Mu2e Minnesota

March 18th

Yan Ke

Setting up for working environment

- Follow the instructions in the link below:
  - http://mu2e.fnal.gov/public/hep/computing/g4fwk.shtml
- After the initial setup, each time log in Fermilab we just need to do:
  - kinit username@FNAL.GOV
  - kinit –l 6d –f
  - ssh –AKX –l username mu2egpvm01.fnal.gov
  - cd /mu2e/users/app/username/Offline
  - setup mu2e
  - source setup.sh
- To have the workbook in your own directory check out 10.4.1
- Art work book is still incomplete. Check Chapter 12 on how to update work book if you want.

Art run-time environment

- What we need to learn is how to write:
  - .fcl file(configuration file)
  - .cc file(module)
- In workbook exercise we use .art file as input data file. For mu2e, we use .root file for input file.
- Cannot setup for workbook exercise now, waiting for Rob's reply.

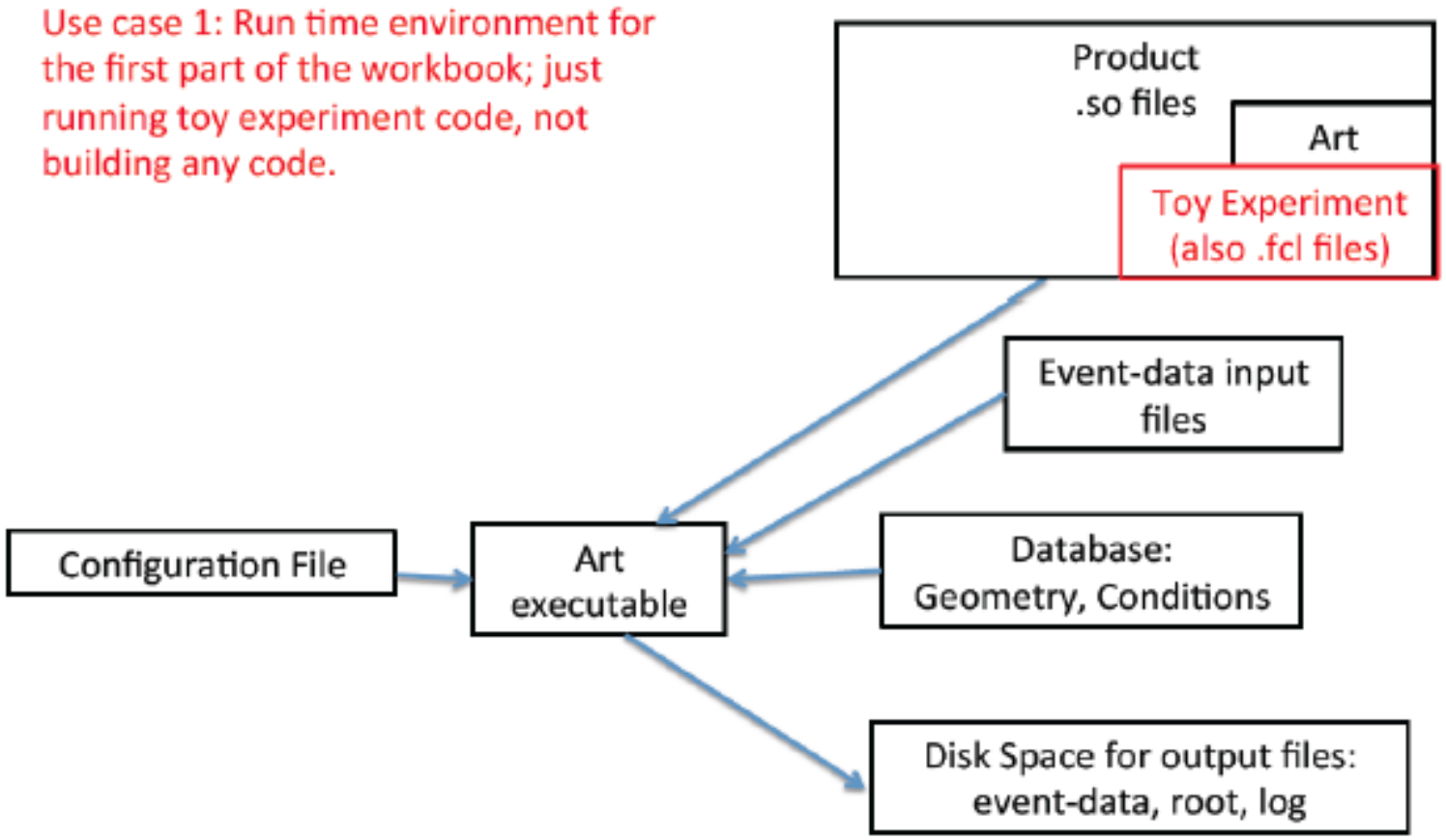


Figure 9.1: Elements of the *art* run-time environment for the first Workbook exercise

The configuration of fcl file

Listing 9.2: Listing of `hello.fcl`

```
1  #include "fcl/minimalMessageService.fcl"
2
3  process_name : hello
4
5  source : {
6      module_type : RootInput
7      fileNames   : [ "inputFiles/input01.art" ]
8    }
9
10 services : {
11     message : @local::default_message
12   }
13
14 physics :{
15     analyzers: {
16       hi : {
17         module_type : HelloWorld
18       }
19   }
20
21   e1         : [ hi ]
22   end_paths : [ e1 ]
23 }
```

- This example consist of three parts
- To run this, type:
  - art -c hello.fcl >& output/hello.log
  - That is:
  - art -c (fcl file) >& (output file, usually log file)
  - In mu2e, we use ‘mu2e’ instead of ‘art’

Output of the fcl file

```
%MSG-i MF_INIT_OK:  art 27-Apr-2013 21:22:13 CDT JobSetup
Messagelogger initialization complete.
%MSG
27-Apr-2013 21:22:14 CDT  Initiating request to open file
inputFiles/input01.art
27-Apr-2013 21:22:14 CDT  Successfully opened file
inputFiles/input01.art
Begin processing the 1st record. run: 1 subRun: 0 event: 1 at
27-Apr-2013 21:22:14 CDT
Hello World! This event has the id: run: 1 subRun: 0 event: 1
Begin processing the 2nd record. run: 1 subRun: 0 event: 2 at
27-Apr-2013 21:22:14 CDT
Hello World! This event has the id: run: 1 subRun: 0 event: 2
Hello World! This event has the id: run: 1 subRun: 0 event: 3
Hello World! This event has the id: run: 1 subRun: 0 event: 4
Hello World! This event has the id: run: 1 subRun: 0 event: 5
Hello World! This event has the id: run: 1 subRun: 0 event: 6
Hello World! This event has the id: run: 1 subRun: 0 event: 7
Hello World! This event has the id: run: 1 subRun: 0 event: 8
Hello World! This event has the id: run: 1 subRun: 0 event: 9
Hello World! This event has the id: run: 1 subRun: 0 event: 10
27-Apr-2013 21:22:14 CDT  Closed file inputFiles/input01.art

TrigReport ---------- Event  Summary ------------
TrigReport Events total = 10 passed = 10 failed = 0

TrigReport ------ Modules in End-Path: e1 ------------
TrigReport  Trig Bit#    Visited     Passed     Failed      Error Name
TrigReport     0    0         10         10          0          0 hi

TimeReport ---------- Time  Summary ---[sec]----
TimeReport CPU = 0.004000 Real = 0.002411

Art has completed and will exit with status 0.
```

Fcl file basic syntax

- Name : value
-

```
5  source : {
6    module_type : RootInput
7    fileNames   : [ "inputFiles/input01.art" ]
8  }
```

- To run on multiple files, add new file name inside the bracket and a “,”
- Physics process syntax: 1. types of modules: analyzer, producer, filters

```
14 physics :{
15   analyzers: {
16     hi : {
17       module_type : HelloWorld
18           }
19     }
20   e1         : [ hi ]
21   end_paths : [ e1 ]
22 }
```

Fcl file basic syntax

- Producer and filter module are in different paths from analyzer module.

```
1    t_anne         : [ p0, p1, p2, f0, p3, f1 ]
2    t_rob          : [ p0, p1, f2, p2, f0, p4 ]
3    trigger_paths  : [ t_anne, t_rob ]
4
5    e_anne         : [ a, b, c, d, e ]
6    e_rob          : [ a, b, f, c, g ]
7    end_paths      : [ e_anne, e_rob ]
```

HelloWorld module (Analyzer module)

- 1 #include "art/Framework/Core/EDAnalyzer.h"
- 2 #include "art/Framework/Core/ModuleMacros.h"
- 3 #include "art/Framework/Principal/Event.h"
- 5 #include <iostream>
- 7 namespace tex {
- 9 class First : public art::EDAnalyzer {
- 11 public:
- 13 explicit First(fhicl::ParameterSet const&);
- 15 void analyze(art::Event const& event) override;
- 17 };
- 19 }
- 21 tex::First::First(fhicl::ParameterSet const& pset) :
- 22 art::EDAnalyzer(pset) {
- 23 std::cout << "Hello from First::constructor."
- 24 << std::endl;
- 25 }
- 27 void tex::First::analyze(art::Event const& event){
- 28 std::cout << "Hello from First::analyze. Event id: "
- 29 << event.id()
- 30 << std::endl;
- 31 }
- 33 DEFINE_ART_MODULE(tex::First)

- Output from first.fcl, module which use this module.
  - Before using this module, we need to build again. In art work book, they use buildtool. In mu2e, we use scons. e.g. scons –j 4 (run 4 threads on building)
  - Art will find the new module by itself. The output is similar to the output from the previous one.

- A few comments:
  - Might need to check out section 10.7 on how the build system works. Sometimes we need to add something in Cmakelist.txt
  - event.ran(), event.subrun(), event.event() provides ranID subrunID and eventID. There is another way, it is a part of the exercise.
  - Naming convention: has to be in the form: xxx_module.cc
  - artmod provides us skeleton of source file.
    - e.g. artmod analyzer tex::Third （create a analyzer source file contains a class named Third）
    - Check out more on artmod by ‘artmod –help’

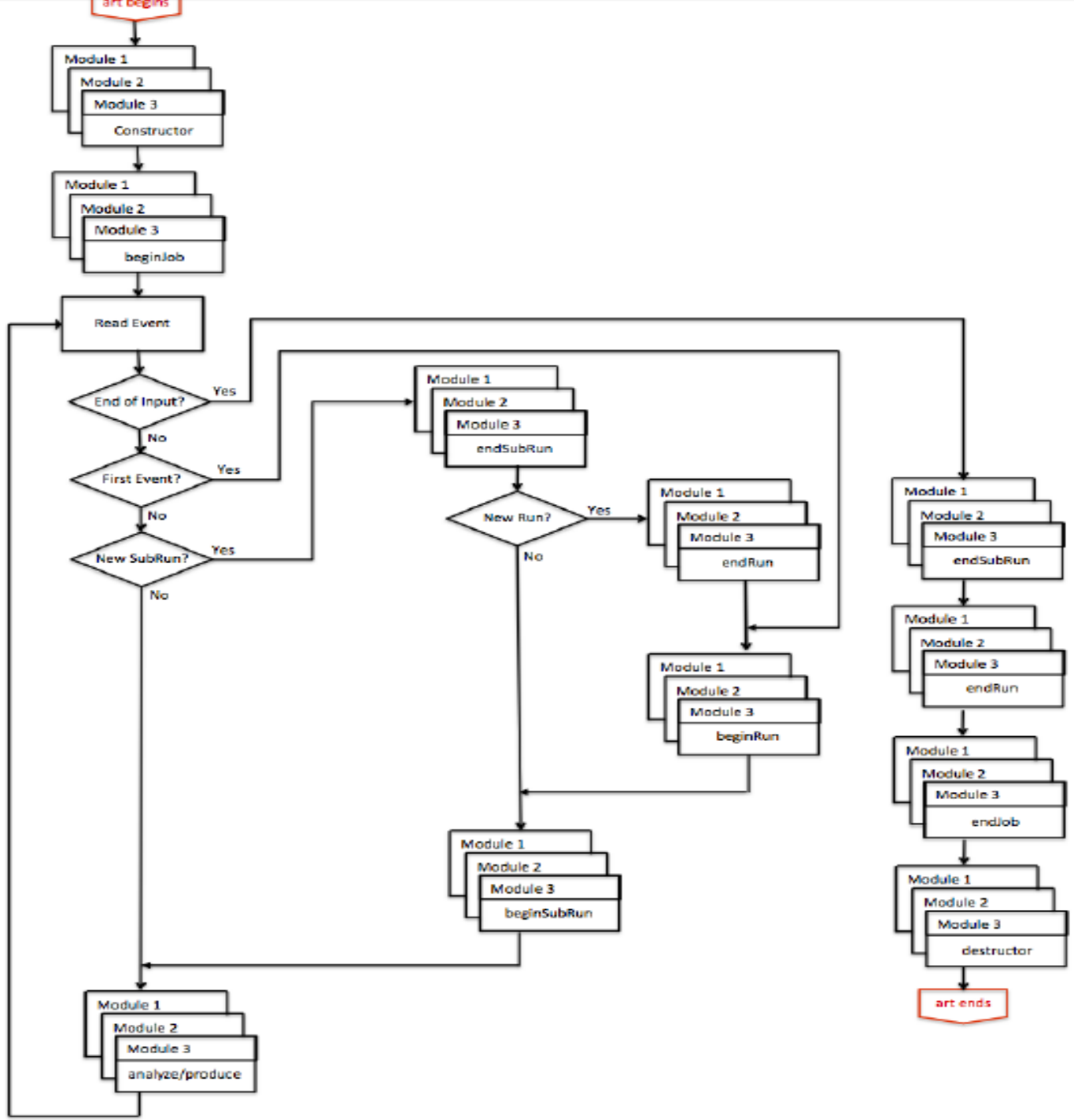
Running many modules in art

Listing 10.5: The `physics` parameter set for `all.fcl`

```
physics :{
  analyzers: {
    hello : {
      module_type : HelloWorld
    }
    first : {
      module_type : First
    }
    second : {
      module_type : Second
    }
    third : {
      module_type : Third
    }
 }

  e1         : [ hello, first, second, third ]
  end_paths : [ e1 ]

}
```



Art event loop (partial)

More clear feature in Figure 3.2

Implementation part of Optional_module.cc

- #include "art/Framework/Core/ModuleMacros.h"
- #include "art/Framework/Principal/Event.h"
- #include "art/Framework/Principal/Run.h"
- #include "art/Framework/Principal/SubRun.h"

- #include <iostream>

- namespace tex {

- class Optional : public art::EDAnalyzer {

- public:

- explicit Optional(fhicl::ParameterSet const&);
- void beginJob () override;
- void beginRun (art::Run const& run) override;
- void beginSubRun(art::SubRun const& subRun) override;
- void analyze (art::Event const& event) override;

- };

About the parameter set

```
1  moduleLabel : {
2     module_type       : ClassName
3     thisParameter     : 1
4     thatParameter     : 3.14159
5     anotherParameter  : "a string"
6     arrayParameter    : [ 1, 3, 5, 7, 11] }
7     nestedPSet        : {
8                           a : 1
9                           b : 2
10                        }
11 }
```

- Module_type parameter is required by art
- The following are user(or say module) defined parameter