

A getting started example to LArSoft and dunetpc

We want to do the stuff I outlined in the first talk.

1. Logon to dunebuild01.fnal.gov
 - a. `kinit php13tkw@FNAL.GOV`
 - b. `ssh php13tkw@dunebuild01.fnal.gov`
2. Setup the local environment
 - a. `source /grid/fermiapp/products/dune/setup_dune.sh`
3. Make sure you are in the /dune/app/ area
 - a. `cd /dune/app/users/USER`
4. Setup LArSoft – we are going to set up an old build so we can do an extra step.
 - a. `ups list -aK+ larsoft ##### List all LArSoft releases`
 - b. `setup larsoft v06_01_00 -q e10:prof`
5. Make a new directory for your LArSoft build.
 - a. `mkdir larDev`
 - b. `cd larDev`
6. Make a new development
 - a. `mrbs newDev`
7. Source the new LocalProducts
 - a. `source localProducts_*/setup`
8. Go to the srcs directory and checkout dunetpc
 - a. `cd $MRB_SOURCE`
 - b. `mrbs g -t v06_01_00 dunetpc`
9. Go to the build directory and build
 - a. `cd $MRB_BUILDDIR`
 - b. `mrbs setenv`
 - c. `setup ninja v1_6_0`
 - d. `mrbs i -j16 --generator ninja`
10. Set our local products to make sure we use them.
 - a. `mrbslp`

However, v06_01_00 of dunetpc is an old release (it was the last release in July '16), so if we want to use the newest version of the code we need to update to v06_XX_XX!

1. Logout of dunebuild01 and relogin :D
 - a. `ssh php13tkw@dunebuild01.fnal.gov`
2. Setup the local environment
 - a. `source /grid/fermiapp/products/dune/setup_dune.sh`
3. Make sure you are in the /dune/app/ area
 - a. `cd /dune/app/users/USER`
4. Setup LArSoft – we are going to set up an old build so we can do an extra step.
 - a. `setup larsoft v06_XX_XX -q e10:prof`
5. Change to the LArSoft directory.
 - a. `cd larDev`
6. Make a new development using the old srcs directory
 - a. `mrbs newDev -p`
 - b. `source localProducts_<NEW VERSION>`
7. Move to dunetpc and pull develop
 - a. `cd srcs/dunetpc`

- b. `git checkout develop`
 - c. `git pull`
8. If you are working on a feature branch, you want to merge the new develop to your feature branch
 - a. `git checkout feature/php13tkw_Test`
 - b. `git merge develop`
9. Do the same for all of your other repositories
10. Go to the build directory and build
 - a. `Cd $MRB_BUILDDIR`
 - b. `mrbs z`
 - c. `mrbssetenv`
 - d. `setup ninja v1_6_0`
 - e. `mrbs i -j16 --generator ninja`
11. Set our local products to make sure we use them.
 - a. `mrbslp`

We now have the most up-to-date version of dune in our local area, so now we want to do some work!

We don't want to use the build node for anything other than building, so we want to change node to one of the gpvm's. **Setting everything up upon a fresh login**

1. Logout of dunebuild01 and relogin to dunegpvm(01-10).fnal.gov
 - a. `ssh php13tkw@dunegpvm05.fnal.gov`
2. Get your local area ready – you may want to make a shell script to do this. Note C,D are for submitting jobs to the cluster.
 - a. `source /grid/fermiapp/products/dune/setup_dune.sh`
 - b. `setup ninja v1_6_0`
 - c. `./grid/fermiapp/products/common/etc/setups.sh`
 - d. `setup jobsub_client`
 - e. `source /dune/app/users/USER/larDev/localProd*/setup`
 - f. `cd $MRB_BUILDDIR`
 - g. `mrbssetenv`
 - h. `mrbslp`
 - i. `mrbslp`
 - j. `cd $MRB_TOP`

It is probably a good idea to have a look through some of the mrbs commands which we have just used to understand them and the different options we can use with the commands.

- `mrbs newDev -h ##` When we made a new development area
- `mrbs g -h ##` When we got a new repository
- `mrbs b -h ##` To just build, not install
- `mrbs I -h ##` To build and install
- `mrbs z -h ##` To delete everything in build area
- `mrbssetenv -h ##` To setup a development area
- `mrbslp -h ##` To setup products in localProducts area
- `mrbs uv -h ##` Update a product version
- `mrbs uc -h ##` Update the master CMake – when you checkout a new repository

It is a good idea to do work on a feature branch, so that any code you develop can be shared with people easily (pushing everything to develop isn't necessary / is undesirable).

For now, we will only do steps 1 – 4.

1. Make sure that everything is setup, local products etc.
2. Change desired repository eg dunetpc
 - a. `cd $MRB_SOURCE/dunetpc`
3. Make sure you have the latest develop
 - a. `git checkout develop`
 - b. `git pull`
4. Make a new feature branch, command has form
 - a. `git flow feature start USER_SPECIFIER`
 - b. `git flow feature start php13tkw_Test`
5. <<<< Do some coding >>>>
6. Check what code you have been changing
 - a. `git status` ## Gives a list of the modules you have changed / added
 - b. `git diff` ## Gives a line by line description of what you've changed
7. <<<< Do some coding >>>>
8. Push to your own feature branch if you aren't ready to publish it yet.
 - a. This means that you can store your work for later, but only you can access what is on there.
 - b. `git add my_file.cc`
 - c. `git commit -m "I've done this, that and the other"`
9. <<<< Do some coding >>>>
10. Publish your feature branch
 - a. `git flow feature publish php13tkw_Test`
11. <<<<< Do some more coding >>>>>
12. Project is finished, so now either delete feature branch or merge it into develop
 - a. Delete feature branch, i deletes locally, ii deletes in origin.
 - i. `git branch -d feature/php13tkw_Test`
 - ii. `git branch -dr origin/feature/php13tkw_Test`
 - b. Merge it into develop
 - i. `git flow feature finish php13tkw_Test`

Let's add a new directory and module to dunetpc!

1. Move to where all the code in dunetpc resides and make a new directory HitDumper
 - a. `cd $MRB_SOURCE/dunetpc/dune/`
 - b. `mkdir HitDumper`
2. Open CMakeLists.txt and add the relevant line to it
 - a. `emacs CMakeLists.txt`

```
add_subdirectory(SpaceCharge)
add_subdirectory(SpaceChargeServices)
add_subdirectory(HitDumper)
```
 - b. `add_subdirectory(HitDumper)`
3. Copy the HitDumper module, fcl files and CMLists.txt from my local area.
 - a. `cp /dune/app/users/php13tkw/HitDumper/* HitDumper/`
4. Check that it uses `art_make` – so our module will be built, if not add it to the list of modules that will be built.
 - a. `emacs HitDumper/CMakeLists.txt`
5. Look around the module and the fcl files to satisfy yourself as to how they work. Notice how the fcl parameters are changed in `RunHitDump_35ton.fcl` and how fcl parameters are accessed in the module.
6. Go to the build directory and do a clean build
 - a. `cd $MRB_BUILDDIR`
 - b. `mrbs z`
 - c. `mrbs setenv`
 - d. `mrbs i -j16 --generator ninja`

If you do this on the build machine, then you need to do the following:

- `ssh php13tkw@dunebuild01.fnal.gov`
- `source /grid/fermiapp/products/dune/setup_dune.sh`
- `cd /dune/app/users/USER`
- `source localProducts_*/setup`
- `cd $MRB_BUILDDIR`
- `mrbs setenv`
- `setup ninja v1_6_0`
- `mrbs i -j16 --generator ninja`

Obviously you don't need to add a new directory every time you make a new module.

- If the subdirectory you want to put your module has a `CMakeLists.txt` with `art_make` then you just need to add the module and do a clean build.
- If it doesn't you will either have to change it to use `art_make` or add it to the list of modules described in it, see `dunetpc/dune/TrackingAna/CMakeLists.txt`

Now that we have a new analysis module on our feature branch and have everything setup let's simulate some things. Most work will no longer be done with the 35 ton, but it's small geometry means that things will run faster, which is all we care about at the moment.

Note that you need to specify the file you want to run over correctly for steps 4,5,6 and 7.

1. Go to the top of LArSoft development area and a directory for our work
 - a. `cd $MRB_TOP`
 - b. `mkdir workspace`
 - c. `cd workspace`
2. Generate 10 CRY events.
 - a. `lar -c prodcosmics_dune35t_owindow.fcl -n 10`
3. As FHiCL is configurable we can change things in the simulation with fcl parameters. CRY lets you simulate only muons or longer time spans for example
 - a. I have a script in my `.bashrc` which locates fcl files
 - b. Copy the fcl file here
 - i. `cp $MRB_SOURCE/dunetpc/fcl/dune35t/gen/single/prodcosmics_dune35t_owindow.fcl`
 - c. List the fcl parameters to see what you want to change. Note that the parameters to change which particles are generated are given default values in the module ie they are not listed by `ART_DEBUG_CONFIG`.
 - i. `ART_DEBUG_CONFIG=1 lar -c prodcosmics_dune35t_owindow.fcl`
 - d. Open the fcl file and change whatever you like
4. Run GEANT4 on the output
 - a. `lar -c standard_g4_dune35t.fcl prodcosmics...`
5. Run the detector simulation on the output
 - a. `lar -c standard_detsim_dune35t.fcl prodcosmics...`
6. Run the reconstruction on the output
 - a. `lar -c standard_reco_dune35t.fcl prodcosmics...`
7. Run the analysis module we just put in `dunetpc` on the output
 - a. `lar -c RunHitDump_35ton.fcl prodcosmics...`

Now we have a fully reconstructed sample and our output we can do two things, look at it on the event display and look what our analysis module did.

We will do the event display first.

Note that the event display won't be the fastest, but there is a trick we can do.

I don't know how much you guys know about running GUIs on the gpvms....

STEP 1: Start a VNC server on the gpvm

Log into the gpvm of your choice.

Start the VNC server. The command is:

- `vncserver :X`

where *X* is a number of your choice. In my case, I chose 8, so my command was:

- `vncserver :8`

The number specifies the display you are going to use. I don't think this will work if someone is already using that display so, in that case, it may demand you use a different one.

If this is the first time setting up a server, it will ask you to pick a password. Pick one.

STEP 2: Push the output of a remote terminal to the VNC desktop

In a terminal where you are connected to the gpvm and are doing work which requires a GUI, issue the following command:

- `export DISPLAY=localhost:X`

Where *X* is the number you chose. In my case the command was:

- `export DISPLAY=localhost:8`

STEP 3: Tunnel the VNC through ssh to keep it all encrypted

On your local machine, via a terminal, issue the following command:

- `ssh -L 59X:localhost:59X -N -f -l USERNAME GPVMADDRESS`

Something to be aware of here. The port forwarding for VNC is via ports 59[0..99] and for numbers less than 10 you have to include the leading 0. So my command was

- `ssh -L 5908:localhost:5908 -N -f -l dbrailsf dunegpvm06.fnal.gov`

The `-L 5908:localhost:5908` says forward information from the local side on port 5908 to the remote host via its port 5908. You have to make sure that the port number (in my case 08) matches with the display used when setting in the vncserver.

STEP 4: Open the VNC window locally

Tell whatever local vncviewer you have installed to open the localhost window using the port (localhost:59X or localhost:5908 in my case). For mac users, there is already one installed which you can access very easily via open. The mac command is

- `open vnc://localhost:59X`

In my case, the command was:

- `open vnc://localhost:5908`

A desktop window should pop up. In any remote terminal window in which you have pushed the output to the VNC window (like step 2), the GUIs should open in this desktop.

STEP 5: Open a GUI remotely and watch it appear in the desktop window

Go back to the terminal in step 2 and open a GUI. A quick test would be a TBrowser

- `root -l`
- `new TBrowser`

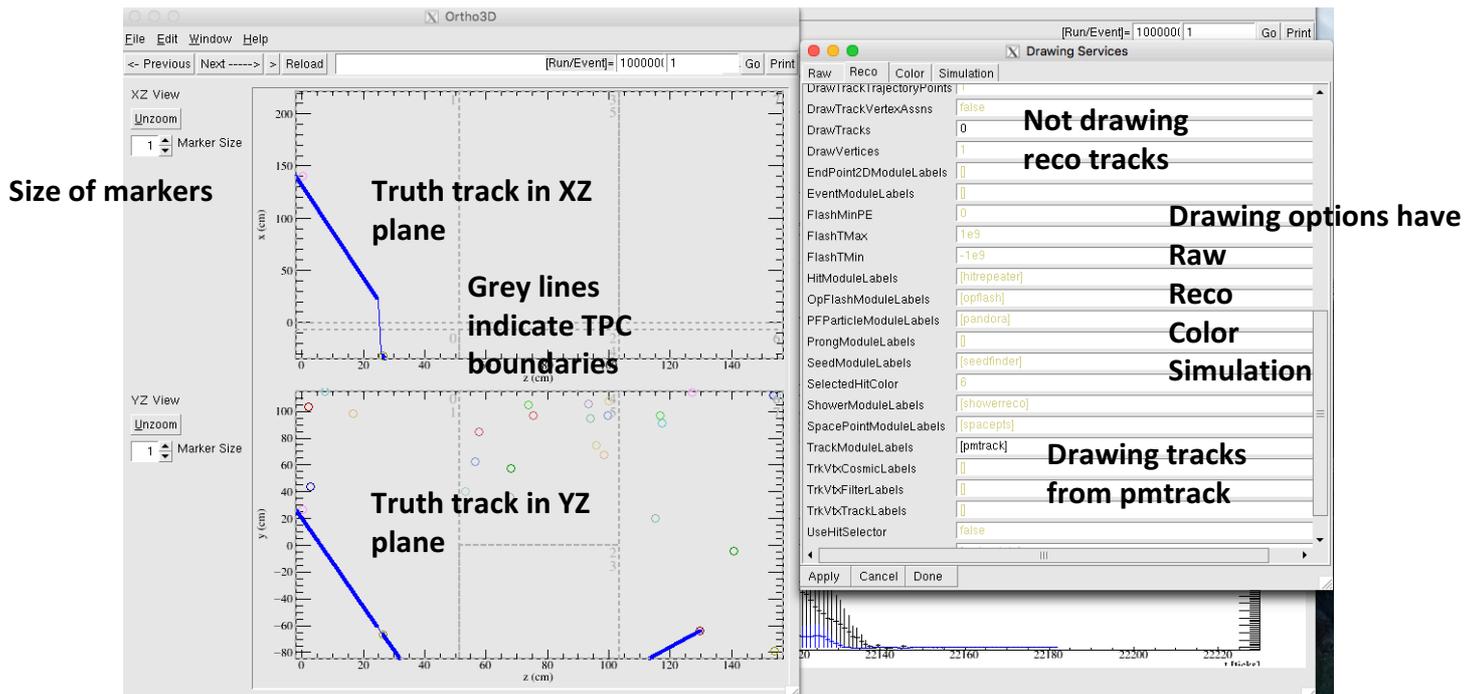
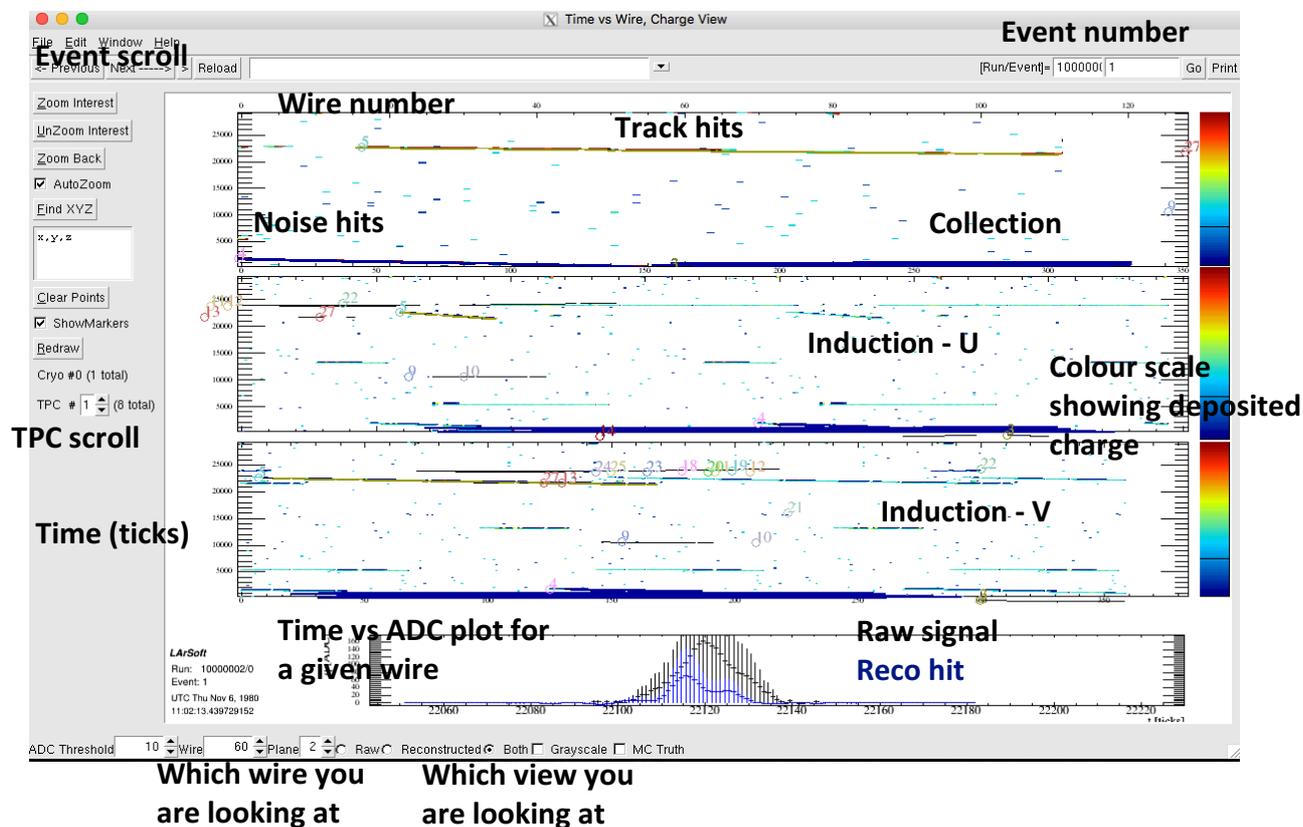
Confirm that the TBrowser opens in the VNC desktop window. Try doing stuff with it and note how quick it is.

I also believe that you can push multiple terminal displays to a single VNC server so you could repeat step 2 for a bunch of different terminal windows.

The default LArSoft event display is very powerful but also very confusing at first. It is ran with the command:

- `lar -c evd_dune35t.fcl prodcosmics*_reco.root`

This will generate a new window which looks something like this, some of the key features have been labelled:



A quick aside about the 35 ton detector for those who don't know.

- It has 4 APAs, and two drift directions giving 8 TPCs (numbered 0-7) in total
- It had cosmic ray scintillator paddles around the outside
- It had photon 8 SiPMs inside the 4 APAs to collect flashes.
- It had 2 induction planes, and one collection plane on each TPC.
- The wires were wrapped, meaning that the induction wires were in multiple TPCs eg 0 and 1.

I can draw some diagrams / explain it all properly to anyone that wishes me to do so.

1. Scroll through the TPCs for event 1 to see what raw signals we generated.
2. Click on the reconstruction tab and do the same to see the reconstructed tracks, see how well they line up.
3. Open the Ortho3D Viewer.
4. Open the Simulation tab option -
 - a. Set *ShowMCTruthTrajectories* to *true*.
 - b. Click Apply. The two windows should refresh and look different, if they don't click update (or do it anyway just to make sure).
 - c. Hopefully you will notice that some blue lines have appeared on the Ortho3D tab – these are the trajectories which our particles took through the detector
5. Change from the Simulation to the Reconstruction tab.
 - a. Set the following parameters – note you need to press enter after you change each field, the number / string will turn from grey to black.
 - i. *DrawTracks* – 2 (shows the track number)
 - ii. *TrackModuleLabels* (change from *cheated* to *non-cheated reco*)
 - iii. *HitModuleLabels* (change from *cheated* to *non-cheated reco*)
 - iv.
 - b. Click Apply. The two windows should refresh and look different, if they don't click update (or do it anyway just to make sure).
 - c. If you want you can also show the OpFlash information on the event display by setting the following parameters:
 - i. *DrawOpFashes* - 1
6. Scroll through the events doing the following:
 - a. Switch between the raw, reconstructed and both views
 - b. Switch between showing cheated and non-cheated hits
 - c. Look at how the truth tracks (in Ortho3D) relate to tracks in the event display window by scrolling through TPCs.

Using the cluster and project python

Use of the cluster is identical to as you do in NOvA, but you submit using your dune credentials not NOvA ones!

I think this just means having a different KCA certificate and using the `--group dune` option.

I don't think that you have project python in NOvA though. This is a really handy tool developed by Herb Greenlee which takes a lot of the hassle out of job submission for you.

- Project python uses XML files to do all of the behind the scenes preparation for you jobs.
- It takes multiple 'stages', each represented by a fcl file and submits jobs, checks whether they are running, and then checks that the files produced are 'good.'
- Allows you to give it a list of files which you want it to load in.
- Runs jobs using either the head of develop, or your local products (if you do this you will want to make them into a tarball)
- Explained fairly thoroughly [here](#) on the dunetpc wiki.

Let's do a set of 500 anti-muons in the 35 ton detector to test it out.

1. Make a tarball of your local products.
 - a. `mkdir $MRB_TOP/tarballs`
 - b. `cd $MRB_TOP/tarballs`
 - c. `make_tarball.sh v06_01_00.tar`
2. Make a directory where you want to have all the XML files (there are ~110 atm)
 - a. `mkdir $MRB_TOP/XML`
 - b. `cd $MRB_TOP/XML`
3. Get the xml files, using your local products and username
 - a. `make_xml_mcc.sh -u USER --local /dune/app/users/USER/larDev/tarball/v06_01_00.tar -r v06_01_00`
4. Using the wiki page do the following to `AntiMuonCutEvents_LSU_dune35t.xml`
 - a. Change the number of events from 10,000 to 500
 - b. Change the number of jobs from 100 to 5
 - c. Add a new stage after mergeana which uses the fcl file for your analysis module and uses the output from the reco stage as its input.

There are two ways to submit jobs using project python, for completeness' sake we will do both.

- Open up a gui and submit via there.
 - Covered substantially on the wiki.
 - `Projectgui.py AntiMuonCutEvents_LSU_dune35t.xml`
 - << GUI opens up >>
 - Click "Gen" on the LHS – the row will change colour
 - Click "submit" – the idle column will increase from 0 to 5
 - << Watch as jobs move from idle to running >>
 - When all 5 jobs have completed, click "Check"
 - << Hopefully, the Events column will show 500 and GoodFiles will show 5 >>
- Submit the jobs via the command line.
 - I assume you have done the above for the Gen stage, so we will submit g4.

- `project.py --xml AntiMuonCutEvents_LSU_dune35t.xml -`
`-stage gen --check`
- << Should get an output which says 5 good files >>
- `project.py --xml AntiMuonCutEvents_LSU_dune35t.xml -`
`-stage g4 --submit`
- << Have now submitted the g4 stage jobs >>
- `project.py --xml AntiMuonCutEvents_LSU_dune35t.xml -`
`-stage g4 --update`
- << Should show that X jobs are idle, Y jobs are running >>
- `project.py --xml AntiMuonCutEvents_LSU_dune35t.xml -`
`-stage g4 --check`
- << If all the jobs have finished running, then hopefully we will have 5 good files >>
- For more options as to how to use the command line use:
 - `project.py --help`
- Go all the way through all the other stages as you see fit, making sure that all of the processes in the previous stage have finished before submitting the next stage.

There is one additional step which we can do for the analysis stages – merging the output TFS histogram files.

- On the command line
 - `project.py --xml AntiMuonCutEvents_LSU_dune35t.xml -`
`-stage analysis --mergentuple`
- Using the GUI.

There is an example of how I submit jobs to the cluster when not using python here:

- `/dune/app/users/php13tkw/LarDevelop/workspace/Splitter/GoodRun`
`List/SubmitGoodRunJobs/`
- `source TheSubmissionScript.sh`

There is an example of using the PROCESS variable here:

- `emacs -nw /dune/app/users/jti3/mytest/job/runjob.sh`
- `jobsub_submit -N 4342 -q --OS=SL6 --group=dune --resource-`
`provides=usage_model=OPPORTUNISTIC`
<file:///dune/app/users/jti3/mytest/job/runjob.sh>

Now to look at what the analysis module did.

The module simply makes a TTree with the reconstructed hit information. We can look at this information using simple TTree draw commands or using a TBrowser.

We can build on this example module to make it do the following:

- Also write out some tracking information
 - We can further write out which hits are associated to what track
 - We can see what particle caused the tracks.
 - There are art associations in the event which you can use to get this.
- Also write out some of the cosmic ray counter information (ExternalTrigger)
- Also write out some of the reconstructed photon detector information (OpFlash)
- We can try to associate flashes and external triggers with tracks.
- Want to add histograms to the TTree
- Want to use some random function defined in another class.
 - Have a think about something that could possibly make sense, if not just use the Counter map function in daqinput35t/PennToOffline.cc.

Things we need to try to do:

- Associations, both creating new ones and accessing existing ones
 - Need to make a producer to do this.
 - Have a look at MCTruth_module.
- Accessing lots of different data members – use DOxygen
 - Try to follow through how calorimetry works, just to get used to the formatting of the LArSoft DOxygen.

For some pre-defined examples of code we can look in larexamples.