Mu2e Minnesota

March 22th

Yan Ke

Correction and something more

- scons –j 4 is preferred. Since the machine has only 32 cores, if one use to much threads, other people will not be able to work.
- cd /mu2e/app/users/username/Offline
- How to find modules header file in art
  - $ART_INC/art/Framework/Principal/Event.h If inside art directory
  - $ROOT_INC/TH1D.h If it is an root header file

About the parameter set

Listing 14.1: Parameter set `psetTester` from `pset01.fcl`

```
1   analyzers: {
2     psetTester : {
3       module_type : PSet01
4       a : "this␣is␣quoted␣string"
5       b : 42
6       c : 3.14159
7       d : true
8       e : [ 1, 2, 3 ]
9       f : {
10        a : 4
11        b : 5
12      }
13    }
14  }
```

- Module_type parameter is required by art
- The following are user(or say module) defined parameter
- Values have no meaning to FHiCL
- Nested parameter can be arbitrary depth

PSet01_module.cc

```
1  tex::PSet01::PSet01(fhicl::ParameterSet const& pset ):
2    art::EDAnalyzer(pset){

4    std::string           a=pset.get<std::string>("a");
5    int                   b=pset.get<int>   ("b");
6    double                c=pset.get<double>("c");
7    bool                  d=pset.get<bool>  ("d");
8    std::vector<int>      e=pset.get<std::vector<int>>("e");
9    fhicl::ParameterSet   f=pset.get<fhicl::ParameterSet>("f");

11   int                   fa=f.get<int>("a");
12   int                   fb=f.get<int>("b");

14   std::string module_type  =
15                 pset.get<std::string>("module_type");

17   std::string module_label =
18                 pset.get<std::string>("module_label");
```

Listing 14.2: First part of constructor in `PSet01_module.cc`

- Two important things:
- 1. the name of the parameter
- 2. the type to which the string representation should be converted

A few comment

- 5 header files included:
  - EDAnalyzer.h, ModuleMacros.h, Event.h, <iostream>, <string>
- The object pset internally represents the value of each parameter as a string.
- int b = pset.get<int>("b");
- std::string a = pset.get<std::string>("a");
  - 1. Check if it has a parameter named a.
  - 2. If it has this parameter, return it as a string.
  - Throw exception if no such value or incorrect type
- When error occurs, unless art cannot find requested data product, art will do orderly shut down. (Do all the end member function and some other things in 14-252)

A few comments

- Default values
  - int b = pset.get<int>("b",0); //default 0
  - std::vector<double>(5,1.0); //vetor of length 5, all elements set to 1.0

Background on Accessing Data Product

- Each generated particle in the simulated event is described by an object of type tex::GenParticle
- typedef std::vector<GenParticle> GenParticleCollection;
- Data product name:
  - MyDataType_MyModuleLabel_MyInstanceName_MyProcessName
  - ModuleLabel is defined in FHiCL file
  - InstanceName is used for distinguish different data products made by the same module instance. The only one can be empty string among 4 field.
  - ProcessName field hold the value of process_name parameter in the FHiCL file which created this data product.
  - MyDataType: not required to write, need to know how to recognize.
    - If it is not collection type, then it's the class name
    - std::vector<T> Ts
    - std::vector<std::vector<T> > Tss
    - cet::map_vector<T> Tmv

Background on Accessing Data Product

- If the event contains more than one data product that matches this specification, then art requires that you also specify the process name. Otherwise, can use wild card match for process name field.
- art::InputTag is used for specify name field other than data type.
- art::InputTag tag("MyModuleLabel:MyInstanceName:MyProcessName");
  - gens.isValid()
  - art::InputTag tag("evtgen::exampleInput"); // Instance name is empty string
  - art::InputTag tag("evtgen"); // process name is rarely needed
- Check this out in section 16.4 if not clear enough (I'm not clear enough)

readGens1.fcl

Listing 16.4: Configuring the module label `read` in `readGens1.fcl`

```
1    read : {
2      module_type           : ReadGens1
3      genParticlesInputTag : "evtgen"
4    }
```

ReadGens1_module.cc

```
9  namespace tex {
10   class ReadGens1 : public art::EDAnalyzer {

12     public:
13     explicit ReadGens1(fhicl::ParameterSet const& );
14     void analyze(art::Event const& event) override;

16     private:
17     art::InputTag gensTag_;
18   };
19 }
20 tex::ReadGens1::ReadGens1(fhicl::ParameterSet const& pset):
21   art::EDAnalyzer(pset),
22   gensTag_(pset.get<std::string>("genParticlesInputTag")){
23 }
24 void tex::ReadGens1::analyze(art::Event const& event ){
```

- One more header file included:
  - #include "toyExperiment/MCDataProducts/GenParticleCollection.h“

ReadGens1_module.cc

```
26   art::Handle<GenParticleCollection> gens;
27   event.getByLabel(gensTag_,gens);

29   std::cout << "ReadGens1::analyze event: "
30             << event.id().event()
31             << " GenParticles: "
32             << gens->size()
33             << std::endl;
34 }

36 DEFINE_ART_MODULE(tex::ReadGens1)
```

- gens.isValid() used for check whether gens is valid or not.

- Handle is a safety pointer, has interface to access the data.
- The event look for data product in line 27. (Do 2 things)
  - There has to be exactly one product matches the 4 name field. Otherwise(no matches, several matches), will leave handle default.
  - Will add data to the handle.
- For exception, unless cannot find product, there would be graceful shut down.

A few comments on ReadGens1_module.cc

```
art::ValidHandle<GenParticleCollection> gens =
    event.getValidHandle<GenParticleCollection>(gensTag_);
```

- We can use ValidHandle<T> instead of Handle<T>
- It's more efficient since we don't need to check the validation of the handle each time

```
auto gens =
    event.getValidHandle<GenParticleCollection>(gensTag_);
```

- Another concise version
- Auto is handy but might cause confusion

Reference

- Art workbook v0.9