GArSoft Notes

Machines

There are dunegpvm##.fnal.gov nodes, where ## is anything from 01 to 15. Nodes 01 through 10 are SLF6; 11-15 are SLF7. Build for SLF6 on dunebuild01, and for SLF7 on .

About the dang grid… well you have something that sort-of works. Assume NOTHING is mounted and all you can do is copy on & off $DUNEpnfs – and the pnfs disk does weird stuff. To copy off them from a condor node, you need ifdc cp. You can debug scripts that run on the condor nodes by running them in a script with pretty much nothing defined. Amit Bayshal gave you the following handy options to the jobsub_submit command, which can also be found with jobsub_submit –help:

```
--resource-provides=usage_model=OPPORTUNISTIC
--expected-lifetime 4h
--memory 500MB
```

It looks like 8 hours = 28800 sec is the default runtime, and you can revive jobs that go into hold at that point with jobsub_release; but in fact those jobs are probably hosed anyway in all likelihood. May as well jobsub_rm 'em and restart. This is a very unreliable system, it seems.

From trying to get jobsub to run in the MINERvA framework, I have the following notes: "/minerva/app/users/bellanto (which contains my cmtuser area) is visible as read-only to the gridjob; and one can spool stuff off to the /pnfs area after the job ends, using the –d <tag> <dir> option to the jobsub_submit command. That option creates a symbol $CONDOR_DIR_<tag> which you can use in the scripts that are executed to specify a local area that the job uses; and then, after the job exits, another machine copies files off that local area to /pnfs. The inverse operation, to get files into the Fermigrid jobs, seems to involve the –f <file> option."

Debugging: You've got allinia and an alias setup in GArSoftsetup.sh. Try e.g. DDTart -n 10 -o dbgen.root -c prodgenie.fcl and ignore the message about the old version. Also ignore the message about not finding the source file for *art*; set your breakpoints by source file name and line number. Finally, ignore messages to the effect that the source is newer than the

executable; because the install step moved the executable, the source is always newer than the (wrong) executable that `allinia` gets its date from. Now you might ask “Which version of the source file name?”. Tom says if the source file was moved to the build area it should be found already by `allinia`. Oh! Look! `valgrind` appears to be available on these nodes via `ups`!

Tom Junk found out how to run gdb on the batch worker to get a stack trace. His example is for running a readoutsim step:

Instead of just running the readoutsimjob step, do this:

```
setup gdb
gdb --args art -c readoutsimjob.fcl -o readoutsim_0025.root
genie_gen_0025.root <<EOF
r
bt
quit
EOF
```

To detach processes running interactively, use the screen command:

`screen`	spawn a sorta subprocess in this terminal.
`screen –list`	see what screens exist & their state.
`screen –d pid.tty.host`	detach screen from the terminal it is in.
`screen –r`	reattach the screen into this terminal.

About *art*

To understand the expansion of a `fcl` file, try `fhicl-expand`, which will use `$FHICL_FILE_PATH` to locate the file. Also, there is the handy `art --debug-config`. Help at `art --help` is reasonably useful. And! `config_dumper` will give you the `fcl` file contents of the *art* modules that created an *art* event data file such as `reco.root`, the output of the reconstruction *art* job.

In a `---.fcl` file, the term `@local::` means "In this file, after all the `#includes` have been expanded". So for example, if `prodgenie.fcl` contains a line

```
#include "Geometry.fcl"
```

and `Geometry.fcl` includes a parameter named `standard_geo`, as in

```
standard_geo:
{
      ForceUseFCLOnly:   false
      .
      .
      .
}
```

then when `prodgenie.fcl` says

```
services:
{
      Geometry:              @local::standard_geo
      .
      .
      .
}
```

that means that `prodgenie.fcl` has `ForceUseFCLOnly` defined for it.

Evidently, if you create a new `fcl` file in the `srcs/garsoft` area, and then build, that `fcl` file gets copied to the appropriate subdirectories in the local products and build areas. It is the one in the build area that gets taken by an *art* job by default if you don't have a `fcl` file of that name in the directory from which you launch *art*. You have `$FHiCL` defined to point to that build area.

`cetskelgen` is a device to automatically make *art* modules, which are also called plugins. In *art*, a "module" is a dynamic library containing classes somehow. They call services and mostly interact with events; they never invoke each other. They have methods such as `beginJob`, `endJob`, `beginRun`, `endSubrun` etc. There is a method to process an event, called `analyze(art::Event const& event)`. There are 5 types: analyzer, producer and filter modules, deriving from `art/Framework/Core/EDAnalyzer`, `EDProducer` and `EDFilter`; and then there are source and output modules.

About *art* data products: the `TTree Events` in a root file containing *art* event data has `TBranches` with names of the form `<data type>_<fhicl file module label>__<process name>`. (Unless there are multiple instances of the same process). For an `art::Assn`, the `<data type>` is the names of the two data products, concatenated, even if the association carries a 3rd data element with it, followed by a concatenated `voidart::Assns`.

These are large files – about 1M per event at this point – and they get larger as generation goes through the steps of GENIE/Helper, readout simulation and reconstruction as all these tasks are *art* producer modules; they add *art* data products to the `::art::Event` but cannot remove anything. But you cannot `rm` the intermediate files and just keep the reco job output ☹.

When creating a class that will be an *art* data product, you need to modify the `classes.h` to include the new header. Then in the `classes_def.xml` file, add 3 lines such as

```
<class name="std::vector < std::pair<float,float> > "/>
<class name="gar::rec::TrackIoniz"  ClassVersion="10" >
</class>
```

The 1st line is there because I made a vector of pairs as a data member of this class, so ROOT has to make a dictionary entry for that.

And then, lower down, add something like

```
<class name="std::vector<gar::rec::TrackIoniz >"    />
```

and

```
<class name="art::Wrapper< std::vector<gar::rec::TrackIoniz > >"  />
```

Finally, a set of six lines for any associations that will be needed. E.g.

```
<class name="std::pair< art::Ptr<gar::rec::TrackIoniz>,
art::Ptr<gar::rec::Track> >" />
```

```
<class name="std::pair< art::Ptr<gar::rec::Track>,
art::Ptr<gar::rec::TrackIoniz>  >" />
<class name="art::Assns<gar::rec::TrackIoniz, gar::rec::Track>" />
<class name="art::Assns<gar::rec::Track, gar::rec::TrackIoniz>" />
<class name="art::Wrapper<art::Assns<gar::rec::TrackIoniz,
gar::rec::Track> >" />
<class name="art::Wrapper<art::Assns<gar::rec::Track,
gar::rec::TrackIoniz> >" />
```

But wait. There is more. Build with `mrb i -j4`, or some other `j`. Watch the build fail, saying

```
INFO: adding version info for class 'gar::rec::TrackIoniz':<version
ClassVersion="10" checksum="2108826658"/>
WARNING: classes_def.xml files have been updated: rebuild
dictionaries.
make[2]: ***
[garsoft/ReconstructionDataProducts/ReconstructionDataProducts_dict_
checked] Error 2
make[1]: ***
[garsoft/ReconstructionDataProducts/CMakeFiles/checkClassVersion_Rec
onstructionDataProducts.dir/all] Error 2
make: *** [all] Error 2
```

Then build again. What will happen on the 2nd build is that a field of the form `<version ClassVersion="10" checksum="2108826658"/>` will be put into the `<class name="gar::rec::TrackIoniz" ClassVersion="10" >`, appearing between that `<class …>` and the following `</class>` delimiters.

To prevent a field named I`Dnumber` that exists in a data products class from appearing on the output data product, put `<field name="IDnumber" transient="true"/>` at the same level as the `<version … />` xml nodes.

Here is an example of what has to go into `classes_def.xml` if you want to build a 3-way association; in this case between a `gar::rec::Vertex` and a `gar::rec::Track`, with a `gar::rec::TrackEnd` as the extra data. `TrackEnd` is a typedef to `int`, but the dictionary-maker does not know that.

```
<class name="std::pair< art::Ptr<gar::rec::Vertex>,
                        art::Ptr<gar::rec::Track> >" />

<class name="std::pair< art::Ptr<gar::rec::Track>,
                        art::Ptr<gar::rec::Vertex> >" />

<class name="art::Assns<gar::rec::Vertex, gar::rec::Track>"   />

<class name="art::Assns<gar::rec::Track,  gar::rec::Vertex>"  />
```

```
<class name="art::Wrapper<art::Assns<gar::rec::Vertex,
                                     gar::rec::Track> >" />

<class name="art::Wrapper<art::Assns<gar::rec::Track,
                                     gar::rec::Vertex> >" />

<class name="art::Assns<gar::rec::Vertex, gar::rec::Track,
                                     gar::rec::TrackEnd>" />

<class name="art::Assns<gar::rec::Track,  gar::rec::Vertex,
                                     gar::rec::TrackEnd>" />

<class name="art::Wrapper<art::Assns<gar::rec::Vertex,
                    gar::rec::Track,  gar::rec::TrackEnd> >" />

<class name="art::Wrapper<art::Assns<gar::rec::Track,
                    gar::rec::Vertex, gar::rec::TrackEnd> >" />



<class name="std::pair< art::Ptr<gar::rec::Vertex>,
art::Ptr<gar::rec::Track> >"   />

<class name="std::pair< art::Ptr<gar::rec::Track>,
art::Ptr<gar::rec::Vertex> >"  />

<class name="art::Assns<gar::rec::Vertex, gar::rec::Track>"    />

<class name="art::Assns<gar::rec::Track,  gar::rec::Vertex>"   />

<class name="art::Wrapper<art::Assns<gar::rec::Vertex,
gar::rec::Track> >"      />

<class name="art::Wrapper<art::Assns<gar::rec::Track,
gar::rec::Vertex> >"       />

<class name="art::Assns<gar::rec::Vertex, gar::rec::Track,
bool>"              />

<class name="art::Assns<gar::rec::Track,  gar::rec::Vertex,
bool>"              />

<class name="art::Wrapper<art::Assns<gar::rec::Vertex,
gar::rec::Track,  bool> >"  />

<class name="art::Wrapper<art::Assns<gar::rec::Track,
gar::rec::Vertex, bool> >"  />

<class name="art::Wrapper<art::Assns<gar::rec::Track,
gar::rec::Vertex, bool> >"  />
```

The build system

When doing a fresh install, one has to know that all the dependent products exist in the correct COMPILER:BUILDTYPE combo. For example, when I created an area in Nov 2018, I could use `e15:debug`; but by Jan 2019, we had moved to *art* v3_00_00 which only exists in `e17:debug`.

Some useful environmental names:

`LD_LIBRARY_PATH`: the load library path; where *art* looks for modules.
`FHICL_FILE_PATH`: where *art* looks for `fcl` files.
`PRODUCTS`: where ups searches for products, including your local ones.
`FW_SEARCH_PATH`: for finding frameworks. Whatever they are.
`NUTOOLS_DIR`: There's probably something useful in there.
`MRB_PROJECT`, `MRB_TOP`, etc: From the install step.

How to pull and build a new version of GArSoft:

```
cd …/srcs/garsoft
git pull
cd $MRB_BUILDDIR
mrb i –j4
```

If you are on `dunebuild01.fnal.gov`, you could type `–j16` because that machine has 16 cores. But `dunegpvm` machines only have 4. Although you might think that everything is being built based on what you see on the screen, in fact `mrb` uses `CMake` uses `make`, and that is not the case.

The `mrb` command actually does the build. The `mrbsetenv` command, which you have in `GArSoftsetup.sh`, sets up this area to build a local, you-are-the-only-customer ups product called `garsoft`. The `mrbslp` command then does a ups setup command for this product.

To "make clean", so to speak, use `mrb z`. This will will delete the contents of `$MRB_BUILDDIR`, and you may also `mrb zd` to delete the localProducts distribution too, which is necessary so you won't set up the old one when you log in and build a new one."

What happens under the hood here is that your local copy of the git repository is in `…/srcs/garsoft` and `mrb` builds in a temporary directory of sorts, which is the `…/build_slf6.x86_64` directory. If that is successful, it installs by

copying the right bits and pieces to the area which serves as the definition of your own, personal ups products, …/localProducts_garsoft_develop_e17_debug.

Perforce, the files you would want to edit are your copy of the git repository, the …/srcs/garsoft area. If you need a source file for the debugger, you could take it from the localProducts area, if it is there; or from the …/srcs/garsoft area because it would be the same as long as the build worked. Tom says that allinia should be able to fine the source from the build area, so if you get an error message saying "no source" then there is no point in looking for the source code in the build area.

Perforce again, in …/srcs/garsoft there is a directory ups, which contains information about the ups information for this you-are-the-only-customer product. For example, …/srcs/garsoft/ups/product_deps tells me that building GArSoft depends on v2.12.10 of genie_xsec and genie_phyopt.

CMake: In our CMakeLists.txt files, we use the art_make macro. The LIBRARY_NAME is the name of the *pair* of libraries that you are making. The MODULE_LIBRARIES are what you link to in order to make the library of *art* modules. The LIB_LIBRARIES are what you link to in order to make the library of all the other stuff in that directory. The MODULE_LIBRARIES should refer to the very same library name, meaning that the MODULE_LIBRARIES link to the LIB_LIBRARIES in the same area. There is only one LIBRARY_NAME field, at most, in each CMakeLists.txt file, so there must be some convention about the names such that the 2 are related and there is some knowable name if there is not a LIBRARY_NAME field.

If there is no module, tool, service or whatever in a library area, then you do not get told that there is no --.so file created. Instead, if some other library has a declared dependency on that area, it tells you that it could not find that library.

Baffling git shit

· To define which repository I am accessing,

```
cd $GArSRC
git remote set-url origin ssh://p-
garsoft@cdcvs.fnal.gov/cvs/projects/garsoft-garsoft
```

· To be prudent!

`git diff file`
– to compare your staging area (red) against your working area (green) using `more` or `less`. With any luck, your staging area matches your repository. Might match the remote repository, might not.

`git status` – to compare staging area against local area, working area against staging area, and to find out what files are not being `git`ted.

`git commit -a --dry-run`

`git fetch; git merge`
– this is the same as `git pull`

When doing `diff` in redmine, it seems that the new one is the version marked in green.

· To remove a file not just from your local area but also from the index file and hence, after `commit`, from the repository,

`git rm file`

To push into the repository, first do a `git status` to figure out what you need to do, then `git add` to the staging area, then `git commit` to the local repository, then `git push` to the project repository. I do this from `$GArSRC` and there are some directories that it won't work from.

```
cd $GArSRC
git status
git add <filenames>
git commit -m "Add a comment"
git pull
git push <remote> <branch>
```

Where I don't actually need to do `<remote> <branch>` - presumably because my `develop` branch is a "tracking branch" and so knows that I am talking to `origin/develop`. The `-m` comment needs a double quote. There is an error message about email not being sent, which is just fine. One can skip the `git add` by using `-a` in the `git command` step. The `git pull` is there on Tom's

advice, to be super-certain that nobody has messed with the repository since you pulled this version; that step will throw an error of some sort if that has happened. It might say:

```
CONFLICT (content): Merge conflict in < filename >
Automatic merge failed; fix conflicts and then commit the result.
```

In which case, go into that file and look for <<<. The 1st part, labeled `HEAD` is your code, and the 2nd part, between the === and the hash code is what's in the repository. Fix it up and then redo the `git add` and `git commit` steps. Then do the `git push`.

If you have moved a file from one location to another in the directory tree, you need to (perhaps) `git add` the new directory, `git add` the file in the new location, and `git rm` the file from the old location.

At the end of it all, look at the 'revisions' section of the repository from the GArSoft Redmine page. It takes a while to be updated; 30 minutes in one case from my `git push` to it being fully updated.

The error message

```
fatal: Not a git repository (or any parent up to mount point
/dune)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM
not set).
```

is due to not being in `$GArSRC` when you type your git command.

Coordinates, conventions:

Right handed coordinate system, z is the direction of the beam, y is straight up. The magnetic field is in the x direction: $\vec{B} = B_0 \hat{x}$.

being an independent variable. The point (x_0, y_0, z_0) is on the helix. An arbitrary point on the helix is given by

$$x = x_0 + r \tan\lambda(\phi - \phi_0) \quad (1)$$
$$y = y_c - r\cos\phi \quad (2)$$

$$z = z_c + r\sin\phi \quad (3)$$

The location of the center of the circle (y_c, z_c) is given by

$$y_c = y_0 + r\cos\phi_0 \quad (4)$$
$$z_c = z_0 - r\sin\phi_0 \quad (5)$$

The slope s of a track is defined to be

$$s = \frac{\sqrt{dy^2 + dz^2}}{dx} = \cot\lambda \quad (6)$$

Early versions of `GarSoft` used s instead of λ as the fifth track parameters, but this was problematic as s diverges for particles traveling perpendicular to the E and B fields (and along the neutrino beam direction!), and changed sign, creating a discontinuity in the mapping between track parameters and track angles where a lot of physically important tracks are. The λ parameter provides a continous, bounded mapping for these tracks and does not diverge for tracks parallel to the fields.

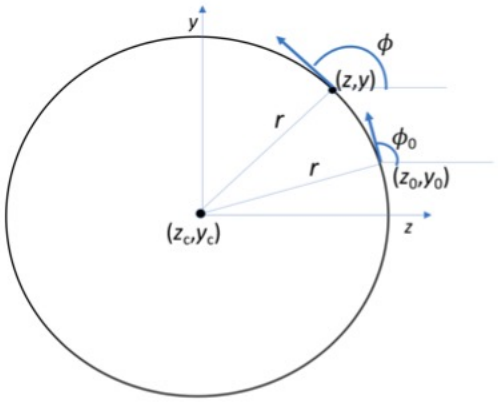

FIG. 1: Track parameter definitions in the (z, y) plane.

The track parameters are said to be: (y_0, z_0, ρ, φ_0, λ; x) where the order reflects their appearance in the code; (x, y_0, z_0) are a point on the helix (not a center as determined at drift position x) and are in cm; $\rho = 1/r$ is curvature of the track at that point in cm^{-1}, φ_0 is the parameterizing angle at (x, y_0, z_0) as seen in the transverse plane w.r.t. the z axis, oriented to be 0 for a track in the z direction and

$+\pi/2$ for a track in the y direction in the transverse plane, and λ is the angle of the track with respect to the transverse plane (equivalently, the angle with respect to the HV membrane). The sign of ρ is the same as the sign of r, and it does *not* give the charge; that also needs some concept of $d\varphi/dt$, i.e. which end of the track the particle started at and which end it ended at. The x parameter is a separate variable not in the track parameters, i.e. the full set at the beginning of the track is `fTrackParametersBegin[5]`, `fXBeg`. The forward fit consumes `TPCClusters` in order of increasing x; the backwards fit, in order of decreasing x.

From the GENIE manual, it uses "$\hbar = c = 1$, so almost every simulated quantity is expressed in powers of [GeV]. Exceptions are the event vertex in the detector coordinate system (in SI units) and particle positions in the hit nucleus coordinate system (in fm)." In GENIE, there is a variable for the position of the initial nucleus in the lab frame and then subsequent particle/vertex locations are relative to this location. However, when GENIE's output, `GHepRecord`, is converted into an *art* structure, the positions in the lab frame are needed. In cm, I guess.

The event display:

About Tom's event display: `art -c Toms_evd.fcl <reco.root file>`. `CNTL SHIFT +` and `CNTL -` to get bigger and smaller. Rodent will rotate, no pan yet. Reco colors are assigned randomly and reco tracks have a little arrow on their end to indicate direction of some sort. For MC Truth tracks, light dashed blue is a ν, solid purple is a proton, red for an electron, blue for a muon, magenta for a pion (?) and little dots for a neutron or photon. Yellow dots are vertices. Green octohedrals are entire ECAL clusters.

What is the geometry, anyway?

The geometry, as of mid-2019, is in …/srcs/garsoft/Geometry/gdml/ND_Strawman_Concept_v09.gdml. This file has 80 Cu-scintillator sandwiches. The Cu is 2mm thick, and the scintillator is 5mm thick; from the C/H ratio it looks like polystyrene. The first 8 layers also have 1mm thick layers of FR4 epoxy-fiberglass. I think these are for the pad readout regions, and the rest of the device is strip readout. Some stats:

> 2mm of Cu is 0.1393 of an X_0; 5mm of polystyrene are $0.0121$of an X_0. Then X_0 of the ECAL is 6.61cm. For λ_{HAD},, the numbers are $0.0132\ \lambda$, $0.0065\ \lambda$ and 51.17cm. Total thickness is $80(0.7\text{cm}) = 56\text{cm}$, so we have $8.5\ X_0$ and $1.1\lambda_{HAD}$.
>
> MIP loss in passing through ECAL: $0.2\text{cm}(12.57\ \text{MeV}/\text{cm}) + 0.5\text{cm}(2.052\ \text{MeV}/\text{cm}) = 3.54\ \text{MeV}$ per layer; and for 80 layers, $283\ \text{MeV}$.
>
> Density: Cu is $8.96\ \text{g}/\text{cm}^3$; polystyrene is $1.06\ \text{g}/\text{cm}^3$. Areal density is thus $80\ (\ 0.2\text{cm}(8.96\text{g}/\text{cm}^3) + 0.5\text{cm}(1.06\text{g}/\text{cm}^3)\ ) = 80\ (1.792\ \text{g}/\text{cm}^2 + 0.530\ \text{g}/\text{cm}^2) = 185.8\ \text{g}/\text{cm}^2$. That is by weight 772m^o Cu, and by volume 286m^o Cu.
>
> Volume: The outer octagon of the barrel has apothem $a = 336.95\ \text{cm}$, corresponding to a side $s = 279.14\ \text{cm}$, and a length of $\ell = 2(385.0\ \text{cm}) = 770\ \text{cm}$. The inner octagon of the barrel has $a = 278.45\ \text{cm}$, $s = 230.68\ \text{cm}$. The area of an octagon is $2(1 + \sqrt{2})\ s^2$, so the volume of the barrel is $2(1 + \sqrt{2})\ (\ (279.14\ \text{cm})^2 - (230.68\ \text{cm})^2)\ (770\text{cm}) = 91.85\ \text{x}10^6\ \text{cm}^3$. The endcaps' thickness is $(430\text{cm} - 385\text{cm}) = 45\text{cm}$; so each has a volume of $2(1 + \sqrt{2})\ (279.14\ \text{cm})^2\ (45\text{cm}) = 16.93\ \text{x}10^6\ \text{cm}^3$. Total volume $125.7\ \text{x}10^6\ \text{cm}^3$.
>
> Mass: The mass of the Cu is $(286\text{m}^o)(125.7\ \text{x}10^6\ \text{cm}^3)(8.96\ \text{g}/\text{cm}^3) = 322.17\ \text{x}10^6\ \text{g}$ which is $322\ \text{metric tons}$. Then for the scintillator, $(\ 1 - 286\text{m}^o)(125.7\ \text{x}10^6\ \text{cm}^3)(1.06\ \text{g}/\text{cm}^3) = 95\ \text{metric tons}$, for a total of $417\ \text{metric tons}$. Eldwan used his GEANT model and got $288\ \text{metric tons}$ in the barrel and $134\ \text{metric tons}$ in the endcap, for a total of $422\ \text{metric tons}$.
>
> Figure Cu, polystyrene and Ar have the same cross sections per nucleon (equivalently, per ton) for the following: $422\ \text{metric tons}$ times [1.50, 0.50] $\text{x}10^6$ [CC, NC] events per standard year is [633, 211] $\text{x}10^6$ / $(56\%\ 31.6\ \text{x}10^6) =$ [35.8, 11.9] [CC, NC] events per spill.

About GENIE and its little Helper

GENIEGen_module.cc is a thin layer over the nutools class evgb::GENIEHelper. It is an *art* producer module, i.e. its interface conforms to the demands of art/Framework/Core/EDProducer.h. GENIEHelper is a not-so-thin layer over GENIE itself which calls the generator event by event in GENIEHelper::Sample(…). Also in that method, GENIEHelper calls PackMCTruth(…) and PackGTruth(…) to unpack the genie::EventRecord into simb::MCTruth and simb::GTruth objects that live in GENIEGen::produce(::art::Event& evt). Then, in some way still obscure to me, GENIEGen::produce(::art::Event& evt) will put vectors of this info into the large root file that it produces. The small file that it produces contains histograms made in GENIEGen::beginJob() and filled in GENIEGen::produce(::art::Event& evt).

UserPhysicsOptions.xml is in a ups product called genie_xsec; there is also a ups product called genie_phyopt as well. These are not in $LD_LIBRARY_PATH,probably because they contain data of various sorts needed by GENIE, but not any executable libraries. Specifically, UserPhysicsOptions.xml exists in 2 locations from /cvmfs/larsoft…genie_xsec/v2_12_10/NULL. One is DefaultPlusMECWithNC/data and the other is DefaultPlusValenciaMEC/data. The selection between these two directories to get UserPhysicsOptions.xml appears to be made with the environmental variable $GXMLPATH although the $GENIEXSECPATH and $GENIE/config (for v2.12.10) seem to be relevant somehow.

In $GENIE/config are the xml files that control what GENIE does. They are documented in Gabe & Tomasz' MINERvA 101 talks in 2014 and 2015. If, in Tom's GENIEGen.fcl, you change the parameter EventGeneratorList, this changes what process GENIE looks up in $GENIE/config/EventGeneratorListAssembler.xml – good options are Default (for a reasonable mix of all interactions), CCCOH for charged-current coherent $\pi^{\pm}$ production, and NCCOH for neutral-current coherent $\pi^{?}$ production.

Evidently, flux files are copied by GENIEHelper to /var/tmp/ifdh_6200_### where ### is the process pid. Now, /var/tmp/ has a garbage collection process that collects items that are not garbage, because if the system runs out of space there it stops the system. It seems that a cet::exception& gets thrown - I'm not sure where because I don't think root knows

about `cet::exception` - and gets caught in `art::Worker::runWorker`, at line 593. (v3.0.0 of *art*). In the 2015 art/LArSoft course materials, I read that *art* will attempt a graceful shutdown. (https://indico.fnal.gov/event/9928/session/2/material/3/2.pdf, pg 9) In principle. In practice, the output files are not closed correctly and the effort in the job is lost.

This is a problem when running GENIE to generate even a modest number of CCCOH events; I am seeing that this particular type of event is produced at the rate of one every 12, 13 seconds on `dunegpvm07` for example. However, I try changing the field `FluxCopyMethod` set to `DIRECT` in `GENIECCOH.fcl` – and this does seem to do the trick.