Coding with dunetpc and LArSoft: tips, tricks, gotchas, and debugging advice

Tom Junk

DUNE Computing Tutorial

November 14, 2017

The Tutorial Example

- Instructions are available at the bottom of the homework link:

- https://cdcvs.fnal.gov/redmine/projects/dune/wiki/Getting_Started_with_DUNE_Computing

- It is one of many similar "getting started" exercises. This one's tested with the setup used for the homework,

  dunetpc v06_34_00

- I won't go through the steps one by one, try them on your own!

The Tutorial Example

- Purpose: Start with an existing analysis module

  larexamples/AnalysisExample/AnalysisExample_module.cc
- and a corresponding fhicl file:

  larexamples/AnalysisExample/AnalysisExample.fcl
- Move it to a new directory
- Rename it to be your own
- Modify build files and check that it builds and runs in its new home
- Add a histogram of the integrated ADC hit charge
- Build and run it.

cetskelgen

- cetskelgen is a command-line utility provided with *art* to generate skeleton examples of modules, producers, filters, sources, and other plugins for *art*.
- cetskelgen --help tells you what you need to know to run it
- Lots of art tools: Go to the *art* wiki to find other tools (bookmark this page!) https://cdcvs.fnal.gov/redmine/projects/art/wiki
  - config_dumper -P
  - fhicl-dump (similar to lar –debug-config <fcl_output_file> -c <input_fcl_file>)
  - fhicl-expand
  - file-info-dumper
  - count_events
  - product_sizes_dumper
  - sam_metadata_dumper
  - cetskelgen
- Note from R. Hatcher: fhicl-dump and fhicl-expand need fcl files that can be found in FHICL_FILE_PATH. A full pathname to a fcl file will result in "file not found" (unexpected but known behavior)

Help with git in the LArSoft Environment

- Many centrally-managed repositories
- Smallest unit of build = the repository
- Check out with mrb g or git clone
- Need to be a developer to get push access
- Tutorial at https://cdcvs.fnal.gov/redmine/projects/dunetpc/wiki/_Tutorial_
- See also the LArSoft documentation https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/Developing_With_LArSoft
- MRB documentation is available at https://cdcvs.fnal.gov/redmine/projects/mrb/wiki

Speeding up the build

- The first time you build the code in a repository (which maps to a UPS product), you need to use mrb i (parses the CMakeLists.txt files, runs make, and installs in local products).
- I always have the current working directory as $MRB_BUILDDIR when doing this.
- Debugging often requires building, rebuilding, and building again.
- Subsequent times you can issue the command make install in $MRB_BUILDDIR
- To distribute the build work across n cores, type mrb –i –j <n> or make install –j <n>
- This is only really effective if you have n cores to use.
- dunebuild01.fnal.gov has 16 cores. Use only for building code!
- Run and test code on a dunegpvm*.fnal.gov machine
- Try using ninja instead of make. Documentation is on the LArSoft code development wiki.
- Try using *studio*, which allows you to develop an *art* module, compile and link it, without rebuilding dunetpc. I haven't tried it yet, but the idea sounds attractive. https://cdcvs.fnal.gov/redmine/projects/studio/wiki

Common "Gotchas"

- Version Mismatch:
  - ups is very good at setting up consistent versions of software
  - When you check out and build code though, you have to watch the versions. Especially if you work on a project over many weeks.
  - Other developers are working on their projects, committing to LArSoft
  - Weekly releases to make committing things to multiple repositories that depend on each other less painful. Less "if you want to work with the latest larreco, you have to also check out the latest lardataobj" sorts of things.
  - The development head of each repository has a version that updates pretty much every week. Checking it out with mrb g will almost certainly conflict with an old prebuilt release.
  - Version numbers are defined in ups/product_deps for each ups product and the ones it depends on.
- Solution(s): Either
  - check out tagged code and work with a stable release (in the instructions), or
  - constantly update to a new release

Updating your Local Release to New Versions

- LArSoft releases once a week, and dunetpc follows for each of these, and extra releases as needed.
- Checking out the development head of something will cause version clashes.
- Sometimes you really want the new version of some other ups product, and want to move your test release to depend on the new larsoft code.

- https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/Tips_on_updating_your_code_after_LArSoft_release

Common "gotchas"

- Art cannot find module with compiler type xxxx
  - check to make sure the build was successful!
  - Look in the build messages to make sure the compile happened. Rebuild with a trivial change to the source if you need to see it again.
  - Check the CMakelists.txt file – missing a block for your new module? Look at a similar example in dunetpc or larsoft.
  - A new CMakelists.txt file or one with art_make in it with a new module requires mrb i to be run to parse all the CMakelists.txt files and make new makefiles. If you are used to running make install, it may miss your new module without the first mrb i step.
  - Look in the local products library directory for your machine architecture – is your module library built? Maybe not installed? make install or mrb i should install it
  - You may have old clutter. mrb z will clear out $MRB_BUILDDIR for a fresh build.
  - Forgot to type mrbslp? Sets up local products and puts them before the pre-installed versions in the search paths.

Common Gotchas

- Undefined Symbol
  - This is a linker error, and usually can be remedied by putting the right library name in the CMakelists.txt file
  - I always look for examples in other CMakelists.txt files to see how the names are assigned.
  - Look through other code to see if someone else is using the symbol you're trying to use and look in the corresponding CMakelists.txt file
  - Not to be confused with undefined or multiply-defined variables.

Common Gotchas

- File "clobbering"
  - Most commands let you overwrite files if you make a new one with the same name in the same directory. But this is settable
    set –o noclobber
  - Sometimes this is desired, sometimes not
  - A case in which it is not – run many jobs on different data samples, but each job writes its output to the same output file ("I ran 100 jobs and only got 1 output file!")
  - Also: Running 100 jobs with the same random seed – you get 100 output files but they contain identical events. This is often visible in histograms with distributions that do not look Poisson
- Code clobbering – failing to merge your changes when someone else has modified code between your checkout and your desired push. Don't just erase the work of other people! Merge wisely. Automated merges must be checked.

A Common ifdh gotcha

- Forgetting –D on ifdh cp

  You get an error message saying ": is a directory" and no copy.

  An artifact of "dd" needing that option.

Miscellaneous Programming Tips

- I like the auto-indent and parentheses-matching and braces-matching features of emacs: esc-x indent-region helps find lots of block-level brace mistakes in C++.
- I always put the close brace right after the open brace before I start filling in the contents. That way I won't forget to close a brace.

- When counting objects, make sure to initialize the counter to zero! Remember your method will be called on every event, so a one-time initialization of a persistent variable (like one in the private block of your class) is not enough. Same with resizing vectors.

Programming Tips

- Program DEFENSIVELY! Paranoia == good!
  - Your analysis module, or producer, or filter will be called on MILLIONS of events if not more, and thus must be bulletproof.
  - Even if crazy stuff happens only one time in a million, it will happen.
  - Check for zero before dividing by anything.
  - Domains: sqrt(negative), log(zero or negative)
  - Check for out-of-bounds data accesses. Use vector::at()
  - What to do when a key is not in a map – make a new one or do something else? I use map::find if I don't want a new entry but want only to find old ones. map::find() will return <mymap>.end() if the key was not found.
  - Think about what to do with overflows in histograms.
  - What to do when data are missing or invalid. (My hit charge is negative! Now what?)
  - grep your log files for nan

Protecting yourself from yourself

- Commit frequently to your local git repository
- I also use some blunter techniques that make me feel better
  - Back up your work (new source files, fcl files, etc) to a directory in your home area, or your laptop, etc.
- /dune/app has snapshots labeled by their time: /dune/app/.snapshot
- Your home directory is also snapshotted /nashome/.snapshot

Reading Code of Others

- Expect to spend more time reading other people's code than writing new code.
  - finding examples. Simple questions like: "How do I access the run and event number?" How do I access the geometry info? etc. are most easily answered by searching for examples.
  - searching for bugs
  - documentation (writing papers – you sometime have to document or check someone else's documentation).
- Always keep questioning things –
  - variables that used to mean one thing but now mean another, but their name hasn't changed.
  - methods that are called at some times during program execution but their name suggests otherwise.

Checking Out Everything

- I always keep a checked-out copy of the source so I can grep it (grep –r –i is my favorite command here)
- I do not try to build all of this

```
#!/bin/sh
USERNAME=`whoami`
LARSOFT_VERSION=v06_42_00
COMPILER=e14
source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh
cd /dune/app/users/${USERNAME}
rm -rf inspect
mkdir inspect
cd inspect
setup larsoft ${LARSOFT_VERSION} -q debug:${COMPILER}
mrb newDev
source /dune/app/users/${USERNAME}/inspect/localProducts_larsoft_${LARSOFT_VERSION}_debug_${COMPILER}/setup
cd srcs
mrb g larsoft_suite
mrb g larsoftobj_suite
mrb g larutils
mrb g uboonecode
mrb g ubutil
mrb g larbatch
mrb g lbne-raw-data
mrb g dunetpc
mrb g duneutil
mrb g dunefgt
mrb g dune-raw-data
mrb g dune-artdaq
```

Finding Bugs

- Takes practice
- The hardest part is isolating the bug.
  - When is the last time in the program execution at which you think the program is working properly?
  - When is the first time you see evidence of failure?
  - Try to narrow the range between these as much as possible.
- It's good to have a working example to compare with. Sometimes this can be an earlier version of the same program.
  - My program worked yesterday! I only made a trivial change and now it produces wrong answers!
  - Maybe another change was made? Maybe it didn't really work yesterday? Maybe you just got lucky (uninitialized variables)
  - Keep track of changes with local git commits

Finding Bugs

- Random behavior:
  - I ran the same program twice and got a different answer!
    - Random number generator by default gives you different random seeds on each run. This is configurable to help you debug. Look for LArSeedService in your fcl configuration document. lar --debug-config <config_output_file> -c <your_fcl_file>
    - Uninitialized memory can cause unpredictable results. Sometimes you get lucky and results can be stable during testing and only become unpredictable for others.
    - Compilers are getting better at spotting uninitialized memory but they are not perfect
    - General advice – do not use random numbers when analyzing data. Data events may appear and disappear from selected samples if this rule is violated. Pseudorandom numbers seeded from the event number or some other property of the event raw data are OK. (35t's event splitter renumbered events, so watch out for that).

Finding Bugs

- Not clearing out counters or vectors between events
  - Counts of objects keep growing from one event to the next, becoming very large
  - Data products grow in size from one event to the next (summing over all previous events).

  - You can peek inside an art-formatted rootfile with tools like these:
    - lar -c eventdump.fcl <myfile.root>
    - Open in root and type Events->Print(); (lots of output – be patient!)
    - product_sizes_dumper <myfile.root>

Finding Bugs

- The dreaded segfault
  - means an attempt to access memory not allocated to your process, or a write access to read-only memory
  - I often need a debugger to find these, though a traceback can be useful.
  - Tracebacks can be corrupted. Local variables are stored on the stack, and memory corruption on the stack can cause the traceback to be unreliable.

Using Allinea ddt

- The GUI debugger we have available.
- ups list –aK+ allinea. The default version is v7_0_1 which is better than v5_1_0.
- ddt is the name of the debugger executable. It is a gui on top of gdb.
- ups sets up a version of gdb that corresponds to the debugger and the compiler. The gdb installed on the system is way out of date and will itself segfault when run with new larsoft code.
- Tips at https://cdcvs.fnal.gov/redmine/projects/dune/wiki/Getting_Started_with_the_Allinea_Forge_Debugger_and_Profiler

Using Allinea ddt

- Invoke the debugger. Note: ddt does not follow your PATH to find the program to debug! You have to give it a full path name. But this is easy:

  ddt `which lar` -c <myconfig.fcl> <myinputfile.root>

- If it complains about MPI use the option –nompi

- Look in the "default breakpoints" menu under "Control" and turn on the throw and catch breakpoints. Some programmers use throw and catch as normal program flow constructs; we discourage this. Only throw an exception if something is wrong.

Using Allinea ddt

- Buttons for "go", "pause", "next (step over)" and "step into" are my go-to things.
- Right-Click on a source line to set a breakpoint (mac users emulate 3-button mouse)
- Expressions window allows you to evaluate expressions and peek at memory
- Variables can be optimized away in the e14:prof builds. e14:debug lets you peek at all variables.
- You can even call some methods in the expressions window, like vector.size().

Using Allinea ddt

- ddt is very bad at discovering source files
- The .so files list the full paths to sources that were used to build them, but both the .so files and the sources get installed in ups product directories (e.g. CVMFS).
- ddt gets confused and thinks you've edited your source files.
- local products are easier as the source files are not moved.
- You can add a source file to its search list with the browse feature. I keep a second window open with the same environment setup as the debugger. Look in $<PRODUCT>_DIR/source (e.g. $DUNETPC_DIR/source) to find full paths to source files to point ddt at.

Stepping "backwards" Through a Program

- Often to isolate a bug, you have a place in the program where you know something is wrong, but not how the program got in that state.
- You can set a breakpoint that's earlier in the program, and restart the session (menu pulldown item). Startup arguments, breakpoints and source file locations will be remembered.
- Sometimes using the stack trace will help you pick a location in the program that's earlier in execution. But sometimes the actual bug is long gone and you need to know some things about what the program has done before you get to your part.

Other Allinea features

- There's a profiler, called map. I found it pretty intuitive. Also lots of online documentation.

- For very slow programs, you can even use the debugger as a substitute profiler.

- Run the program, click the pause button, and look at the stack window to see what it's doing. Repeat. Often code will be in some system utilities but look up the stack until you see something familiar

Other Allinea Features

- Breakpoints can be configured to stop every n'th time, or stop on a condition you specify.
- Watchpoints stop when a particular piece of memory you specify changes its contents
  - makes a program run slowly, but if you're looking for memory corruption, this is often the only way to find it.
  - Every machine instruction, the program stops, compares the value stored in memory, and if the same, continues.
- Sometimes you are looking for a very rare bug. Code executes many times before failing. In that case, a cout or logdebug statement can be your best friend. See *art*'s messagefacility mf.
- Try to make a minimal example of your bug before asking for help. Find an input file where the program fails on the first event, if possible. Speeding the test cycle will make finding the bug much quicker.

Finding Bugs

- Sometimes bugs only show up in batch programs.
- Run your job on a few events interactively before submitting many jobs
- Estimate output file size and compare against available disk.
- Watch memory usage: I use "top". vsize is the column to watch during program execution. Profiling tools can be useful here.
- dunegpvm machines have 12 GB of RAM and 2 GB of swap. More than is granted to you on a batch worker. Use more than 13 GB and your program will crash on an "out of memory" error. Use too much virtual memory on a batch worker (default=2GB), and the job gets held.

Batch job bugs

- /dune/data, /dune/data2, /pnfs/dune and your home area on /nashome/ are all *not* mounted on the batch workers.
- /dune/app is, however, but only on the Fermilab batch workers, not on general OSG computers. (CMS Tier-1 is "Offsite"). And only until January 2018, so don't start using it! And stop using it! (use CMVFS instead, and copy your own code in a tarball on dCache with ifdh cp and unwind it yourself on the batch worker).
- Try your job on
  gpgtest.fnal.gov
  which is set up "like" a batch worker. Though it has home directories mounted, so it's not perfect.
- New changes to the batch system (containers!) mean that gpgtest.fnal.gov is even less representative of what you get on a batch worker
- An interactive test may need to define _CONDOR_SCRATCH_DIR and PROCESS in order to work.

Batch Job Bugs

- Look at your logfiles! You will get a .out file and a .err file – look at both!
- jobsub_fetchlog –list-sandboxes will tell you which logfiles are stored on the jobsub logfile server
- jobsub_fetchlog -J <jobid> will give you a gzipped tarfile with your stdout and stderr files and the wrapper scripts.
- tar -zxf <logfiletarball.tgz> will unpack it. Best to do this in an empty directory.
- Only the first and last 5MB of stdout and stderr are stored.

Running over a Dataset with project.py

- Documentation is here:
  https://cdcvs.fnal.gov/redmine/projects/larbatch/wiki/User_guide
- It's linked on the "submitting grid jobs" how-to page:
  https://cdcvs.fnal.gov/redmine/projects/dune/wiki/Submitting_Jobs_at_Fermilab
- Also project.py –help
- Handy interface: projectgui.py xml_files
- project.py is good for running lar jobs – you specify a fcl file and a dataset, and things like njobs and memory and time limits, input and output files, and project.py will use its batch script to run the set.
- See example input xml files at
  http://dune-data.fnal.gov
- Those are for MC samples, but you can run over existing datasets using the <inputdef>...</inputdef> tag in the xml file

Continuous Integration Tests

- Automated tests help catch unintended behavior of code, configuration files, and auxiliary data files.
- CI tests are run on every commit (with a 15-minute cooling-off period in case many commits come all together).
- Commits to LArSoft run experiment CI tests. And due to ease of configuration, vice versa.
- Tests take some time to run.
- Check results at: http://dbweb5.fnal.gov:8080/LarCI/app/view_builds/index
- This link is availble on the larsoft wiki https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki

About CI Tests

- Two kinds of CI Tests
  - unit tests: "standalone" Tests of pieces of code outside of the framework. Quick to run, harder to construct
    - need to provide configuration and enough services to get the piece of code to run in a way that is useful to test
    - need a script/configuration file that defines the desired output
  - integration tests: These run an actual job.
    - easier to write – corresponds to workflows we run anyway.
    - example: generate and simulate an event. Count data products and sizes.
  - Discussions of these at 2014's LArSoft CI Workshop https://indico.fnal.gov/event/8571/

CI How-to

- Look in dunetpc/test for examples.
- Integration tests in the ci subdirectory – test standard fcl files
- Check the CMakeLists.txt files in the directories. Top-level CMakeLists.txt files make sure the subdirectories are added. The ones in the test directories have cet_test items in them.
- A unit test example:
  Look in larreco/test/RecoAlg/HitFinder/
  contains GausFitCache_test.cc and a CMakeLists.txt file that sets it up and runs it.

- See instructions on "How to Check out Everything" to look for tests in all LArSoft repositories.