers: {
      @table::experiment.producers
      ...
   }
   ar
      a4: {...}
   }
   producerPath: [@sequence::experiment.producerPath, p4]
   analyzerPath: [@sequence::experiment.analyzerPath, a4]
}
```

Primary benefit of splicing facilities:

- ***if experiment's configuration needs to change, user code stays the same.***

Assignment protection

- Syntax available to ignore value reassignment:

```
pi @protect_ignore: 3.14159
pi: 4.13159   # emit warning and ignore when parsed
```

- Can also trigger an error on value reassignment:

```
pi @protect_error: 3.14159
pi: 4.13159   # throw exception when parsed
```

- Useful not only for protecting parameters, but also for preserving a single point of maintenance (see Mu2e talk).

When no default will do …

- Sometimes there is no suitable default value for a configuration parameter. In such cases, the `@nil` symbol is appropriate.

```
source: {
   module_type: RootInput
   fileNames: @nil
}
```

- An attempt to retrieve the parameter is an error (*if no default specified in source code*):

```
using strings = vector<string>;
pset.get<strings>("fileNames"); // exception thrown
pset.get<strings>("fileNames", {"a.root"}); // OK, but not encouraged
```

Nested table pattern

- Use nested tables to group parameters for a common purpose.

Nested table pattern

- Use nested tables to group parameters for a common purpose.

Good

```
mod: {
   module_type: MyMod

   gen_settings: {
      a1: ...
      a2: ...
   }

   g4_settings: {
      b1: ...
      b2: ...
      b3: ...
   }

   c: ...
}
```

Nested table pattern

- Use nested tables to group parameters for a common purpose.

Good

```
mod: {
   module_type: MyMod

   gen_settings: {
      a1: ...
      a2: ...
   }

   g4_settings: {
      b1: ...
      b2: ...
      b3: ...
   }

   c: ...
}
```

Not good

```
mod: {
   module_type: MyMod


   a1: ...
   a2: ...



   b1: ...
   b2: ...
   b3: ...


   c: ...
}
```

Best practices

- Strive for a single point of maintenance in your FHiCL files (use references like **`@local`**).
- Only **`#include`** files that contain only prologs.
- Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would specify include directories for C++ header files.
- Group modules with a common purpose into one table and use the **`@table`** and **`@sequence`** splicing facilities.
- Use the nested table pattern.

Best practices

- Strive for a single point of maintenance in your FHiCL files (use references like **`@local`**).
- Only **`#include`** files that contain only prologs.
- Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would specify include directories for C++ header files.
- Group modules with a common purpose into one table and use the **`@table`** and **`@sequence`** splicing facilities.
- Use the nested table pattern.
- Do not put physics defaults in C++ code.

Best practices

- Strive for a single point of maintenance in your FHiCL files (use references like **`@local`**).
- Only **`#include`** files that contain only prologs.
- Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would specify include directories for C++ header files.
- Group modules with a common purpose into one table and use the **`@table`** and **`@sequence`** splicing facilities.
- Use the nested table pattern.
- Do not put physics defaults in C++ code.

```
pset.get<double>("energyThreshold", 20.);  // BAD
pset.get<int   >("verbosityLevel", 2);     // good
pset.get<double>("timingWindowStart");     // good
```

Facility to assist job configuration

- **`fhicl-dump`** prints the fully-processed FHiCL file
  - **`--help`** for options
  - `#includes` expanded,
  - referenced variables substituted,
  - comments omitted, etc.
  - **Can also include source information.**

Facility to assist job configuration

- **`fhicl-dump`** prints the fully-processed FHiCL file
  - **`--help`** for options
  - `#includes` expanded,
  - referenced variables substituted,
  - comments omitted, etc.
  - **Can also include source information.**

```
# Produced from 'fhicl-dump' using:
#   Input  : art/test/Integration/ProductMix_r1a.d/ProductMix_r1a.fcl
#   Policy : cet::filepath_maker
#   Path   : "FHICL_FILE_PATH"

outputs: {  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1a.fcl:6
}
physics: {  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1.fcl:4
   analyzers: {  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1a.fcl:5
   }
   e1: [  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1a.fcl:7
   ]
   end_paths: [  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1a.fcl:8
   ]
   filters: {  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1.fcl:6
      mixFilter: {  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1.fcl:8
         expectedRespondFunctionCalls: 2  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/Prod
         fileNames: [  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/ProductMix_r1.fcl:12
            "../ProductMix_w.d/mix.root"  # /home/knoepfel/scratch/build-art-prof/art/test/Integration/ProductMix_r1a.d/Prod
         ]
```

Configuration validation & description

- **A common frustration:**

  What is the allowed configuration for a given module?

- **One solution: look at the source code.**

  1. *Okay…where is it?*
  2. *I'm a newcomer. Do I have to look at (potentially complicated) C++ to figure out how to configuration a presumably simple job?*

- Better solution:

  Devise a system that documents itself, providing description and validation capabilities.

What configurations are described/validated?

- The following facilities have description/validation enabled:
  - all *art*-provided modules (including **`RootOutput`**)
  - all *art*-provided services except **`floating_point_control`** and **`message`**.
  - **`EmptyEvent`** and **`RootInput`** sources.
  - Any of your **`module`**s, **`service`**s, or **`plugin`**s for which you want description/validation.

What configurations are described/validated?

- The following facilities have description/validation enabled:
  - all *art*-provided modules (including **`RootOutput`**)
  - all *art*-provided services except **`floating_point_control`** and **`message`**.
  - **`EmptyEvent`** and **`RootInput`** sources.
  - Any of your **`module`**s, **`service`**s, or **`plugin`**s for which you want description/validation.

- **Documentation:**
  - ***See the back-up slides.***
  - https://cdcvs.fnal.gov/redmine/projects/art/wiki/Configuration_validation_and_description
  - https://cdcvs.fnal.gov/redmine/projects/fhicl-cpp/wiki/Configuration_validation_and_fhiclcpp_types

Conclusion

- We've discussed issues related to job configurations:
  - Best practices for FHiCL configuration design
  - **`fhicl-dump`**
  - Configuration description and validation

Conclusion

- We've discussed issues related to job configurations:
  - Best practices for FHiCL configuration design
  - **`fhicl-dump`**
  - Configuration description and validation

- **Bottom line**: don't settle on what's expeditious. Think about your design, choose methods that:
  - Favor ease of maintenance
  - Make it simple for you and (more importantly) others to understand
  - **Allow you to spend more time on physics!**

Conclusion

- We've discussed issues related to job configurations:
  - Best practices for FHiCL configuration design
  - **`fhicl-dump`**
  - Configuration description and validation

- **Bottom line**: don't settle on what's expeditious. Think about your design, choose methods that:
  - Favor ease of maintenance
  - Make it simple for you and (more importantly) others to understand
  - **Allow you to spend more time on physics!**

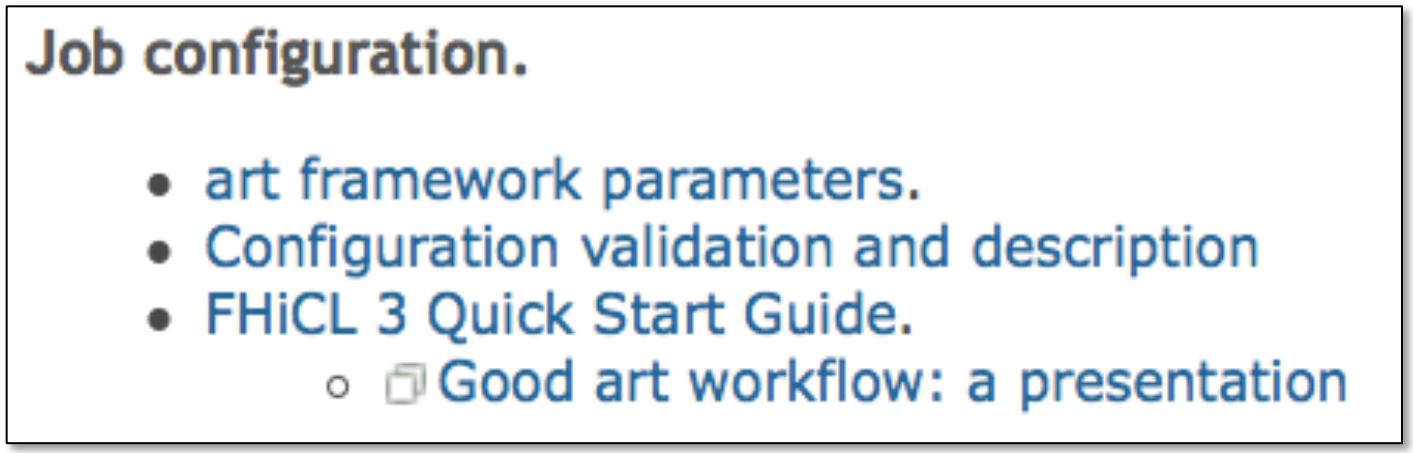
Thank you.

Extra slides

Today's material:

- https://cdcvs.fnal.gov/redmine/projects/art/wiki/

Scroll down until:



Difference between FHiCL and `fhiclcpp`

- FHiCL – Fermilab Hierarchical Configuration Language
  - *art* configuration files are written in FHiCL

Difference between FHiCL and `fhiclcpp`

- FHiCL – Fermilab Hierarchical Configuration Language
  - *art* configuration files are written in FHiCL

```
process_name: MultiInputMerge

physics.stream1: [out1]

outputs.out1: {
   module_type: RootOutput
   fileName: "out_%#.root”
   fileSwitch: {
      boundary: InputFile
      force: true
   }
}
```

Difference between FHiCL and `fhiclcpp`

- FHiCL – Fermilab Hierarchical Configuration Language
  - *art* configuration files are written in FHiCL

- **`fhiclcpp`** – The C++ binding to FHiCL, enabling access to the configuration.

Difference between FHiCL and `fhiclcpp`

- FHiCL – Fermilab Hierarchical Configuration Language
  - *art* configuration files are written in FHiCL

- **`fhiclcpp`** – The C++ binding to FHiCL, enabling access to the configuration.

```
TestOutputModule::TestOutputModule(fhicl::ParameterSet const& ps):
  art::OutputModule{ps},
  name_{ps.get<std::string>("name")},
  bitMask_{ps.get<int>("bitMask")},
  expectTriggerResults_{ps.get<bool>("expectTriggerResults", true)}
{
}
```

Difference between FHiCL and `fhiclcpp`

- FHiCL – Fermilab Hierarchical Configuration Language
  - *art* configuration files are written in FHiCL

- **`fhiclcpp`** – The C++ binding to FHiCL, enabling access to the configuration.

```
TestOutputModule::TestOutputModule(fhicl::ParameterSet const& ps):
  art::OutputModule{ps},
  name_{ps.get<std::string>("name")},
  bitMask_{ps.get<int>("bitMask")},
  expectTriggerResults_{ps.get<bool>("expectTriggerResults", true)}
{
}
```

Using `#include`

- In order for a file to be included, its containing directory must be present on the `FHICL_FILE_PATH` environment

```
#include "simulation_services.fcl"
#include "generation_services.fcl"

physics: { ... }
```

  - Specify directories on your `FHICL_FILE_PATH` in a manner similar to how you would specify include directories for C++ headers.
    - For code you're developing it's better if you have to be explicit in your FHiCL file.

Using `#include`

- In order for a file to be included, its containing directory must be present on the `FHICL_FILE_PATH` environment

```
#include "simulation_services.fcl"
#include "generation_services.fcl"

physics: { ... }
```

```
FHICL_FILE_PATH=
    package/package/simulation:
    package/package/simulation/generation
```

  - Specify directories on your `FHICL_FILE_PATH` in a manner similar to how you would specify include directories for C++ headers.
    - For code you're developing it's better if you have to be explicit in your FHiCL file.

Removing undesired parameters

- Occasionally, you may need to remove undesired parameters—can occur if using the substitutions.

```
BEGIN_PROLOG
default_services: {
   MemoryTracker: {...}
   TimeTracker: {...}
   Tracer: {...}
   message: {...}
}
END_PROLOG

services: {
   @table::default_services
   Tracer: @erase
   MyService: {...}
}
```

Removing undesired parameters

- Occasionally, you may need to remove undesired parameters—can occur if using the substitutions.

```
BEGIN_PROLOG
default_services: {
   MemoryTracker: {...}
   TimeTracker: {...}
   Tracer: {...}
   message: {...}
}
END_PROLOG

services: {
   @table::default_services
   Tracer: @erase
   MyService: {...}
}
```

```
services: {
   MemoryTracker: {...}
   TimeTracker: {...}
   message: {...}
   MyService: {...}
}
```

Using `#include`

- In order for a file to be included, its containing directory must be present on the `FHICL_FILE_PATH` environment
  - Specify directories on your `FHICL_FILE_PATH` in a manner

```
#include "simulation_services.fcl"
#include "generation_services.fcl"

physics: { ... }
```

```
FHICL_FILE_PATH=
    package/package/simulation:
    package/package/simulation/generation
```

```
FHICL_FILE_PATH=package
```

```
#include "package/simulation/generation/
generation_services.fcl"
#include "package/simulation/simulation_services.fcl"

physics: { ... }
```

Removing undesired parameters

- Occasionally, you may need to remove undesired parameters—can occur if using the substitutions.

```
BEGIN_PROLOG
default_services: {
   MemoryTracker: {...}
   TimeTracker: {...}
   Tracer: {...}
   message: {...}
}
END_PROLOG

services: {
   @table::default_services
   Tracer: @erase
   MyService: {...}
}
```

```
services: {
   MemoryTracker: {...}
   TimeTracker: {...}
   message: {...}
   MyService: {...}
}
```

Use judiciously:
frequent use of `@erase` implies a factorization problem.

Nested table pattern

- How you structure the allowed configuration influences how you design your module, and vice versa.
- Design the expected configuration in a way that is:
  - amenable to good C++ usage
  - robust against future changes (i.e. maintainable)
  - easy to understand
- Avoid “flat” configuration designs—i.e. use the nested table pattern.
- For example ...

Nested table pattern

- Your module:

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(...);  // Needs parameters 'a1' and 'a2'
  g(...);  // Needs parameters 'b1', 'b2', and 'b3'
  h(...);  // Needs parameter  'c'
}
```

Nested table pattern

- Your module:

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(...);  // Needs parameters 'a1' and 'a2'
  g(...);  // Needs parameters 'b1', 'b2', and 'b3'
  h(...);  // Needs parameter  'c'
}
```

```
mod: {
   module_type: MyMod

   a1: ...
   a2: ...

   b1: ...
   b2: ...
   b3: ...

   c: ...
}
```

Nested table pattern

- Your module:

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(...);  // Needs parameters 'a1' and 'a2'
  g(...);  // Needs parameters 'b1', 'b2', and 'b3'
  h(...);  // Needs parameter  'c'
}
```

```
mod: {
   module_type: MyMod

   a1: ...
   a2: ...

   b1: ...
   b2: ...
   b3: ...

   c: ...
}
```

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(pset); // 'f' has more info than it needs
  g(pset); // 'g' ""
  h(pset.get<T>("c"));
}
```

Nested table pattern

- Your module:

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(...);  // Needs parameters 'a1' and 'a2'
           // parameters 'b1', 'b2', and 'b3'
           // parameter  'c'
```

```
mod: {
   module_type: MyMod

   a: {
      a1: ...
      a2: ...
   }

   b: {
      b1: ...
      b2: ...
      b3: ...
   }

   c: ...
}
```

Nested table pattern

- Your module:

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(...);  // Needs parameters 'a1' and 'a2'
           // rameters 'b1', 'b2', and 'b3'
           // rameter  'c'
```

```
mod: {
   module_type: MyMod

   a: {
      a1: ...
      a2: ...
   }

   b: {
      b1: ...
      b2: ...
      b3: ...
   }

   c: ...
}
```

```
MyMod::MyMod(fhicl::ParameterSet const&
pset)
{
  f(pset.get<ParameterSet>("a"));
  g(pset.get<ParameterSet>("b"));
  h(pset.get<T>("c"));
}
```

Nested table pattern

- Your module:

```
MyMod::MyMod(fhicl::ParameterSet const& pset)
{
  f(...);  // Needs parameters 'a1' and 'a2'
           // parameters 'b1', 'b2', and 'b3'
           // parameter  'c'
```

```
mod: {
   module_type: MyMod

   a: {
      a1: ...
      a2: ...
   }

   b: {
      b1: ...
      b2: ...
      b3: ...
   }

   c: ...
}
```

```
using Parameters = EDAnalyzer::Table<Config>;
MyMod::MyMod(Parameters const& ps)
{
  f(ps().a());
  g(ps().b());
  h(ps().c());
}
```

Preliminaries

Atom: a value with no underlying structure.

```
module_type: ParticleViewer
```

Sequence: a list of values.

```
pdgIDs: [11,13,15]
```

Table: a collection of name-value pairs.

```
prod1: {
    module_type: ParticleViewer
    pdgIDs: [11,13,15]
}
```

art --help

```
$ art --help

  Usage: art <-c <config-file>> <other-options> [<source-file>]+

  Basic options:
    -h [ --help ]                         produce help message
    --version                             Print art version (2.01.00)
    -c [ --config ] arg                   Configuration file.
    --process-name arg                    art process name.
    --print-available arg                 List all available plugins with the
                                          provided suffix.  Choose from:
                                              'module'
                                              'plugin'
                                              'service'
                                              'source'
    --print-available-modules             List all available modules that can be
                                          invoked in a FHiCL file.
    --print-available-services            List all available services that can be
                                          invoked in a FHiCL file.
    --print-description arg               Print description of specified module,
                                          service, source, or other plugin
                                          (multiple OK).
  ...
  ...
  ...
```

art --help

```
$ art --help

  Usage: art <-c <config-file>> <other-options> [<source-file>]+

  Basic options:
    -h [ --help ]                         produce help message
    --version                             Print art version (2.01.00)
    -c [ --config ] arg                   Configuration file.
    --process-name arg                    art process name.
    --print-available arg                 List all available plugins with the
                                          provided suffix.  Choose from:
                                              'module'
                                              'plugin'
                                              'service'
                                              'source'
    --print-available-modules             List all available modules that can be
                                          invoked in a FHiCL file.
    --print-available-services            List all available services that can be
                                          invoked in a FHiCL file.
    --print-description arg               Print description of specified module,
                                          service, source, or other plugin
                                          (multiple OK).
  ...
  ...
  ...
```

art --print-available-modules

```
================================================================================================================================

   module_type               Type        Source location
--------------------------------------------------------------------------------------------------------------------------------
   AddIntsProducer           producer    /home/knoepfel/art/art/test/Integration/AddIntsProducer_module.cc
   AssembleProducts          producer    /home/knoepfel/art/art/test/Integration/product-aggregation/AssembleProducts_module.cc
   AssnsAnalyzer             analyzer    /home/knoepfel/art/art/test/Integration/AssnsAnalyzer_module.cc
   AssnsProducer             producer    /home/knoepfel/art/art/test/Integration/AssnsProducer_module.cc
   BlockingPrescaler         filter      /home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
   CheckProducts             analyzer    /home/knoepfel/art/art/test/Integration/product-aggregation/CheckProducts_module.cc
   CompressedIntProducer     producer    /home/knoepfel/art/art/test/Integration/CompressedIntProducer_module.cc
   .
   .
   .
   InputProducer             producer    /home/knoepfel/art/art/test/Integration/event-shape/InputProducer_module.cc
***InputProducerNoEvents     producer    /home/knoepfel/art/art/test/Integration/event-shape/InputProducerNoEvents_module.cc
***InputProducerNoEvents     producer    /home/knoepfel/art/art/test/Integration/run-subrun-shape/
InputProducerNoEvents_module.cc
   InputProducerOnlyEvents   producer    /home/knoepfel/art/art/test/Integration/event-shape/InputProducerOnlyEvents_module.cc
   .
   .
   .
```

art --print-available-modules

```
================================================================================================================================

   module_type               Type        Source location
--------------------------------------------------------------------------------------------------------------------------------
   AddIntsProducer           producer    /home/knoepfel/art/art/test/Integration/AddIntsProducer_module.cc
   AssembleProducts          producer    /home/knoepfel/art/art/test/Integration/product-aggregation/AssembleProducts_module.cc
   AssnsAnalyzer             analyzer    /home/knoepfel/art/art/test/Integration/AssnsAnalyzer_module.cc
   AssnsProducer             producer    /home/knoepfel/art/art/test/Integration/AssnsProducer_module.cc
   BlockingPrescaler         filter      /home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
   CheckProducts             analyzer    /home/knoepfel/art/art/test/Integration/product-aggregation/CheckProducts_module.cc
   CompressedIntProducer     producer    /home/knoepfel/art/art/test/Integration/CompressedIntProducer_module.cc
   .
   .
   .
   InputProducer             producer    /home/knoepfel/art/art/test/Integration/event-shape/InputProducer_module.cc
***InputProducerNoEvents     producer    /home/knoepfel/art/art/test/Integration/event-shape/InputProducerNoEvents_module.cc
***InputProducerNoEvents     producer    /home/knoepfel/art/art/test/Integration/run-subrun-shape/
InputProducerNoEvents_module.cc
   InputProducerOnlyEvents   producer    /home/knoepfel/art/art/test/Integration/event-shape/InputProducerOnlyEvents_module.cc
   .
   .
   .
```

To get information on a module (e.g.):

```
art --print-description AddIntsProducer
```

art --print-description AddIntsProducer

```
================================================================================================

    module_type : AddIntsProducer (or "art/test/Integration/AddIntsProducer")

        provider: art
        type    : producer
        source  : /home/knoepfel/art/art/test/Integration/AddIntsProducer_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_test_Integration_AddIntsProducer_module.so

    Allowed configuration
    ---------------------

        [ None provided ]

================================================================================================
```

art --print-description AddIntsProducer

```
================================================================================================

    module_type : AddIntsProducer (or "art/test/Integration/AddIntsProducer")

        provider: art
        type    : producer
        source  : /home/knoepfel/art/art/test/Integration/AddIntsProducer_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_test_Integration_AddIntsProducer_module.so

    Allowed configuration
    ---------------------

        [ None provided ]

================================================================================================
```

If configuration description enabled for the module/plugin, the allowed configuration is printed...

art --print-description BlockingPrescaler

```
================================================================================================

    module_type : BlockingPrescaler (or "art/Framework/Modules/BlockingPrescaler")

        provider: art
        type    : filter
        source  : /home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_Framework_Modules_BlockingPrescaler_module.so

    Allowed configuration
    ---------------------

        <module_label>: {

           module_type: BlockingPrescaler

           errorOnFailureToPut: true  # default

           blockSize: 1  # default

           stepSize: <unsigned long>

           offset: 0  # default
        }

================================================================================================
```

art --print-description BlockingPrescaler

```
================================================================================================

    module_type : BlockingPrescaler (or "art/Framework/Modules/BlockingPrescaler")

        provider: art
        type    : filter
        source  : /home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_Framework_Modules_BlockingPrescaler_module.so

    Allowed configuration
    ---------------------

        <module_label>: {

           module_type: BlockingPrescaler

           errorOnFailureToPut: true  # default      (added by art)

           blockSize: 1  # default                   (default value provided)

           stepSize: <unsigned long>                 (default not provided, must be convertible to C++ std::size_t)

           offset: 0  # default
        }

================================================================================================
```

art --print-description BlockingPrescaler

```
================================================================================================

    module_type : BlockingPrescaler (or "art/Framework/Modules/BlockingPrescaler")

        provider: art
        type    : filter
        source  : /home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_Framework_Modules_BlockingPrescaler_module.so

    Allowed configuration
    ---------------------

        <module_label>: {

           module_type: BlockingPrescaler

           errorOnFailureToPut: true  # default

           blockSize: 1  # default

           stepSize: <unsigned long>

           offset: 0  # default
        }

================================================================================================
```

Let's use **`BlockingPrescaler`** *in a job.*

Our FHiCL file

```
# test.fcl
physics: {
   filters: {
      f1: {
         module_type: BlockingPrescaler
         blocksize: 2
      }
   }
   p1: [f1]
}
```

Our FHiCL file

```
# test.fcl
physics: {
   filters: {
      f1: {
         module_type: BlockingPrescaler
         blocksize: 2
      }
   }
   p1: [f1]
}
```

- Some errors:
  - missing "`stepSize`"
  - "`blockSize`" not "`blocksize`"

Our FHiCL file

```
# test.fcl
physics: {
   filters: {
      f1: {
         module_type: BlockingPrescaler
         blocksize: 2
      }
   }
   p1: [f1]
}
```

- Some errors:
  - missing "`stepSize`"
  - "`blockSize`" not "`blocksize`"

Without validation:

art throws an exception for parameters without a default in the source code.

Our FHiCL file

```
# test.fcl
physics: {
   filters: {
      f1: {
         module_type: BlockingPrescaler
         blocksize: 2
      }
   }
   p1: [f1]
}
```

- Some errors:
  - missing "`stepSize`"
  - "`blockSize`" not "`blocksize`"

Without validation:

art continues without incident, using the default value for **`blockSize`**, not the intended user-provided one.

With validation:

```
art -c test.fcl
```

```
---- Configuration BEGIN

  ================================================================================================

  !! The following modules have been misconfigured: !!

  ------------------------------------------------------------------------------------------------


  Module label: f1
  module_type : BlockingPrescaler

  Missing parameters:

   -    stepSize: <unsigned long>

  Unsupported parameters:

   + blocksize                      [ ./test.fcl:6 ]


  ================================================================================================

---- Configuration END
```

What configurations are described/validated?

- The following facilities have description/validation enabled:
  - all *art*-provided modules (including **`RootOutput`**)
  - all *art*-provided services except **`floating_point_control`** and **`message`**.
  - **`EmptyEvent`** and **`RootInput`** sources.
  - Any of your **`module`**s, **`service`**s, or **`plugin`**s for which you want description/validation.

art --print-description InputProducerNoEvents

```
================================================================================================

    module_type : InputProducerNoEvents (or "art/test/Integration/event-shape/InputProducerNoEvents")

        provider: art
        type    : producer
        source  : /home/knoepfel/art/art/test/Integration/event-shape/InputProducerNoEvents_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_test_Integration_event-shape_InputProducerNoEvents_module.so

    Allowed configuration
    ---------------------

        [ None provided ]

================================================================================================

    module_type : InputProducerNoEvents (or "art/test/Integration/run-subrun-shape/InputProducerNoEvents")

        provider: art
        type    : producer
        source  : /home/knoepfel/art/art/test/Integration/run-subrun-shape/InputProducerNoEvents_module.cc
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_test_Integration_run-subrun-shape_InputProducerNoEvents_module.so

    Allowed configuration
    ---------------------

        [ None provided ]

================================================================================================
```

art --print-description InputProducerNoEvents

```
================================================================================================

    module_type : InputProducerNoEvents (or "art/test/Integration/event-shape/InputProducerNoEvents")

        provider: art
        type    : p
        source  : /
        library : /home/knoepfel/scratch/build-art-prof/lib/libart_test_Integration_event-shape_InputProducerNoEvents_module.so

    Allowed configuration
    ---------------------

        [ None provided ]

================================================================================================

    module_type : InputProducerNoEvents (or "art/test/Integration/run-subrun-shape/InputProducerNoEvents")

        provider: art
        type    : producer
        source  : /
        library : /                                                                    le.so

    Allowed config
    --------------

        [ None provided ]

================================================================================================
```

"art/test/Integration/event-shape/InputProducerNoEvents"

"art/test/Integration/run-subrun-shape/
InputProducerNoEvents"

Using the long specification disambiguates between modules.

✗ No description
✗ No validation

```
class MyProducer : public art::EDProducer {
public:


  explicit MyProducer(fhicl::ParameterSet const&);

};
```

✓ Yes description
✗ No validation

```
struct Config {
  ...
};

class MyProducer : public art::EDProducer {
public:

  using Parameters = Table<Config>;
  explicit MyProducer(fhicl::ParameterSet const&);

};
```

✓ Yes description
✗ No validation

```
struct Config {
  ...
};
```

The allowed configuration for **`MyProducer`**.

```
class MyProducer : public art::EDProducer {
public:

  using Parameters = Table<Config>;
  explicit MyProducer(fhicl::ParameterSet const&);

};
```

✓ Yes description
✗ No validation

```
struct Config {
  ...
};
```

The allowed configuration for **`MyProducer`**.

```
class MyProducer : public art::EDProducer {
public:

  using Parameters = Table<Config>;
  explicit MyProducer(fhicl::ParameterSet const&);

};
```

art looks for **`Parameters`** in your module to gain access to the description.

✓ Yes description
✗ No validation

```
struct Config {
  ...
};
```

```
class MyProducer : public art::EDProducer {
public:

  using Parameters = Table<Config>;
  explicit MyProducer(fhicl::ParameterSet const&);

};
```

✓ Yes description
✓ Yes validation

```
struct Config {
  ...
};
```

```
class MyProducer : public art::EDProducer {
public:

  using Parameters = Table<Config>;
  explicit MyProducer(Parameters const&);

};
```

What do you put in the allowed configuration?

```
struct Config {
  Atom<double> energyCutoff { Name("energyCutoff") };
  Atom<bool> verbose { Name("verbose"), false };
  Sequence<int> numbers { Name("numbers"), {1,-2,3} };
};
```

What do you put in the allowed configuration?

```
struct Config {
  Atom<double> energyCutoff { Name("ener
  Atom<bool> verbose { Name("verbose"),
  Sequence<int> numbers { Name("numbers"
};
```

```
art --print-description
MyProducer
```

```
Allowed configuration
----------------------

<module_label>: {

   module_type: MyProducer

   errorOnFailureToPut: true  # default

   energyCutoff: <double>

   verbose: false  # default

   numbers: [
      1,  # default
      -2,  # default
      3  # default
   ]
}
```

What do you put in the allowed configuration?

```
struct Config {
  Atom<double> energyCutoff { Name("energyCutoff") };
  Atom<bool> verbose { Name("verbose"), false };
  Sequence<int> numbers { Name("numbers"), {1,-2,3} };
};
```

You can add nested tables.

What do you put in the allowed configuration?

```
struct Config {
  Atom<double> energyCutoff { Name("energyCutoff") };
  Atom<bool> verbose { Name("verbose"), false };
  Sequence<int> numbers { Name("numbers"), {1,-2,3} };

  struct Settings {
    Atom<double> stepSize { Name("stepSize") };
    Atom<double> density { Name("density"),
                           Comment("Density should be in g/cm^3.") };
  };
  Table<Settings> settings { Name("settings") };
};
```

What do you put in the allowed configuration?

```
struct Config {
  Atom<double> energyCutoff { Name("en
  Atom<bool> verbose { Name("verbose")
  Sequence<int> numbers { Name("number

  struct Settings {
    Atom<double> stepSize { Name("step
    Atom<double> density { Name("densi
                           Comment("De
  };
  Table<Settings> settings { Name("set
};
```

```
art --print-description
MyProducer
```

```
Allowed configuration
---------------------

<module_label>: {

   module_type: MyProducer

   errorOnFailureToPut: true  # default

   energyCutoff: <double>

   verbose: false  # default

   numbers: [
      1,  # default
      -2,  # default
      3  # default
   ]

   settings: {

      stepSize: <double>

      # Density should be in g/cm^3.

      density: <double>
   }
}
```