Session 22: Good *art* Workflow

or

Guidelines for better job configuration

Kyle J. Knoepfel

Fermi National Accelerator Laboratory

Aug. 6, 2015

By now …

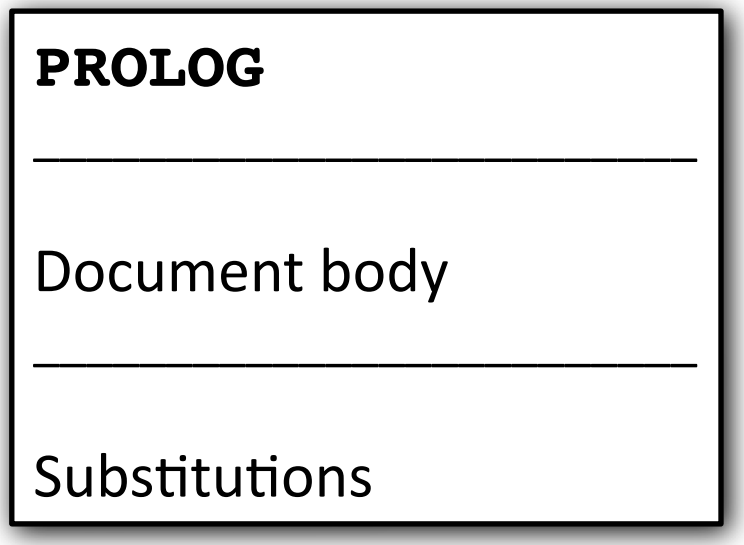
- You know how to configure simple *art* jobs using the FHiCL language
- Now it's time to design your configurations to be robust:
  - against future configuration changes
  - for submitting jobs to the grid

By now …

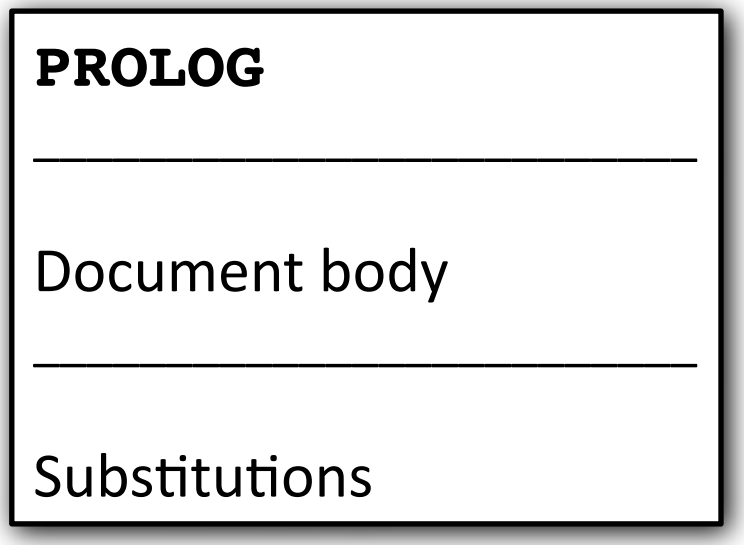
- You know how to configure simple *art* jobs using the FHiCL language
- Now it's time to design your configurations to be robust:
  - against future configuration changes
  - for submitting jobs to the grid

- We will learn:
  - Various reporting tools
  - **`PROLOG`**s and substitutions
  - How to minimize points of maintenance
  - **`FHICL_FILE_PATH`** and **`#include`**
  - Various FHiCL syntax issues

Layout of FHiCL document



Layout of FHiCL document



- The final configuration does not contain any **`PROLOG`** or substitutions.

- To check the final configuration*:

```
fhicl-dump <your FHiCL file>
```

*`fhicl-dump` has been provided in `art-workbook` for this course.
In the near future, it will be relocated in `fhicl-cpp`.

Substitutions

```
table1: {
    table2: {
        foo : bar
    }
}
```

Substitutions

```
table1: {
    table2: {
        foo : bar
    }
}

// substitutions
table1.table2.foo : ey
```

Substitutions

```
table1: {
    table2: {
        foo : bar
    }
}

// substitutions
table1.table2.foo : ey
```

Substitutions must be fully-qualified

Substitutions

```
table1:
    table
        fo
    }
}

// subst
table1.t
```

You **cannot** substitute partially-qualified parameters

```
table1: {
    table2: {
        foo : bar
    }
    table2.foo : ey // THIS WILL BE
                    // A PARSE ERROR
}
```

Substitutions must be fully-qualified

Substitutions

```
table1: {
    table2: {
        foo : bar
    }
    list : [ 0, 4, "fun" ]
}

// substitutions
table1.table2.foo : ey
```

Substitutions

```
table1: {
    table2: {
        foo : bar
    }
    list : [ 0, 4, "fun" ]
}

// substitutions
table1.table2.foo : ey
table1.list[2] : 7
```

Substitutions

```
table1: {
   table2: {
      foo : bar
   }
   list : [ 0, 4, "fun" ]
}

// substitution
table1.table2.f
table1.list[2]
```

Output from **fhicl-dump**

```
table1: { list: [ 0
                , 4
                , 7
                ]
          table2: { foo: "ey"
                    }
        }
```

Workflow best practices

1. Strive for a **Single Point of Maintenance** (SPoM) in your FHiCL files.

Workflow best practices

1. Strive for a **Single Point of Maintenance** (SPoM) in your FHiCL files.

   - Whenever we code in C++, we design our programs in a way so that they can be minimally burdensome to maintain.
   - We want to do the same thing in FHiCL …

SPoM: Illustrated

```
physics : {

   producers : {
      prod1  : { pdgId : 11 }
      prod2  : { pdgId : 11 }
   }

   analyzers : {
      mod1   : { pdgId : 11 }
   }

}
```

SPoM: Illustrated (`@local`)

```
pid : 11
physics : {

   producers : {
      prod1  : { pdgId : @local::pid }
      prod2  : { pdgId : @local::pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::pid }
   }

}
```

SPoM: Illustrated (`@local`)

```
pid : 11
physics : {

   producers : {
      prod1  : { pdgId : @local::pid }
      prod2  : { pdgId : @local::pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::pid }
   }

}
```

- **`@local`** rules:
  - referenced parameter must be defined previously
  - referenced parameter must be fully qualified

SPoM: Illustrated (**`@local`**)

```
parameters : {
   pid : 11
}
physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

SPoM: Illustrated (**`@local`**)

```
parameters : {
   pid : 11
}
physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

`@local::pid` would be a parse error.

SPoM: Illustrated

```
parameters : {
   pid : 11
}
physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

- Now we have unwanted FHiCL parameters in our configuration:
  - **`parameters`**
  - **`parameters.pid`**

SPoM: Illustrated

```
parameters : {
   pid : 11
}
physics : {

   producers : {
      prod1  : { pdg
      prod2  : { pdg
   }

   analyzers : {
      mod1   : { pdg
   }

}
```

- Now we have unwanted FHiCL parameters in our configuration:
  - **`parameters`**
  - **`parameters.pid`**

```
// final configuration after FHiCL processing
parameters: { pid: 11
              }
physics: { analyzers: { mod1: { pdgId: 11
                                }
                      }
           producers: { prod1: { pdgId: 11
                                 }
                        prod2: { pdgId: 11
                                 }
                      }
         }
```

SPoM: Illustrated (**PROLOG**)

```
BEGIN_PROLOG
parameters : {
   pid : 11
}
END_PROLOG

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

SPoM: Illustrated (**PROLOG**)

```
BEGIN_PROLOG
parameters : {
   pid : 11
}
END_PROLOG

physics : {

   producers : {
      prod1  : { pd
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

- **PROLOG**s define parameters:
  - at the beginning of the document (always!)
  - that can be used later on in the **PROLOG**/ rest of the document
  - that *are not included* as part of the final FHiCL file.
- Multiple **PROLOG**s are supported as long as they are contiguous.

SPoM: Illustrated (**PROLOG**)

```
BEGIN_PROLOG
parameters : {
   pid : 11
}
END_PROLOG

physics : {

   producers : {
      prod1  : { pd
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers
      mod1
   }

}
```

- **PROLOG**s define parameters:
  - at the beginning of the document (always!)
  - that can be used later on in the **PROLOG**/ rest of the document
  - that *are not included* as part of the final FHiCL file.
- Multiple **PROLOG**s are supported as long as they are contiguous.

```
// final configuration after FHiCL processing
physics: { analyzers: { mod1: { pdgId: 11
                              }
                      }
           producers: { prod1: { pdgId: 11
                               }
                        prod2: { pdgId: 11
                               }
                      }
         }
```

SPoM: Illustrated (**PROLOG**)

```
BEGIN_PROLOG
parameters : {
   pid : 11
}
END_PROLOG

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

SPoM: Illustrated (**PROLOG**)

```
BEGIN_PROLOG
parameters : {
   pid : 11
}
END_PROLOG

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

- Can place **PROLOG** in separate file.

SPoM: Illustrated (`#include`)

```
#include "prolog_parameters.fcl"

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

Workflow best practices

1. Strive for a **single point of maintenance** (SPoM) in your FHiCL files.
2. Only **`#include`** files that contain **`PROLOG`**s.

Workflow best practices

1. Strive for a **single point of maintenance** (SPoM) in your FHiCL files.
2. Only **`#include`** files that contain **`PROLOG`**s.

   - Adopting this principle makes a FHiCL configuration file:
     - easiest to understand, and
     - hardest to get wrong
   - Some of the exercises you will go through deal with configurations that violate this principle.

`#include` and `FHICL_FILE_PATH`

- The **`#include`** syntax is very similar in behavior to the C++ include directive.
  - Any **`#include`**d file is expanded in place before the FHiCL file is parsed.
  - Nested includes are allowed.
  - ➢ In order for the file to be included, its containing directory *must be present* on the **`FHICL_FILE_PATH`** environment variable. If not, you will get an error like:

```
---- search_path BEGIN
  Can't find file "databaseFiles/pdt.fcl"
---- search_path END
```

(see extra slides)

Workflow best practices

1. Strive for a **single point of maintenance** (SPoM) in your FHiCL files.
2. Only **`#include`** files that contain **`PROLOG`**s.
3. Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would include C++ headers.

Workflow best practices

1. Strive for a **single point of maintenance** (SPoM) in your FHiCL files.
2. Only **`#include`** files that contain **`PROLOG`**s.
3. Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would include C++ headers.

    - Too many **`FHICL_FILE_PATH`** directories make it difficult to locate configuration files.

Aside: An expanded FHiCL file

- To see a fully-expanded FHiCL file—i.e. all included files have been expanded—type:

```
fhicl-expand –o=expanded.fcl config.fcl
```

```
#include "fcl/minimalMessageService.fcl"
#include "fcl/standardModules.fcl"

process_name : SampleFHiCLFile

source : {
  module_type : EmptyEvent
  maxEvents   : 10
}

services : { @table::services }

physics  : {
   producers : {
      evtgen      : @local::evtgen_default
      detsim      : @local::detsim_default
      makeHits    : @local::makeHits_default
      fitter      : @local::fitter_default
      randomsaver : @local::randomsaver_default
   }

   analyzers : {
      inspectGens : @local::inspectGens_default
   }

   p1            : [ evtgen, detsim, makeHits, fitter, randomsaver ]
   e1            : [ inspectGens ]
   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}

physics.producers.evtgen.seed   : 5
physics.producers.detsim.seed   : 6
physics.producers.makeHits.seed : 7
physics.producers.fitter.seed   : 8
```

le—i.e. all

ded—type:

config.fcl

```
#include "fcl/minimalMessageService.fcl"
#include "fcl/standardModules.fcl"

process_name : SampleFHiCLFile

source : {
  module_type : EmptyEvent
  maxEvents   : 10
}

services : { @table::services }

physics  : {
   producers : {
      evtgen      : @local::evtgen_default
      detsim      : @local::detsim_default
      makeHits    : @local::makeHits_default
      fitter      : @local::fitter_default
      randomsaver : @local::randomsaver_default
   }

   analyzers : {
      inspectGens : @local::inspectGens_default
   }

   p1            : [ evtgen, detsim, makeHits, fitter, randomsaver ]
   e1            : [ inspectGens ]
   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}

physics.producers.evtgen.seed    : 5
physics.producers.detsim.seed    : 6
physics.producers.makeHits.seed : 7
physics.producers.fitter.seed    : 8
```

le—i.e. all

ded—type:

config.fcl

Expanded

```
#include "fcl/minimalMessageService.fcl"
#include "fcl/standardModules.fcl"

process_name : SampleFHiCLFile

source : {
  module_type : EmptyEvent
  maxEvents   : 10
}

services : { @table::services }

physics  : {
   producers : {
      evtgen      : @local::evtgen_default
      detsim      : @local::detsim_default
      makeHits    : @local::makeHits_default
      fitter      : @local::fitter_default
      randomsaver : @local::randomsaver_default
   }

   analyzers : {
      inspectGens : @local::inspectGens_default
   }

   p1            : [ evtgen, detsim, makeHits, fitter, randomsaver ]
   e1            : [ inspectGens ]
   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}

physics.producers.evtgen.seed    : 5
physics.producers.detsim.seed    : 6
physics.producers.makeHits.seed : 7
physics.producers.fitter.seed    : 8
```

le—i.e. all

ded—type:

config.fcl

Expanded

SPoM: Insertion facilities

```
#include "prolog_parameters.fcl"

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

}
```

SPoM: Insertion facilities

```
#include "prolog_parameters.fcl"

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities

```
#include "prolog_parameters.fcl"

physics : {

   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }

   analyzers : {
      mod1   : { pdgId : @lo
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

- Assume these producers are owned by the experiment.
- Need to adjust configuration to be robust against future changes.

SPoM: Insertion facilities

```
#include "prolog_parameters.fcl"

physics : {

   producers : {
      prod1  : { pdgId : @local::parame
      prod2  : { pdgId : @local::parame
   }

   analyzers : {
      mod1   : { pdgId : @local::parame
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      prod1 : @local::experiment.producers.prod1
      prod2 : @local::experiment.producers.prod2
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities (**`@table`**)

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      prod1 : @local::experiment.producers.prod1
      prod2 : @local::experiment.producers.prod2
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities (`@table`)

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      prod1 : @local::experiment.producers.prod1
      prod2 : @local::experiment.producers.prod2
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities (**`@table`**)

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p
   e

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      prod1 : @local::experiment.producers.prod1
      prod2 : @local::experiment.producers.prod2
   }

   analyzers : {

                                                  id }

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

`@table` insert the *contents* of the value of the specified parameter.

SPoM: Insertion facilities (**`@table`**)

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1    : { pdgId : @local::parameters.pid }
   }

   p
   e

   t
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      prod1 : @local::experiment.producers.prod1
      prod2 : @local::experiment.producers.prod2
   }

   analyzers : {

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

- If owner of **`experiment.producers`** wants to add/remove a producer, the user will not have to change anything in **`physics.producers`**.

Workflow best practices

1. Strive for a **single point of maintenance** (SPoM) in your FHiCL files.
2. Only **`#include`** files that contain **`PROLOG`**s.
3. Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would include C++ headers.
4. Group modules with a common purpose into one table and use the **`@table`** insertion facility.

Workflow best practices

1. Strive for a **single point of maintenance**

```
producers : {
    sim1          : @local::experiment.producers.sim1
    sim2          : @local::experiment.producers.sim2
    sim3          : @local::experiment.producers.sim3
    tracking1     : @local::experiment.producers.tracking1
    tracking2     : @local::experiment.producers.tracking2
    bookkeeping1: @local::experiment.producers.bookkeeping1
    bookkeeping2: @local::experiment.producers.bookkeeping2
}
```

Not great.

insertion facility.

Workflow best practices

1. Strive for a **single point of maintenance**

```
producers : @local::experiment.producers
```

Not good either.

insertion facility.

Workflow best practices

1. Strive for a **single point of maintenance**

```
producers : {
    @table::experiment.simulation_producers
    @table::experiment.tracking_producers
    @table::experiment.bookkeeping_producers
}
```

Good, and extensible …

insertion facility.

Workflow best practices

1. Strive for a **single point of maintenance**

```
producers : {
    @table::experiment.simulation_producers
    @table::experiment.tracking_producers
    @table::experiment.bookkeeping_producers
    myownProducer : {}
}
```

   insertion facility.

SPoM: Insertion facilities

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities

```
#include "prolog_parameters.fcl"

# Experiment-provided producers
BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities (**`@sequence`**)

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
   prodList  : [ prod1, prod2 ]
}
END_PROLOG
```

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ @sequence::experiment.prodList ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities (**`@sequence`**)

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
   prodList  : [ prod1, prod2 ]
}
END_PROLOG
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ @sequence::experiment.prodList ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

SPoM: Insertion facilities (**`@sequence`**)

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
   prodList  : [ prod1, prod2 ]
}
END_PROLOG
```

```
// experiment_producers.fcl
#include "prolog_parameters.fcl"

BEGIN_PROLOG
experiment : {
   producers : {
      prod1  : { pdgId : @local::parameters.pid }
      prod2  : { pdgId : @local::parameters.pid }
   }
}
END_PROLOG
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {
   mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ @sequence::experiment.prodList ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {
   mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ prod1, prod2 ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

`@sequence` insert the *contents* of the value of the specified parameter.

SPoM: Insertion facilities (**`@sequence`**)

```
#include "prolog_parameters.fcl"
#include "experiment_producers.fcl"

physics : {

   producers : {
      @table::experiment.producers
   }

   analyzers : {
      mod1   : { pdgId : @local::parameters.pid }
   }

   p1 : [ @sequence::experiment.prodList ]
   e1 : [ mod1 ]

   trigger_paths : [ p1 ]
   end_paths     : [ e1 ]

}
```

Miscellany

- There are cases where no suitable default can/should be specified. In these cases, it is best to use:

```
parameter : @nil
```

The parameter value *must be* specified later in the configuration in order for a parameter value to be retrieved in a (e.g.) module.

Miscellany - not discussed

There is other FHiCL syntax out there:

```
parameter : @erase
```

Erases a parameter from the processed FHiCL configuration. Wide-spread use of it is a sign of a design weakness.

Miscellany - not discussed

There is other FHiCL syntax out there:

```
parameter : @erase
```

Erases a parameter from the processed FHiCL configuration. Wide-spread use of it is a sign of a design weakness.

Also available:

```
parameter @protect_ignore: value
parameter @protect_error:  value
```

Means of ensuring that the value of **`parameter`** cannot be overridden later on.

Workflow best practices

1. Strive for a **single point of maintenance** (SPoM) in your FHiCL files.
2. Only **`#include`** files that contain **`PROLOG`**s.
3. Specify directories on your **`FHICL_FILE_PATH`** in a manner similar to how you would include C++ headers.
4. Group modules with a common purpose into one table and use the **`@table`** insertion facility (with **`@sequence`** for path insertions).

See extra slides for more on **`FHICL_FILE_PATH`**

Extra slides

art -c and **FHICL_FILE_PATH**

- If the FHiCL file (e.g. **config.fcl**) is in a directory on the **FHICL_FILE_PATH**, then you can type:

art –c config.fcl

even if **config.fcl** is not in your current directory.

Checking **FHICL_FILE_PATH**

```
printenv FHICL_FILE_PATH | tr : '\n'
```

If the directory that contains the file you need is not in the printed list, you need to add it.

The Linux machines for this course use bash, and **FHICL_FILE_PATH** has already been defined. You can, therefore, use the following to add the directory:

```
FHICL_FILE_PATH=<DIR_TO_ADD>:$FHICL_FILE_PATH
```