# Session 2:

# Write and run your own module, step by step

updated on
16.06.2017: code examples are now in develop
09.11.2017: minor code updates (changes to geometry services)



R. Sulej, LArSoft Tutorial at CERN, 31.10.2016

## Start this excersises when you have:

- LArSoft environment configured
- dunetpc repository checked out and compiled
- file with reconstructed events in your hands

## What is illustrated:

- create local branch in dunetpc for your code, basic git commands
- prepare everything needed for a new analyzer module

- read/use FHiCL configuration parameters
- access data products
- access association between data products
- find MC truth corresponding to the reconstructed object
- use FHiCL validation

We will go to dunetpc and create new (local) branch there

At each point you can swich between branches:

- develop
- your branch (feature/user_Branchname)
- this tutorial sources (feature/cern_Tutorial): *tutorial branch has been merged into dunetpc develop now, slides are modified accordingly*

→ nothing is lost if it was commited

Follow instructions on slides exactly…

…or do something similar to what is shown

e.g.: count tracks instead of clusters, calculate completness instead of cleanliness

## Create a branch in dunetpc repository:

[robert@localhost Protodune]$ pwd
/home/robert/fnal/v6/**srcs/dunetpc**/dune/Protodune ← use git commands <u>anywhere inside</u> repository

[robert@localhost Protodune]$ **git flow feature start *user_Branchname*** ← use ***username_Branchname*** format
Switched to a new branch 'feature/*user_Branchname*'

Summary of actions:
- A new branch 'feature/*user_Branchname*' was created, based on 'develop'
- You are now on branch 'feature/*user_Branchname*'

Now, start committing on your feature. When done, use: ← we'll do this on next slides

git flow feature finish *user_Branchname* ← this will merge your branch with develop and then will delete your branch

[robert@localhost Protodune]$ **git status** ← use this often
On branch feature/*user_Branchname*
Untracked files:
(use "git add <file>..." to include in what will be committed)

../TrackFinderDUNE/trackfindermodules_dune.fcl~ ← this may be empty, depending on your files and status of work
../Utilities/SignalShapingServiceDUNE10kt_service.cc~
../Utilities/SignalShapingServiceDUNEDPhase_service.cc~
../../fcl/dune35t/reco/standard_reco_dune35t.fcl~
../../fcl/dunefd/reco/standard_reco_dune10kt.fcl~

nothing added to commit but untracked files present (use "git add" to track)

[robert@localhost Protodune]$

## Useful git commands:

[robert@localhost Protodune]$ **git branch** ← list local branches
 develop
* feature/user_Branchname
 master

[robert@localhost Protodune]$ **git branch –a** ← list also remote branches (checkout remote one to have it on your local list)
 develop
* feature/user_Branchname
 master
 remotes/origin/HEAD -> origin/develop
 remotes/origin/develop
 remotes/origin/feature/35tGeometry
 remotes/origin/feature/Issue1083
 remotes/origin/feature/OpticalRecoUpdate
 remotes/origin/feature/RestoreDUNE35tParticleStitcher
...

[robert@localhost Protodune]$ **git checkout develop** ← will switch you to develop branch (listed here for reference no need to do it now unless you'd like to „git pull")
Switched to branch 'develop,
Your branch is up-to-date with 'origin/develop'.

[robert@localhost Protodune]$ **git checkout feature/*user_Branchname*** ← switch back to your branch
Switched to branch 'feature/*user_Branchname*,

[robert@localhost Protodune]$ **git merge develop** ← merge develop head into your current branch (e.g. after pulling new release)
Already up-to-date.

## …and also useful:

| | |
|---|---|
| **git pull** | ← pull HEAD version from the repository (from the current branch) |
| **git add code.cxx** | ← add new file |
| **git commit code.cxx –m „message about changes”** | ← locally commit changes in the file |
| **git commit –a –m „changes in many files”** | ← locally commit all changes in the repository |
| **git flow feature publish *user_Branchname*** | ← publish your branch: you and others will see the, as well as all previous local commits to the branch |
| **git push** | ← push changes to the remote repository (current, public branch) |
| **git stash** | ← record the current state at index, revert to the HEAD state |
| **git checkout -- code.cxx** | ← drop the not-commited changes |
| **git diff code.cxx** | ← difference w.r.t. the HEAD |

Finally:

| | |
|---|---|
| **git flow feature finish *user_Branchname*** | ← merge your branch with develop, delete local and remote branch |

or:

| | |
|---|---|
| **git branch –d** | ← delete local branch |
| **git branch –D** | ← delete remote branch |

Commands pushing to remote repository (e.g. also publish) require write access. To do this you need FNAL computing account and ask to add you to repository developers. Today we are working in read-only mode for remote repository, all commits are saving your work locally (in neut cluster or on your laptop).

## **Prepare place for the new module:** (if you did not merge develop into your branch, so „starting from scratch”)

[robert@localhost Protodune]$ pwd
/home/robert/fnal/v6/srcs/dunetpc/dune/Protodune

[robert@localhost Protodune]$ mkdir MyWork ← create a directory for your code (if it is not already there)
[robert@localhost Protodune]$ ls
**CMakeLists.txt** singlephase **MyWork**

[robert@localhost Protodune]$ gedit **CMakeLists.txt** &

```
add_subdirectory(singlephase)
add_subdirectory(MyWork)
```

← add (or note if present) this line, using your directory name

[robert@localhost Protodune]$ cd MyWork
[robert@localhost MyWork]$ **artmod analyzer MyClusterCounter** ← create template for module class only, we'll need to add CMakeList.txt for it, and default config .fcl for convenience

INFO: Wrote /home/robert/fnal/v6/srcs/dunetpc/dune/Protodune/MyWork/MyClusterCounter_module.cc
[robert@localhost MyWork]$ ls
**MyClusterCounter_module.cc**

[robert@localhost MyWork]$ gedit CMakeLists.txt &

You need to write from scratch CMakeLists.txt for the module:
(similar files are in each diirectory, please, compare)

## CMakeList.txt

```
simple_plugin(MyClusterCounter "module"
              lardataobj_RecoBase
              larcorealg_Geometry
              larcore_Geometry_Geometry_service


              larsim_MCCheater_BackTracker_service


              nusimdata_SimulationBase
              ${ART_FRAMEWORK_CORE}
              ${ART_FRAMEWORK_PRINCIPAL}
              ${ART_FRAMEWORK_SERVICES_REGISTRY}
              ${ART_FRAMEWORK_SERVICES_OPTIONAL}
              ${ART_FRAMEWORK_SERVICES_OPTIONAL_TFILESERVICE_SERVICE}
              art_Persistency_Common canvas
              art_Persistency_Provenance canvas
              art_Utilities canvas
              ${MF_MESSAGELOGGER}
              ${MF_UTILITIES}
              cetlib cetlib_except
              ${ROOT_BASIC_LIB_LIST}
              BASENAME_ONLY
)

install_headers()
install_fhicl()
install_source()
install_scripts()
```

← modify name of the module

← minimal list for our example module, you'll need to add libraries depending on your code

← today needed only for MC truth matching

# MyClusterCounter_module.cc

```
#include "art/Framework/Core/EDAnalyzer.h"
#include "art/Framework/Core/ModuleMacros.h"
#include "art/Framework/Principal/Event.h"
#include "art/Framework/Principal/Handle.h"
#include "art/Framework/Principal/Run.h"
#include "art/Framework/Principal/SubRun.h"
#include "canvas/Utilities/InputTag.h”
#include "fhiclcpp/ParameterSet.h"
#include "messagefacility/MessageLogger/MessageLogger.h"

namespace tutorial {

class MyClusterCounter : public art::EDAnalyzer {
public:
  explicit MyClusterCounter(fhicl::ParameterSet const & p);

  // Plugins should not be copied or assigned.
  MyClusterCounter(MyClusterCounter const &) = delete;
  MyClusterCounter(MyClusterCounter &&) = delete;
  MyClusterCounter & operator = (MyClusterCounter const &) = delete;
  MyClusterCounter & operator = (MyClusterCounter &&) = delete;

  // Required functions.
  void analyze(art::Event const & e) override;

private:
};

MyClusterCounter::MyClusterCounter(fhicl::ParameterSet const & p) : EDAnalyzer(p) {}

void MyClusterCounter::analyze(art::Event const & e)
{
   std::cout << "My module on event #" << e.id().event() << std::endl;
}

} // tutorial namespace

DEFINE_ART_MODULE(tutorial::MyClusterCounter)
```

← **make sure there is no bug in the auto-generated code** (canvas/Utilities)

← put your code in a namespace, please (namespace tutorial {)

← just display event number to see it alive in the modules chain

← add the end of your namespace

← and remmber to add namespace also here

## myclustercounter.fcl – default / reference configuration for your module

[robert@localhost MyWork]$ gedit myclustercounter.fcl & ← create the default configuration file

```
# #include "trackfinderalgorithms.fcl"

BEGIN_PROLOG

my_cluster_counter:
{
  module_type:          "MyClusterCounter"

  ClusterModuleLabel:  "linecluster"
  MinSize:              10
}

END_PROLOG
```

← hash is also the comment tag in the FHiCL language

← module label of your choice

← <u>must match the module class name</u>

← a parameter, e.g. name of clusters producer

← a parameter, e.g. min. number of hits in cluster

← .fcl encapsulated in BEGIN/END_PROLOG may be included in another .fcl

You will include this file later on in the job configuration file. Remember the convention:

- default / reference configuration .fcl close to the module or algorithm code
- job configuration includes this file and can owerwrite parameters

Pay attention also to:

- fcl file names has to be unique across entire code
- module class name has to be unique as well, note that namespace is ignored

# Build it!

[robert@localhost Protodune]$ pwd
/home/robert/fnal/v6/srcs/dunetpc/dune/Protodune

[robert@localhost Protodune]$ **git status**
On branch feature/user_Branchname
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   CMakeLists.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        CMakeLists.txt~
        MyWork/

no changes added to commit (use "git add" and/or "git commit -a")

[robert@localhost Protodune]$ **cd $MRB_BUILDDIR**
[robert@localhost build_slf6.x86_64]$ **mrbsetenv**
The working build directory is /home/robert/fnal/v6/build_slf6.x86_64
The source code directory is /home/robert/fnal/v6/srcs
----------- check this block for errors -----------------------
----------------------------------------------------------------
[robert@localhost build_slf6.x86_64]$ **mrb i -j2**
/home/robert/fnal/v6/build_slf6.x86_64
calling buildtool -I /home/robert/fnal/v6/localProducts_larsoft_v06_11_00_e10_prof -i -j2
INFO: Install prefix = /home/robert/fnal/v6/localProducts_larsoft_v06_11_00_e10_prof
INFO: CETPKG_TYPE = Prof.
… … …
------------------------------------
**INFO: Stage install / package successful.**
------------------------------------

# …and commit

```
[robert@localhost build_slf6.x86_64]$ cd $MRB_SOURCE/dunetpc/dune/Protodune

[robert@localhost Protodune]$ git add MyWork/
[robert@localhost Protodune]$ git status

On branch feature/user_Branchname
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

            modified:   CMakeLists.txt
            new file:   MyWork/CMakeLists.txt
            new file:   MyWork/MyClusterCounter_module.cc
            new file:   MyWork/myclustercounter.fcl

[robert@localhost Protodune]$ git commit -a -m "add tutorial module"
[feature/user_Branchname 2f47352] add tutorial module
 4 files changed, 89 insertions(+)
 create mode 100644 dune/Protodune/MyWork/CMakeLists.txt
 create mode 100644 dune/Protodune/MyWork/MyClusterCounter_module.cc
 create mode 100644 dune/Protodune/MyWork/myclustercounter.fcl

[robert@localhost Protodune]$ git status
On branch feature/user_Branchname
nothing to commit, working directory clean
```

# Run the module 1: configure job

```
[robert@localhost v6]$ pwd
/home/robert/fnal/v6
```
← **create a place for your job files, outside of the development area**

```
[robert@localhost v6]$ mkdir job
[robert@localhost v6]$ gedit job/my_job.fcl &
```
← write new file or use the one which is added to the TutorialExample directory

```
#include "services_dune.fcl”
#include "myclustercounter.fcl”

process_name: TutorialAna

services:
{
  TFileService: { fileName: "tutorial_hist.root" }
  TimeTracker:      {}
  MemoryTracker:     {}
  RandomNumberGenerator: {}
  message:            @local::dune_message_services_prod_debug
  FileCatalogMetadata:  @local::art_file_catalog_mc
                    @table::protodune_services
                    @table::protodune_simulation_services
}

source:
{
  module_type: RootInput
  maxEvents: -1
  fileNames: ["input_file.root"]
}

physics:
{
 analyzers:
 {
  clusterana:        @local::my_cluster_counter
 }

 ana:  [ clusterana ]

 end_paths:    [ ana ]
}
```

- `#include "services_dune.fcl”` ← many experiment and detector services: geometry, simulation, electronics, … …
- `#include "myclustercounter.fcl”` ← **your module default configuration**
- `TFileService` ← load the service that manages root files for histograms
- `RandomNumberGenerator` ← ART native random number generator
- `source:` ← set source as a root file
- `maxEvents: -1` ← -1 for all events in the input file or any positive number you whish
- `clusterana:` ← your label for the my_cluster_counter module defined in myclustercounter.fcl
- `ana:` ← „path”, vector of analyzer modules (can be many, comma separated)
- `end_paths:` ← vector of paths to run analyzers, after all filters and producers finished in their paths in trigger_pats (not used in this job)

**art / LArSoft running on input file:**
(a scheme simplified for our purposes
please, see art reference book)

events flow
output debugging variables
output analysis variables

ROOT file with **art::Events**

source: {module_type: RootInput}

reco1
bool filter(art::Event& evt)
void produce(art::Event& evt)
void produce(art::Event& evt)

reco2
void produce(art::Event& evt)
void produce(art::Event& evt)

**trigger_paths:** [reco1, reco2]

ROOT file with **TTree**'s of your **variables for analysis**

ana
void analyze(art::Event **const** & evt)
void analyze(art::Event **const** & evt)
void analyze(art::Event **const** & evt)

stream1
out1: {module_type: RootOutput}

**end_paths:** [ ana, stream1 ]

**output file:** ROOT file with **art::Events**

# Run the module 2: run LArSoft

```
[robert@localhost v6]$ pwd
/home/robert/fnal/v6
```
← run LArSoft from „one level up" of the srcs folder

[robert@localhost v6]$ **lar -c job/my_job.fcl  your_path_to_data/example_reco.root**

```
27-Oct-2016 04:10:41 CDT  Initiating request to open input file "your_path_to_data/example_reco.root"
27-Oct-2016 04:10:41 CDT  Initiating request to open input file "your_path_to_data/example_reco.root"
27-Oct-2016 04:10:41 CDT  Opened input file "your_path_to_data/example_reco.root"
27-Oct-2016 04:10:41 CDT  Opened input file "your_path_to_data/example_reco.root"
MyClusterCounter module on event #1
MyClusterCounter module on event #2
MyClusterCounter module on event #3
MyClusterCounter module on event #4
MyClusterCounter module on event #5
27-Oct-2016 04:10:42 CDT  Closed input file "your_path_to_data/example_reco.root"
27-Oct-2016 04:10:42 CDT  Closed input file "your_path_to_data/example_reco.root"

TrigReport ---------- Event  Summary ------------
TrigReport ---------- Event  Summary ------------
TrigReport Events total = 5 passed = 5 failed = 0
TrigReport Events total = 5 passed = 5 failed = 0

TrigReport ------ Modules in End-Path: end_path ------------
TrigReport ------ Modules in End-Path: end_path ------------
TrigReport  Trig Bit#   Visited   Passed   Failed    Error Name
TrigReport  Trig Bit#   Visited   Passed   Failed    Error Name
TrigReport    0    0       5        5        0         0 clusterana
TrigReport    0    0       5        5        0         0 clusterana

TimeReport ---------- Time  Summary ---[sec]----
… … …
… … …
… … …
==============================================================
```

← **output from your module!**

← OK, your analyzer was running in the end-path

← more output from all reports

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

## Read FHiCL parameter, access a data product, fill histogram

(or use ClusterCounter2_module.cc)

```
#include "art/Framework/Services/Optional/TFileService.h"
#include "lardataobj/RecoBase/Cluster.h"
#include "TTree.h"

class MyClusterCounter : public art::EDAnalyzer {
public:
  explicit MyClusterCounter(fhicl::ParameterSet const & p);

  // Plugins should not be copied or assigned.
  MyClusterCounter(MyClusterCounter const &) = delete;
  MyClusterCounter(MyClusterCounter &&) = delete;
  MyClusterCounter & operator = (MyClusterCounter const &) = delete;
  MyClusterCounter & operator = (MyClusterCounter &&) = delete;

  // Required functions.
  void analyze(art::Event const & e) override;

  // Selected optional functions.
  void beginJob() override;
  void endJob() override;

private:
  size_t fEvNumber;

  TTree *fEventTree;
  size_t fNClusters;

  TTree *fClusterTree;
  size_t fNHits;

  // ******* fcl parameters *******
  art::InputTag fClusterModuleLabel;
  size_t fMinSize;
};
```

← include these headers for the new types used here

← called before the events are processed

← ..and this one after all events are done

← will keep event number to be stored in trees

← TTree for variables made for each event

← ..like number of clusters reconstructed in each event

← TTree for variables made for each cluster

← like number of hits in each cluster

← **tag*** of cluster producer module

← a parameter: minimum size of cluster

* **art::InputTag tag(„moduleLabel:instanceName:processName”);**

# Read FHiCL parameter, access a data product, fill histogram

(or use ClusterCounter2_module.cc)

```
MyClusterCounter::MyClusterCounter(fhicl::ParameterSet const & p) : EDAnalyzer(p)
{
  fClusterModuleLabel = p.get< std::string >("ClusterModuleLabel");   ← simple way to read parameter value
  fMinSize = p.get< size_t >("MinSize");                                 (easy to mess up e.g. when overwriting value)
}
void MyClusterCounter::beginJob()
{
  art::ServiceHandle<art::TFileService> tfs;                            ← TTree's are managed by ROOT (don't delete them)

  fEventTree = tfs->make<TTree>("EventTree", "event by event info");
  fEventTree->Branch("event", &fEvNumber, "fEvNumber/I");
  fEventTree->Branch("nclusters", &fNClusters, "fNClusters/I");

  fClusterTree = tfs->make<TTree>("ClusterTree", "cluster by cluster info");
  fClusterTree->Branch("event", &fEvNumber, "fEvNumber/I");
  fClusterTree->Branch("nhits", &fNHits, "fNHits/I");
}
void MyClusterCounter::analyze(art::Event const & evt)
{
  fEvNumber = evt.id().event();
  mf::LogVerbatim("MyClusterCounter") << "MyClusterCounter module on event #" << fEvNumber;   ← message facility, see analyze_job.fcl for settings

  auto const & clusters = *evt.getValidHandle< std::vector<recob::Cluster> >(fClusterModuleLabel);   ← art::ValidHandle< std::vector<recob::Cluster> >

  fNClusters = 0;

  for (auto const & clu : clusters)                                      ← loop over recob::Cluster's stored in the vector
  {
    fNHits = clu.NHits();
    fClusterTree->Fill();

    if (fNHits >= fMinSize) { ++fNClusters; }
  }
  fEventTree->Fill();
}
void MyClusterCounter::endJob()
{
  mf::LogVerbatim("MyClusterCounter") << "MyClusterCounter finished job";   ← just to see when it is called
}                                                                             (e.g. you can close files opened in beginJob())
```

# Compile changes, setup job file… and produce histogram

```
[robert@localhost MyWork]$ cd $MRB_BUILDDIR
[robert@localhost build_slf6.x86_64]$ make install -j2
[robert@localhost build_slf6.x86_64]$ cd ..
[robert@localhost v6]$ gedit job/my_job.fcl &
```

← only code changed: „make" is enough and faster than „mrb i"

← note / change what is now being useful in the job config

```
#include "services_dune.fcl"
#include " myclustercounter.fcl"
process_name: TutorialAna
services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "tutorial_hist.root" }
  TimeTracker:       {}
  MemoryTracker:     {}
  RandomNumberGenerator: {} #ART native random number generator
  message:           @local::dune_message_services_prod_debug
  FileCatalogMetadata:  @local::art_file_catalog_mc
     @table::protodune_services  @table::protodune_simulation_services
}
services.message.destinations.LogStandardOut.threshold: "INFO"
source:
{
  module_type: RootInput
  maxEvents: -1
  fileNames: ["input_file.root"]
}
physics:
{
 analyzers: { clusterana: @local::my_cluster_counter }
 ana:  [ clusterana ]
 end_paths:     [ ana ]
}
physics.analyzers.clusterana.ClusterModuleLabel: "linecluster"
physics.analyzers.clusterana.MinSize:                  15
```

← **this ROOT file will contain TTree's with your variables**

← **setup log level INFO, WARN (default), or ERROR**

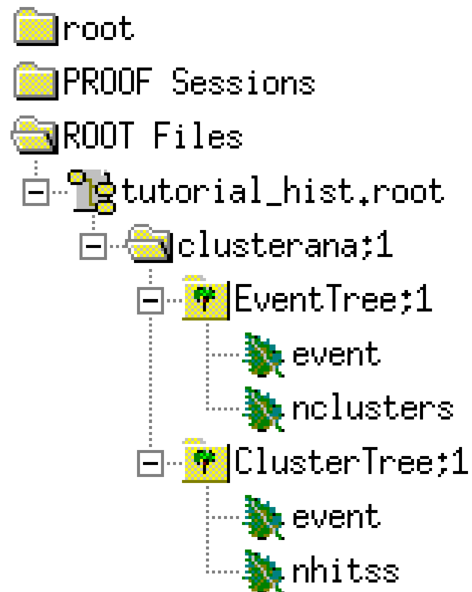
← no changes in the paths, but default parameters of module can be changed:

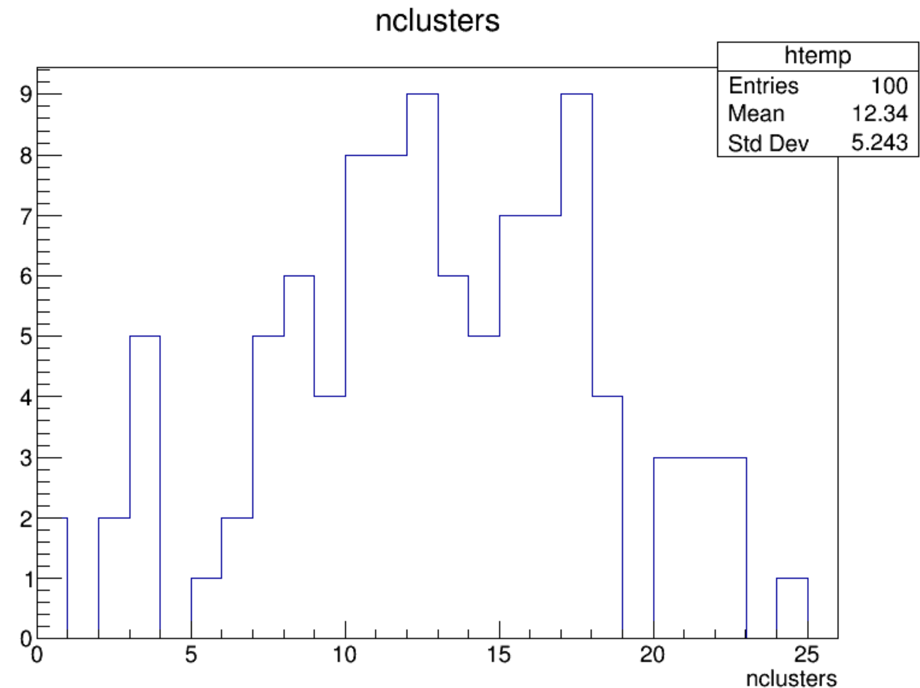
← select clusters producer label (actually this one was default)

← ..and setup any other prameters of the module

## …and produce histogram

[robert@localhost v6]$ **lar -c job/my_job.fcl your_path_to_data/example_reco.root**
[robert@localhost v6]$ root tutorial_hist.root
…
root [0] TBrowser t;



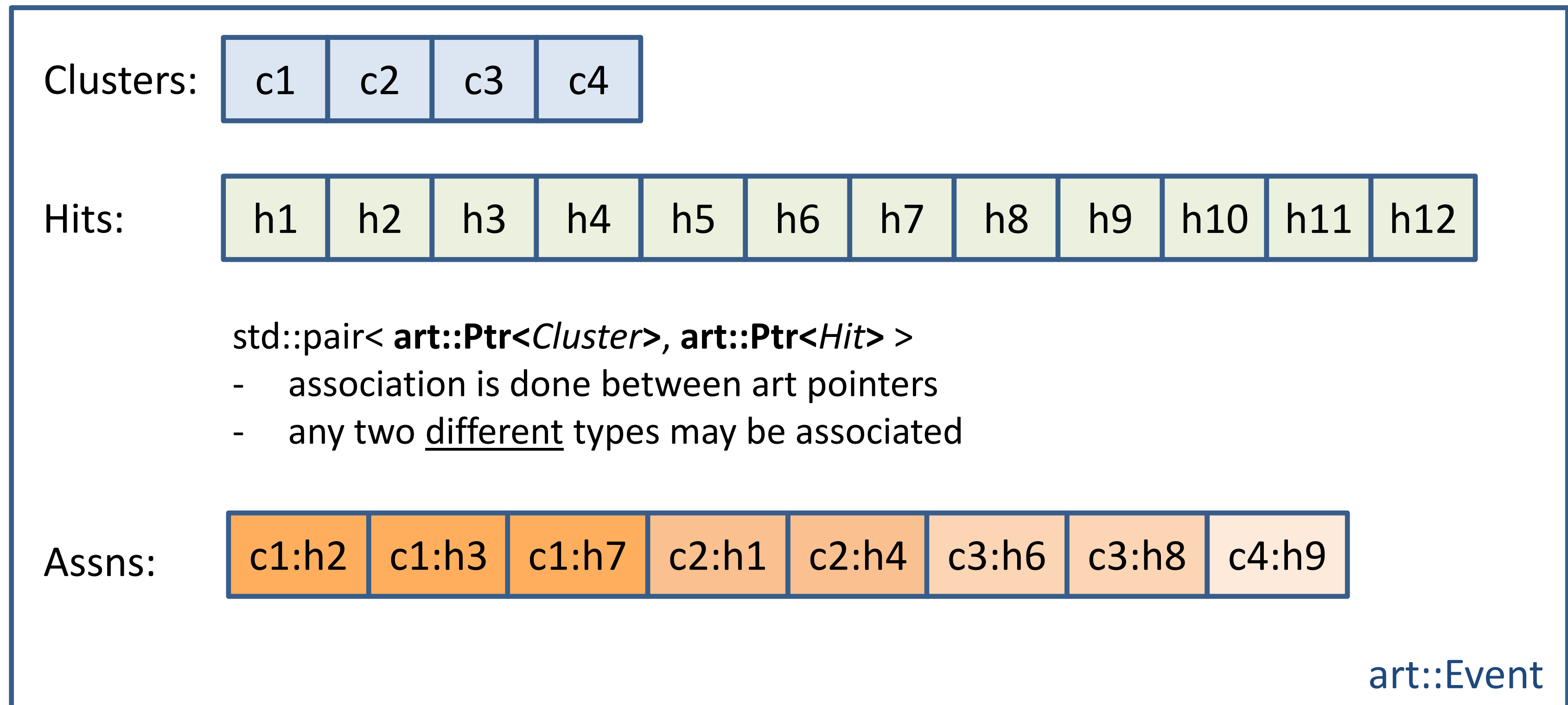
# Cluster object is NOT a container of hits

recob::Cluster describes properties of a cluster:

- which TPC, which view, how many hits, sum of charges, …
- see: http://nusoft.fnal.gov/larsoft/doxsvn/html/classrecob_1_1Cluster.html

Data products do not contain other data products nor pointers.

**Data products can be associated to other data products**: one-to-one, one-to-many.
**Association is just another data product** made by producer module.



## Access hits associated to cluster

(or use ClusterCounter3_module.cc)

```
#include "canvas/Persistency/Common/FindManyP.h"
#include "lardataobj/RecoBase/Hit.h"
class MyClusterCounter : public art::EDAnalyzer {
  ... ... ...
  float SumAdc(const std::vector< art::Ptr<recob::Hit> > & hits) const;
  float fAdcSum;
};
void MyClusterCounter::beginJob()
{  ... ... ...
   fClusterTree->Branch("adcsum", &fAdcSum, "fAdcSum/F");
}
void MyClusterCounter::analyze(art::Event const & evt)
{
   auto cluHandle = evt.getValidHandle< std::vector<recob::Cluster> >(fClusterModuleLabel);
   art::FindManyP< recob::Hit > hitsFromClusters(cluHandle, evt, fClusterModuleLabel);
   fNClusters = 0;
   for (size_t i = 0; i < cluHandle->size(); ++i)
   {
      fNHits = cluHandle->at(i).NHits();
      fAdcSum = SumAdc( hitsFromClusters.at(i) );
      fClusterTree->Fill();
       mf::LogVerbatim("MyClusterCounter")
        << "NHits() = " << fNHits << ", assn size = " << hitsFromClusters.at(i).size()
        << " SummedADC() = " << cluHandle->at(i).SummedADC() << ", sum hits adc = " << fAdcSum;
      if (fNHits >= fMinSize) { ++fNClusters; }
   }
   fEventTree->Fill();
}
float MyClusterCounter::SumAdc(const std::vector< art::Ptr<recob::Hit> > & hits) const
{
   float sum = 0;
   for (auto const & h : hits) { sum += h->SummedADC(); }
   return sum;
}
```

← include headers for the two new types

← declare function to sum up ADC's

← ..and a place for result

← ..and a branch for histo

← this time not-dereferenced handle is needed

← **reads associations made by cluster producer between clusters in cluHandle and recob::Hit's**

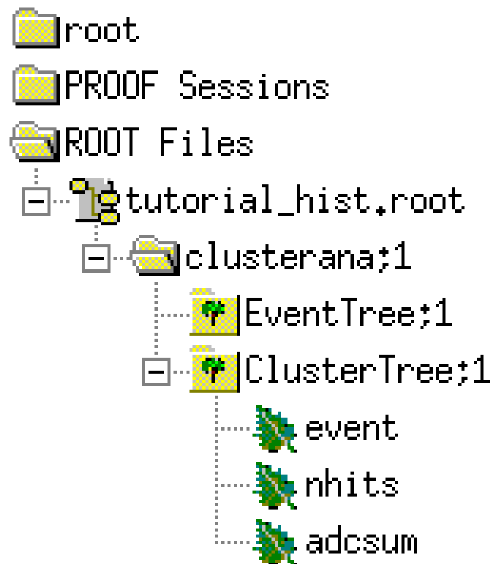
← use **vector of art::Ptr<recob::Hit>** associated to *i*'th cluster

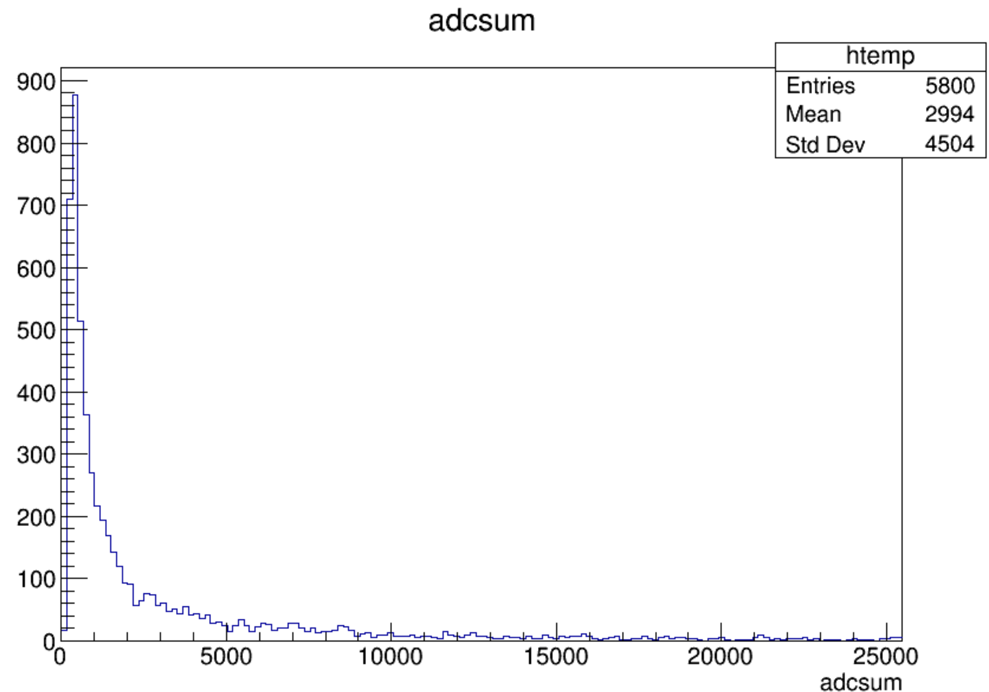
← number of hits stated by cluster and assns

← ADC sum read from cluster and directly from hits

# …compile, run and produce histogram

```
[robert@localhost MyWork]$ cd $MRB_BUILDDIR
[robert@localhost build_slf6.x86_64]$ make install -j2
…
[robert@localhost build_slf6.x86_64]$ cd ..
[robert@localhost v6]$ lar -c job/my_job.fcl  your_path_to_data/example_reco.root
…
[robert@localhost v6]$ root tutorial_hist.root
…
root [0] TBrowser t;
```





## Which MC truth particle is matching the cluster?

(or use ClusterCounter4_module.cc)

```
const simb::MCParticle* MyClusterCounter::getTruthParticle(
   const std::vector< art::Ptr<recob::Hit> > & hits,
   float & fraction, bool & foundEmParent) const
{
   const simb::MCParticle* mcParticle = 0;
   fraction = 0;
   foundEmParent = false;
   art::ServiceHandle<cheat::BackTracker> bt;
   std::unordered_map<int, double> trkIDE;
   for (auto const & h : hits)
   {
      for (auto const & ide : bt->HitToTrackID(h)) { trkIDE[ide.trackID] += ide.energy; }
   }
   int best_id = 0;
   double tot_e = 0, max_e = 0;
   for (auto const & contrib : trkIDE)
   {
      tot_e += contrib.second;
      if (contrib.second > max_e)
      {
         max_e = contrib.second;
         best_id = contrib.first;
      }
   }
   if ((max_e > 0) && (tot_e > 0)) // ok, found something reasonable
   {
      if (best_id < 0) { best_id = -best_id; foundEmParent = true; }
      mcParticle = bt->TrackIDToParticle(best_id);
      fraction = max_e / tot_e;
   }
   else { mf::LogWarning("MyClusterCounter") << "No energy deposits??"; }
   return mcParticle;
}
```

MCParticle at Doxygen documentation
BackTracker at Doxygen documentation

← **BackTracker can e.g. find MC truth energy deposits which contributed to reconstructed hits**

← loop over std::vector<**sim::TrackIDE**> and sum energy contribution by each G4 track ID

← sum total energy in provided hits

← find track ID corresponding to max energy

← **BackTracker finds MCParticle corresponding to track ID**

## Which MC truth particle is matching the cluster?

(or use ClusterCounter4_module.cc)

```
#include "larsim/MCCheater/BackTracker.h"
#include "nusimdata/SimulationBase/MCParticle.h"
class MyClusterCounter : public art::EDAnalyzer {
  ... ... ...
  const simb::MCParticle* getTruthParticle(
      const std::vector< art::Ptr<recob::Hit> > & hits,
      float & fraction, bool & foundEmParent) const;
  float fClean;
};
void MyClusterCounter::beginJob()
{   ... ... ...
    fClusterTree->Branch("clean", &fClean, "fClean/F");
}
void MyClusterCounter::analyze(art::Event const & evt)
{   ... ... ...
    for (size_t i = 0; i < cluHandle->size(); ++i)
    {   ... ... ...
        bool isEM = false;
        const simb::MCParticle* p = getTruthParticle(hitsFromClusters.at(i), fClean, isEM);
        if (p)
        {
            if (isEM) { mf::LogVerbatim("MyClusterCounter") << "matched mother particle PDG: " << p->PdgCode(); }
            else { mf::LogVerbatim("MyClusterCounter") << "matched particle PDG: " << p->PdgCode(); }
        }
        else { mf::LogWarning("MyClusterCounter") << "No matched particle??"; }
        fClusterTree->Fill();
         mf::LogVerbatim("MyClusterCounter")
            << "NHits() = " << fNHits << ", assn size = " << hitsFromClusters.at(i).size()
            << " SummedADC() = " << cluHandle->at(i).SummedADC() << ", sum hits adc = " << fAdcSum;
        if (fNHits >= fMinSize) { ++fNClusters; }
    }
    fEventTree->Fill();
}
```

← include more headers for new types

← function to match MCParticle to hits

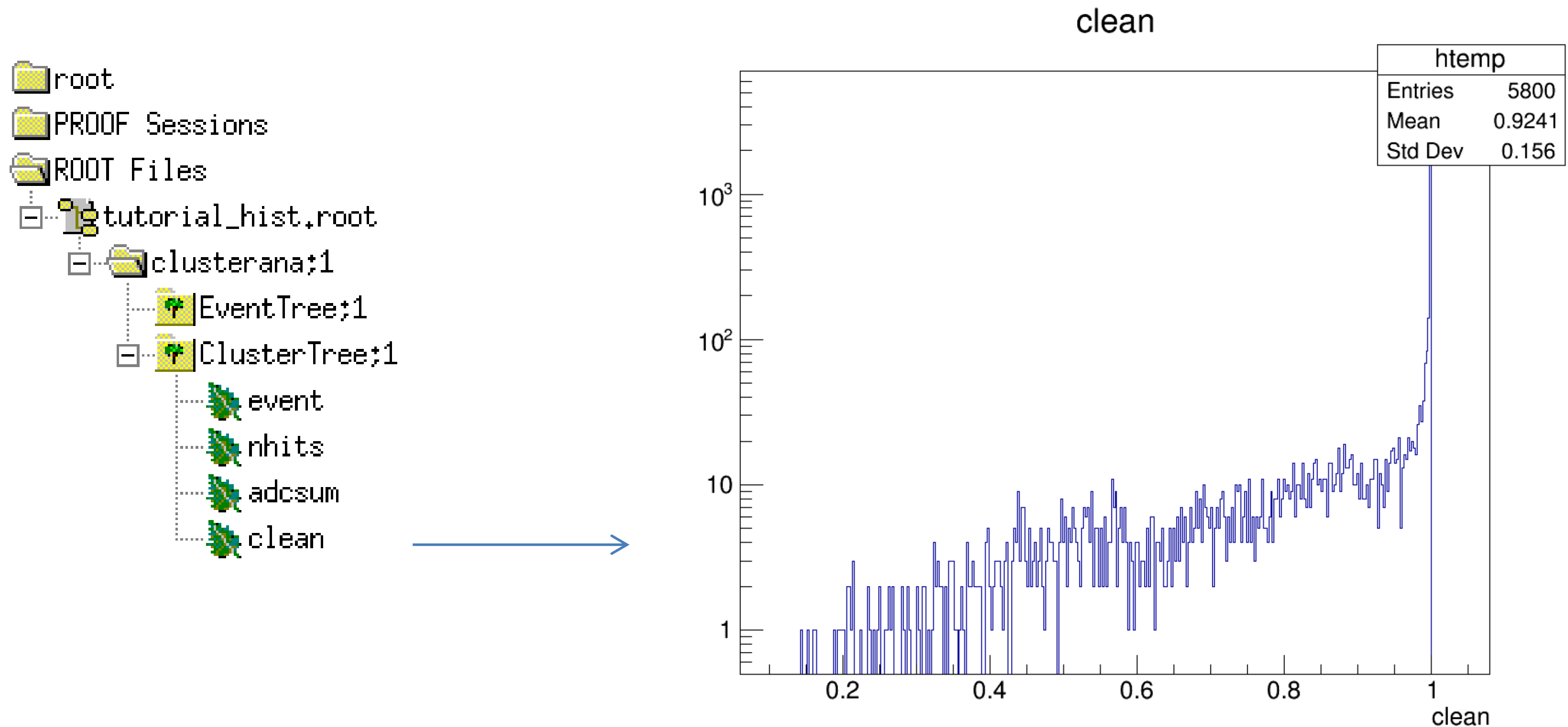
← ..and a place for some result

← ..and a branch for histo

**← use matching function**

## …compile, run and produce histogram

[robert@localhost MyWork]$ **cd $MRB_BUILDDIR**
[robert@localhost build_slf6.x86_64]$ **make install -j2**
…
[robert@localhost build_slf6.x86_64]$ cd ..
[robert@localhost v6]$ **lar -c job/my_job.fcl your_path_to_data/example_reco.root**
…
[robert@localhost v6]$ root tutorial_hist.root
…
root [0] TBrowser t;



# Little effort to make the FHiCL paremeters validated

(or use ClusterCounter5_module.cc)

```
#include "fhiclcpp/types/Atom.h„
#include "fhiclcpp/types/Table.h„
#include "fhiclcpp/types/Sequence.h”
// #include "fhiclcpp/ParameterSet.h”
class MyClusterCounter : public art::EDAnalyzer {
public:
  struct Config {
    using Name = fhicl::Name;
    using Comment = fhicl::Comment;
    fhicl::Atom<art::InputTag> ClusterModuleLabel {
      Name("ClusterModuleLabel"), Comment("tag of cluster producer")
    };
    fhicl::Atom<size_t> MinSize {
      Name("MinSize"), Comment("minimum size of clusters")
    };
  };
  using Parameters = art::EDAnalyzer::Table<Config>;
  explicit MyClusterCounter(Parameters const& config);
  // explicit MyClusterCounter(fhicl::ParameterSet const & p);

private:
  art::InputTag fClusterModuleLabel;
  size_t fMinSize;
};

MyClusterCounter::MyClusterCounter(Parameters const& config) : art::EDAnalyzer(config),
  fClusterModuleLabel(config().ClusterModuleLabel()),
  fMinSize(config().MinSize())
{}
```

← int, float, …
← FHiCL tables: { param1: value1 … }
← FHiCL vectors: [val1, val2, …]

See more validation in this examples: https://cdcvs.fnal.gov/redmine/projects/larexamples

## Some useful commands and links

Check the final shape of FHiCL parameters:
lar -c job/my_job.fcl example_reco.root **--config-out debug_out_file**

Check where are all the contributing FHiCLs:
lar -c job/my_job.fcl example_reco.root **--config-out debug_out_file --annotate**

Doxygen: http://nusoft.fnal.gov/larsoft/doxsvn/html/index.html
– very convenient to use „Search” to find class, function, type, …
– FHiCL files are not linked there

Redmine:
– LArSoft (click one of packages and go to repository): https://cdcvs.fnal.gov/redmine/projects/larsoft
– dunetpc: https://cdcvs.fnal.gov/redmine/projects/dunetpc/repository

Modules, code examples:
– best practice examples, very well described:
https://cdcvs.fnal.gov/redmine/projects/larexamples
– MC dumpers:
https://cdcvs.fnal.gov/redmine/projects/larsim/repository/revisions/develop/show/larsim/MCDumpers
– DumpHits module:
https://cdcvs.fnal.gov/redmine/projects/larreco/repository/revisions/develop/show/larreco/HitFinder

**Once more, reach info, slides, videos (!) on LArSoft workshop at FNAL:**
http://larsoft.org/larsoft-workshop-report-august-2016/

git flow feature finish cern_Tutorial