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

Session 15: Controlling Output

Rob Kutschke

art and LArSoft Course

August 5, 2015

In this Session You will Learn

- Three properties of a filter module
- The concept of an art path
  - How to stop processing of the remaining module labels in a path
- How to tell an output module:
  - which events it should write
  - which data products it should write
- How to tell an analyzer module:
  - which events it should process.
- Demonstrate that when a module label appears in more than one path, it is executed only once.
- How to drop data products on input.
- A dangerous misunderstanding of the path rules.

Property 1: Inherit from `art::EDFilter`

```
#include "art/Framework/Core/EDFilter.h”

namespace tex {
  class Prod : public art::EDFilter {
  public:
    explicit Prod(fhicl::ParameterSet const& pset);
    bool filter( art::Event& event) override;
  };
}
```

- Analyzer modules inherit from `art::EDAnalyze`
- Producer modules inherit from `art::EDProduce`
- The called-every-event member function is called filter, not analyze or produce.

Property 2: non-const Arguments

```
bool filter( art::Event& event) override;
```

```
void analyze( art::Event const& event) override;
```

- Analyzer modules have only const access to the event.
- Filter and Producer modules have full access to the event
  - They may add new data products
  - They may NOT modify existing data products
- Similarly for arguments of beginRun/endRun etc

Property 3: Filters return bool

```
bool filter( art::Event& event) override;
```

```
void analyze( art::Event const& event) override;
```

- EDFilter
  - `filter` member function returns bool
- Analyzer and Producer modules
  - `produce` and analyze member functions return void
- Similarly for of `beginRun/endRun`

Filters May Produce Their own Data Products

- We advise that filters only produce data products that document the filter decision.
- Suppose you have a task that involves
  - producing some data products
  - making a filter decision based on those data products
- In that case we strongly recommend that you write two modules:
  - one to produce the data products
  - a second to make the filter decision
  - Why? Separation of concerns.
- Your experiment may have standards and practices that address this question. Ask them.

This Exercise has Two Filter Modules

- The code is in the directory art-workbook/FirstFilter
- OddEventNumber_module.cc
  - Returns true if the event number is odd
  - Makes some printout for diagnostic purposes
- MinGens_module.cc
  - Takes a parameter set argument to specify a minimum number of GenParticles
  - Returns true if the number of GenParticles in the event is >= this minimum.
- When you get a chance look, at them and understand them.
- The exercise also uses the FirstHist1 analyzer module that you saw in a previous exericse – it histograms the number of GenParticles in the event.

This Exercise has a Fake Producer Module

- PseudoProducer_module.cc
  - It is an EDProducer
  - So it can fit in any slot that a producer may.
  - It never produces anything; that's OK
  - It produces diagnostic printout
    - This is it's only reason to exist!
- When you get a chance look, at it and understand it.

Recap two Critical Design Rules

- Modules may only communicate with each other via the art::Event
- Analyzer and output modules may NOT modify the art::Event.
- Therefore:
  - If modules obey these rules, it is safe for art to execute analyzer and output modules in any order!
  - If we had the tools to do it, it could even execute them all in parallel! Maybe this will happen some day?

Structure of the physics Parameter Set

```
physics :{
  analyzers :{ }
  filters   :{ }
  producers :{ }

  // 0 or more path definitions

  trigger_paths : []
  end_paths     : []
}
```

- 5 identifiers in red are reserved to *art*
- *art* interprets all other identifiers to be path names
  - A path name must be sequence of module labels
- trigger_paths and end_paths sequences of path names

Ordering rules

- A path used inside trigger_paths, is called a trigger path.
  - It may only contain module labels of filter and producer modules
  - The modules specified in a path will be executed in order
  - No promise about which path is executed first
- A path used inside end_paths, is called an end path
  - It may only contain module labels of analyzer and output modules
  - Modules may executed in any order
- If a module label appears in multiple paths, it is only executed once.

We will Work Through these .fcl Files

- `split1.fcl`
- `split2.fcl`
- `minGens.fcl`
- `andOr.fcl`
- `dropOnOutput.fcl`
- `dropOnInput1.fcl`
- `dropOnInput2.fcl`

`split1.fcl`

- Writes odd numbered events output/oddEvents1.art
- Writes even numbered events to output/evenEvents1.art

split1.fcl

```
physics :{
  filters:{ odd : { module_type : OddEventNumber }}

  oddPath    : [  odd    ]
  evenPath   : [ "!odd" ]
  e1         : [  oddOutput, evenOutput ]

  trigger_paths : [ evenPath, oddPath ]
  end_paths     : [ e1 ]
}
outputs : {
  oddOutput : {
    module_type  : RootOutput
    fileName     : "output/oddEvents1.art"
    SelectEvents : { SelectEvents: [ oddPath ] }
  }
  evenOutput : {
    module_type  : RootOutput
    fileName     : "output/evenEvents1.art"
    SelectEvents : { SelectEvents: [ evenPath ] }
  }
}
```

`split1.fcl`

- There are two trigger paths `oddPath` and `evenPath`
  - One passes only odd numbered events
  - One passes only even numbered events
  - The module is only run once – the result is retained by art and used on the second path.
- An event is written to an output file if the path in the SelectEvents parameter passes all filters in that path
- Remember:
  - Paths are sequences of module labels
  - Trigger_paths and end_paths are sequences of paths

Run split1.fcl

- Run split1.fcl

```
art -c fcl/FirstFilter/split1.fcl
```

- In the output see that the printout from the filter module only appears once
- Inspect to two output files to see that the expected events are there:

```
art -c fcl/FirstModule/first.fcl
-s output/oddEvents1.art

art -c fcl/FirstModule/first.fcl
-s output/evenEvents1.art
```

split2.fcl

```
physics :{
  filters:{ odd : { module_type : OddEventNumber }}

  producers : {
    a : { module_type : PseudoProducer }
    b : { module_type : PseudoProducer }
    c : { module_type : PseudoProducer }
 }


  oddPath    : [  a,   odd,  b  ]
  evenPath   : [  a, "!odd", c  ]
  e1         : [  oddOutput, evenOutput ]

  trigger_paths : [ evenPath, oddPath ]
  end_paths     : [ e1 ]
}
```

- Module b is executed for odd events only and c for even events only!

Run split2.fcl

- Run split1.fcl

```
art -c fcl/FirstFilter/split2.fcl
```

- Observe the printout and verify that module a is run only once per event.
- Verify that modules b, and c are run when expected
- Inspect to two output files to see that the expected events are there:

```
art -c fcl/FirstModule/first.fcl
-s output/oddEvents1.art

art -c fcl/FirstModule/first.fcl
-s output/evenEvents1.art
```

Run minGens.fcl

- This exercise shows that the SelectEvents mechanism can also be used to choose which events are seen by an Analyzer module:
- Run minGens.fcl

```
art -c fcl/FirstFilter/minGens.fcl
```

- This creates a root file output/minGens.root
- Inspect the histograms in this file to see that they behaved as predicted.

andOr.fcl

```
physics :  // Define filters odd and minGens as before
  oddOnly      : [ odd ]
  minGensOnly : [ minGens ]
  AND          : [ minGens, odd  ]   // Logical AND
  trigger_paths : [ oddOnly, minGensOnly, AND ]

  e1             : [ outAND, outOR ]
  end_paths      : [ e1 ]
}
outputs : {
  outAND : {   module_type  : RootOutput
    fileName     : "output/and.art"
    SelectEvents : { SelectEvents: [ AND ] }
  }
  outOR : {   module_type  : RootOutput
    fileName     : "output/or.art"
    SelectEvents: { SelectEvents:
           [ oddOnly,     minGensOnly ] } // Logical OR
  }
}
```

Run andOr.fcl

- This exercise shows how to compose the logical AND and logical OR of two filters:
- Run andOr.fcl

```
art -c fcl/FirstFilter/andOR.fcl
```

- Inspect the event numbers in each of the two output files to verify that they are as expected.

Remaining Files

- dropOnOutput.fcl
  - Shows shows how to use the outputCommands mechanism to choose which data products are written to which file
  - Run this fcl file.
  - Use file dumper to inspect the dat products in the output file
- dropOnInput1.fcl, dropOnInput2.fcl
  - Shows how to use inputCommands to drop data products on input
  - If you drop a data product, art will drop all data products that are derived from it.
  - So if you drop the GenParticles, then everything disappears!

Questions so Far?

Get Started

- Go to your source directory
- Follow the instructions in this pdf file.