# Outline for the *art* Workbook

Rob Kutschke
June 2, 2016

Abstract:

This document is the working outline for the the Workbook section of the *art* documentation suite. The Workbbook is Part II of the current large document. The chapter numbers in this outline correspond to the exercise numbers in the Workbook. At this writing, Workbook has exercises 1 through 5 in place.

Items written in red in chapters 1 through 5 are things that need to be added to chapters that have already been released.

# Contents

# 1 Run Pre-built art Modules

1. when to use the site-specific setup procedure
2. a little bit about the *art* run-time environment
3. how to set up the toyExperiment UPS product
4. copy some example .fcl files from the toyExperiment UPS product and run them
5. run an *art* job that prints out the event ID's of all events in a file.
6. learn about 3 part event id
7. how to control the number of events to process
8. how to select different input files ( many ways )
9. how to start at an event that is not the first event in the file
10. how to start at a run that is not the first run in the file
11. how to start at a subrun that is not the first subrun in the file
12. how to concatenate input files
13. how to write an output file
14. some basics about the grammar and structure of a FHiCL file
15. how to use art-fcl mode in emacs

# 2 Build and Run Your First Module

- what is the *art* development environment
- how to establish the *art* development environment
- how to checkout the Workbook exercises from the `git` source code management system
- how to use the **cetbuildtools** build system to build the code for the Workbook exercises
- where does the compiler look for included header files
- how can you find the source for the header files that your code included
- what is a *link list* and where is it specified
- LD_LIBRARY_PATH
- where does the build system find files in the link list
- what the `art::Event` is and how to access it
- what the `art::EventID` is and how to access it
- what makes a class an *art module*
- where the build system puts the `.so` files that it makes
- how does *art* know where to look for your `.so` file
- Some experiments use the 3 file mantra for modules. Mention it.
- The namespace tex.

# 3 Begin/End Job/Run/SubRun

1. learn about the member functions that can be used at the other state transitions within the event loop:
   (a) `beginJob()`
   (b) `beginRun( art::Run const&)`
   (c) `beginRun( art::SubRun const&)`
   (d) `endJob()`
   (e) `endRun( art::Run const&)`
   (f) `endRun( art::SubRun const&)`
   (g) Mention the open/close input/outfiles member functions and point to exercises with them.
2. you only need to provide these member functions when you have something useful to do at those times; if you do not have anything useful to do, you need not provide them. As you gain experience you will learn what is useful to do when.
3. learn about the classes:
   (a) `art::RunID`
   (b) `art::Run`
   (c) `art::SubRunID`
   (d) `art::SubRun`

# 4 A First Look At Parameter Sets

1. learn the basics of how to provide run-time configuration to a module via the parameter set in the .fcl file
   (a) read parameters from the parameter set
   (b) require that a particular parameter be present in a parameter set
   (c) specify that a parameter has a default value that will be used if the parameter is not present in a parameter set
   (d) print a parameter set
2. First encounter with templates: provide minimal intro. Few of you will write templates but you will use them.
3. First encounter exception handling system (module code does not throw but the parameter set system can ).
4. Use data members to propagate parameters from ctor to other member functions.
5. Can hold a copy by value.
6. Do not discuss includes, prologs or @local.
7. Do not discuss overriding parameters or the dot separated notation.
8. Some gotchas.
9. Your experiment may have policies about the use of optional parameters - learn them.
10. Numerical types and representations.
11. References to advanced info about parameter sets

# 5 Making Multiple Instances of a Module

1. for this exercise, the module reads an identifying number from the parameter set and prints it for each event.
2. use this module to illustrate what it means to make multiple instances of one module within an art job; given each instance a distinct identifying number.
3. Reinforce the idea that for analyzer modules, art may run them in any order.
4. Give examples of situations in which this might be useful.
5. The name "module label" is used to mean two subtly different things: an identfier of a fhicl table and as the set an indentifier of a module that has been configured using that fhicl table as a parameter set.
6. Command line option: –tracer

# 6 Read GenParticleCollection

This chapter plus the next three belong as a unit. This material is broken into four chapters so that each is a manageable size.

Need to update to a newer version of toyExperiment that has some background tracks. Also need to update the intro to say that the toy experiment has N background particles per event; and update some of the figures in the intro.

1. Overview: get a data product from the event and, once per event, print the size of the data product. Do it two ways, both getByLable and getValidHandle.
2. Tell reader to review the toy experiment in the intro.
3. Name of the class that the describes a generated particle is GenParticle.cc . Find the header file. Superficially describe the info available. Say that chapter 8 will get into the details of the class.
4. Name of the data product class is GenParticleCollection. Just a std:vector<GenParticle> Why didn't we just call it that?
5. Introduce the 4 part naming convention for data products.
6. State that we want GenParticles_evtgen__exampleInput; reminder no underscores in module labels, instance names and process names. The friendly name code removes underscores from the data type.
7. art::InputTag
8. art::Handle
9. art::Event::getByLabel
10. art::Event::getValidHandle
11. isValid method of a handle. Tell people not to use it unless there is a good reason to.
12. Use of auto
13. Big red print: caution about unnecessary copies and remembering the &.

14. If you plan to use a handle a lot, dereference it once.
15. Strongly recommend that you get the input tag from the parameter set and do NOT provide a default. Say why.
16. Explain why you need to extend the link list and how you know what you need to do.

# 7 Making Histograms

1. Overview: shows how to book and fill a histogram using root. Use the number of GenParticles in an event as an example.
2. This example shows how to create and view histograms; the packed used for this is called ROOT. ROOT is also used for peristency. ROOT has many other capabilities. References to ROOT documentation.
3. Caution that in our environment we recommend some practices that differ from those shown in the ROOT examples.
4. Remember to define that the word "book" means "create" or "instantiate"; usually binning is specified at booking time but there is a dynamic binning option.
5. art provides a tool that helps out with some of the bookkeeping needed to make ROOT work. This tool is called the TFileService.
6. This is an example of an art::Service. Will talk more about services in another chapter ( add reference to it).
7. art::TFileService, art::ServiceHandle, TH1D
8. Describe what the `tfs->make<T>` template does; need one version for the ROOT pros and one for beginners.
9. Setting the file name: in .fcl file and command line; command line wins.
10. Pattern:
    (a) TH1D* as a data member
    (b) initialize to nullptr in initializer list ( comment on nullptr being part of C++11 standard; prior to C++11 many people defined their own version of this - if you have it, get rid of it).
    (c) Book in any of beginJob, beginRun, beginSubRun
    (d) Fill in analyze
11. Do not discuss making ROOT subdirectories - save that for later. But reference that chapter.

12. Open the root output file and browse it.
13. Save the TCanvas as pdf and as .gif
14. View the root file using a cint script.
15. Print the pdf file.
16. Put my browse function somewhere and tell people about it.
17. Checklist of ROOT hygeine.

# 8 Looping over the GenParticleCollection

1. Longer discussion of the information in the information in a GenParticle but ignore art::Ptr's for now. Refer to the CLHEP documenation. Need a draft the CLHEP appendix before we release this.
2. Put a pmin parameter in the pset.
3. Loop over a GenParticle collection.
4. Make plots of some properties of a GenParticle
5. Exercise some member functions of CLHEP::Hep3Vector
6. Show several methods of writing the loop. This is really C++ basics, not art proper.
   (a) .at(i)
   (b) [i]
   (c) ::const_iterator
   (d) ranged for with explcit type
   (e) ranged for with auto
7. Recommend ranged for because it has the fewest ways to make an error; art-workbook will use this pattern whenever it can.
8. Remember the const& after auto.
9. Bonus points: don't use $i < n$ for loop control; don't use $i++$ when $++i$ will do.
10. Introduce operator << for GenParticle; add these to your own data classes.
11. Also introduces the particle type enum.
12. cint script to make a multipage pdf file.

13. Write a fcl job to make two instances of the module, with two different momentum cuts. View the ROOT file and see the two directories. Verify that the momentum spectra have different pmin cuts.
14. This is the place to talk about the 3 line mantra for getting data products.
15. Suggested exercises:
    (a) Add histograms of the other attributes of the GenParticle

# 9 The Particle Data Table Service

1. Overview: some more details about services. Use the PDT service as an example. Talk about services from the users' point of view, not from the service writers' point of view. Reference the example that shows how to write a service.
2. Emphasize that this is a toy particle data table, stripped of functionality so that the big picture is not lost in the details.
3. Loop over GenParticle collection and print out the name of the species of each particle, not just the PDG id. Limit printout to first 20 events.
4. Make histograms of charge of each particle and the total charge of the generated event.
5. Maybe: count number of each type of gen particle and print out a table at the end. Use the pdt to print the name of the particle in the table. This introduces std::map. Put this into a separate class to illustrate the additional library.
6. services.user "namespace"-ish. Warn that it is not robust.
7. art::ServiceHandle
8. tex::PDT, tex::ParticleInfo
9. Need to add the PDT_service.so to the link list.
10. Comment on why we chose to put the PDT in a service: don't want singletons because of lack of control over lifetime and lack of control over run time configuration.
11. A service Handle is valid from module c'tor onwards but its contents may not be — you just gotta know.
12. PDT is defined at all times that any module is being called: c'tor, beginJob, beginJob ....

# 10 art::Ptr

The GenParticles use art::Ptr to express the parent/child relationship. First generation particles have no parent. Use this to introduce art::Ptr.

1. Introduce art::Ptr.
   (a) Describe the problem with persisting pointers.
   (b) art::Ptr acts as a persistable pointer.
   (c) under the covers it contains an art::ProductID and an offset
   (d) Can only point to a second tier object and can only do it if the the container has a random access iterator: std::vector, cet::map_vector not sure about std::map<integral_type,T>?
2. Find the phi meson in the GenParticleCollection; find its daughters.
3. Compute the invariant mass of the two daughters and plot it.
4. Verify that Ptr to parent for first generation particles is null; is isNull and isNonNull.
5. Distinguish the many ideas of validity/invalidity.
   (a) offset is the reserved value.
   (b) ProductID does not exist in the registry.
   (c) ProductID exists but the data product is not in memory and not in the input file.
   (d) ProductID exists and the product is in the input file but not yet in memory.
   (e) ProductID exists and the product is in memory but the index does not exist.
   (f) ProductID exists and the product is in memory and the index does exist.

# 11 The Geometry Service and Conditions Service

1. The reason for doing this now is that these are needed for the next exercises.
2. Nothing new about services. This is just a tour of the tracker geometry and the tracker efficiency.
3. tex::Geometry, tex::Tracker, tex::TrackerComponent, tex::Shell
4. Geometry is only valid if a run is in memory - so you cannot use it at c'tor or beginJob time.
5. print the geometry information
6. make some histograms of the geometry info
7. Change some geometry numbers and watch the histograms change.

# 12 Instance Names of Data Products

In the input files, the Intersections object come in two instances, one for each detector.

1. Show how to get a data product that has an instance name.
2. Refer to future examples for getMany and selector functions; don't do them here.
3. Advise people NOT to use getMany or selectors as a substitute for "getByType", which we removed from the API.

# 13 InRun DataProducts

The DetectorSimulation module also creates a Run summary object that describes the results of the simulation.

1. Get the MCRunSummary data product from the event.
2. Histogram the number of particles with 5 or more intersections.
3. No example provided for InSubRun but just follow the pattern.
4. Show that you can also get the Run and SubRun objects from the event; and the Run from the SubRun.
5. This data product is not a collection type. That is allowed everywhere.

# 14 Provenance

1. Get Provenance pointer from the handle.
2. Get full product name, and names of each part
3. Friendly and non-friendly type name
4. Get parameter set(s) in the creation chain: GenParticles are in the chain for Intersections.

# 15 Listing the Data Products in a File

1. Tell user to re-read story of the toyExperiment from the intro.
2. 4-part name of a data product.
3. How to list what data products are in a file
4. No code, just a .fcl file this time.
5. Maybe: Add a 5th example input file which is the same as the 4th one but done in two steps so that we can illustrate ProductID's better. Maybe defer this until later?

# 16 Producer Module

Need to extend toyExperiment/RecoDataProducts to have a class to represent a reconstructed phi meson candidate, tex::Candidate has art::Ptr to FittedHelixData Collection, plus mass hypothesis, plus mass and covariance. Both read facade and write facade this time.

1. Loop over all pairs of FittedHelixData.
2. Only consider oppositely charged pairs
3. Compute the invariant mass; use RecoTrk.
4. Add candidates that pass some loose cuts to the data product?
5. Read it back and make histograms from it; compare to histograms in the creator.
6. This module will show how to create Ptrs to the fitted helix data.
7. This introduces trigger paths.
8. Write a companion analyzer module to make some plots; run it both within the writing job and in a separate readback job.
9. Use the filedumper to see that it really is there.

# 17 Filter Module

1. Loop over data product created by the example producer ( tex::Candidate ). Select events that have at least one tex::Candidate that passes some tighter cuts than the producer cuts.
2. We could have written the producer as a filter but we recommend that you keep the two ideas separate so that you can write different workflows.
3. in a trigger path you can do: "!modlabel" will invert the sense of the filter "-modlabel" will run the module and acrue side effects ( like histograms) but art will ignore the return value of the filter.

# 18 Configuring Output Modules

1. Select events from trigger paths completion status. Mention that you can do boolean logic on multiple paths.
2. Multiple output files in one job.
3. Select which data products go into which file.
4. Use filedumperoutput to verify what is in each file.
5. mention fast cloning; superficial description; when might you want to enable/disable it? document the default and how to change it.
6. Show the grammar to select events that threw exceptions or otherwise failed.

# 19 Creating Your Own Data Products

1. Not sure yet what to use as an example; maybe the right answer is to make this orthogonal to the rest of the toyExperiment.
2. Create a data product library with just this new data product.
3. You should learn your experiment's policy on whether they want to have many libraries with few data products each or if they want to bundle related data products into a smaller number of libraries.
4. classes.h, classes_def.xml
5. Your data product class must be either movable or swappable.
6. art::Wrapper - this holds the TObjectness.
7. Structure of an art event-data file.

# 20 Module Name Collisions

1. A second HelloWorld module that prints something that is different enough from the HelloWorld module in exericse 2.
2. Need to delay making the .so until we are at this exercise or else the first module exercise will fail.

# 21 More FCL Concepts

1. This will be a standalone module with its's own toy .fcl file. I initially thought to use the full MC Chain but that is too much to swallow at once.
2. Explain PROLOG, @local, scoping rules, dot notation
3. When they arrive, @table and @sequence (and @echo?)
4. true, false, infty, +infty, -infty, @nil
5. Check Mu2e tutorial for additional details needed here.
6. To get fully resolved fcl file. ART_DEBUG_CONFIG=1; art ....
7. Remember –art-debug file.fcl and –config-out file
8. Refer to but do not explain how to specialize get_if_present¡T¿
9. config_dumper
10. Evangelize the future coming of ParameterSet validation.

# 22 art::Assns Connecting Hits to Intersections

1. The model is that a tex::Intersection is an MC truth object but a TrkHit is reco object - ie it could come from the real experiment so it must not contain any MCTruth info
2. Because of inefficiency there are Intersections with no corresponding TrkHit.
3. When we run on actual experimental data we don't want to load any MC libraries. Therefore we follow the pattern that we segregate MC classes into their own directories. Therefore Classes in RecoDataProducts must not know about classes in MCDataProducts. Classes in MCDataProducts may know about classes in RecoDataProducts. This pattern is common in many experiments.
4. Simulations/HitMaker_module.cc:
   (a) Has a model of inefficiency; look at run-time config for this but not the code.
   (b) produces TrkHitCollection - a single collection for both tracker components - just because.
   (c) produces TrkHitMatch - connection between Hits and Intersections
5. The name "God's block" - does anyone still use it?
6. Example code: shows how to find an intersection given a hit.
7. Example code: shows how to find a hit given an intersection - it is allowed to not have a match.
8. Example code: loop over all matches.
9. Assns< $A$,B> can be used as Assns< $B$,A>.
10. under the cover the data product is just pairs of art::Ptr<A> and art::Ptr<B>
11. Not illustrated Assns<A,B,D>. Statement about this and a reference to a later example.

12. Disscussion of when to use Ptr and when to use Assns.
13. Third option: data products with lock-step indexing.

# 23 Facade Pattern: Fitted Helices

1. This has a read facade only.
2. Fitted helix is a geometric object. No knowledge of momentum until you add external information (ie the magnetic field).
3. The data product is FittedHelixData
4. The fully powerful object is FittedHelix - it is not a data product because it contains non-data-product information, like it's connection to services. Breaking this rule can lead to linkage loops.
5. Some code to explore the properites of a fitted helix and make histograms.

# 24 art::View

1. Do we really want to do this without a good example of polymorphic data?
2. Show the mechanics of how to do it using one of the GenParticle exercises.
3. State that it is useful for polymorphic data ( which we won't have an example of here unless we can make one using TOF and tracker).

# 25 Random Numbers

1. Brief discussion: engines vs distributions; seeds vs state;
2. CLHEP weirdness: fire and shoot and global engines.
3. Two modules: one with a single engine and one with two engines
4. The single engine module will print an int from a flat distribution, limited by maxprint. Printout will be event number and value of the random variate.
5. Histograms of flat, gauss, poission and a few others.
6. Always use RandGausQ and RandPoissonQ
7. Special case of Geant4 and CLHEP global engine
8. RandGeneral - belongs in UserGuide not here.
9. Discussion of seeding strategies; should I put the seed service into the toyExperiment?
10. Save the seed. Write an output file.
11. Upon request art team could provide StdRandomNumberService that works with random engines from the C++11 std library instead of the CLHEP ones. But, for now, G4 controls.

# 26 Repeating Random Numbers

1. Start at event N and verify that random numbers are properly repeated.
2. Run 1000 events; then run 500 events and save the state at end of job; continue the job and compare to the full job. Just use the module that prints int's drawn from a flat distribution.
3. Warning: sometimes G4 hadronics code does not play nice.

# 27 Writing Your Own Service

1. Discuss legacy service, global service and schedule local services.
2. Inspect PDT, Geometry and Conditions services in the toyExperiment code.
3. Assign a project to write a service that duplicates the behaviour of tracer service? Is there a better project?

# 28 Running the MC Chain

1. Copy input04.fcl.
2. Explain what it does.
3. add the phi meson candidate producer
4. Run the MC chain.
5. Run some analyzer modules on the output.

# 29 Reconstruction on Demand

1. Describe reco on demand
2. Show the .fcl file that implements the MC chain as reco on demand.
3. Use –tracer to follow what it is doing.

# 30 Run the Exisitng 2D EventDisplay

1. Describe what the fcl parameters do.
2. Scan some events.
3.

# 31 Review of all Modules in toyExperiment

1. Discuss the code and other files in the toyExperiment package.
2. Say what each does.
3. Comment on features found in each that were not covered by the above.

# 32 Running a Grid Job

1. Cartoon description of the grid
2. Run the MC chain as a grid job.
3. Compare histograms
4. Need to work with FIFE people to define this.

# 33 Introducing SAM and dcache

1. Need to work with FIFE people to define this.

# 34 3D Event Displays

1. Mike Wang CMS style event display
2. Exercise the event processor that supports rewind

# 35 Live Histogram Update

1. Run MC events and update live histograms.

# 36 Checklist

Make sure that all of the following have a home above:

1. Add suggested exercises (and solutions) to each chapter; some are already there but it is incomplete. Some exercises should be examples of code that does not compile or which produces incorrect results; the task is to fix the problem. Others should be of the form: write code to do something. In both cases we need to provide verified solutions.

2. What is the best way to illustrate art::Assns<A,B,D>? One option is MCTruth matching of Reco tracks; here D is the chisquared of the match or something like that. Will I get enough multiple matches to make this example work? Maybe reduce the resolution of the tracker until I do get a few dozen cases of multiple matches in 1000 events?e A second option is a barrel TOF system in which the D object is something like the inferred time and particle ID probabilities. Use the particle ID probabilities when making the K+K- mass plot.

3. Changes to the toyExperiment product

   - Follow the guidelines so that only member data, c'tor and d'tor are expossed to genreflex
   - Add version numbers to the classes following Chris Green's model. A path for future expansion is to make a change to a class that requires a handwritten streamer?
   - Add the extra genparticles and modify downstream code; maybe exponentially falling
   - Find a way to naturally introduce map_vector
   - Find a way to introduce EnumToString - maybe in GenParticle? Or is that too much too soon.

4. admonition to always start a new login session for new projects - how many different places?

5. run and subrun data products

6. scrub .fcl discussion for inappropriate use of “keyword”.
7. Event::getManyByType and Selector functions.
8. Exercises about making Ptrs. This includes Ptr to a data product already in the event and Ptr to a data product that is being produced in the current module.
9. PtrVector<T> and std::vector< $art :: PTr$ <T>>;
10. map_vector
11. Making subdirectories in the TFileService
12. Polymorphic data and Views. Maybe the right model for a view is to have HOTs and hits. If a Hit subclasses off of hit, then it is a good model to excercsie View.
13. ParameterSet part 2:
    (a) include, prolog, @local, redefining parameters
    (b) ART_DEBUG_CONFIG=1 mu2e -c Ex03/ex03.fcl
    (c) FHICL_FILE_PATH
14. Grid/SAM
    (a) Generate and Reco MC
    (b) Probe job
    (c) Stage in and stage out files.
15. Check to see which of the CLHEP things I really need in the dictionary for DataProducts.
16. Modify toyExperiment to add more tracks; update the description and figures in the document.
17. Timebomb module to illustrate exceptions.
18. Memory leak module to illustrate the crude memory leak checker.
19. Show how to send events that give an exception to a separate ouptut file.
20. Parse a separate fcl file.
21. Message logger system
22. Services calling services
23. How to ensure that service B called before service A?
24. Drop on output; drop on input
25. write your own TNtuple and TTree; for TTree need to discuss provision of dictionaries.

26. enum matched to string class template.
27. validity checking of psets
28. examples to illustrate how to use gdb, totalview etc
29. Example of how to use the timing service; do we need to slow modules for this to work well? Maybe a producer module that finds all prime numbers less than N - choose N from RandGauss?
30. VERBOSE flag on buildtool - probably belongs in the Build and Run the first module section?
31. TFileDirectory and making directories within the TFileService.
32. std::vector hygiene: understand that it has a capacity and that it auto-reallocates as needed. Use reserve. Understand what it means that push_back ( emplace_back) invalidates all iterators. Know that insertion on a list or deque does not invalidate iterators.
33. scrub the text to remove instances of "variable" - prefer to use object in most cases. There may be a few cases in which variable is what we want.
34. scrub the text to remove "method" in favor of "member function" - may be complete
35. scrub the text for definitions of site. It needs to be expanded to include "your laptop". A particular experiment+institution may have several sets of computing hardware - their own cluster, their desktops, their laptops, the cluster beloning to their friend from which they borrow cycles...
36. an exercise to illustrate cet::map_vector
37. Does it make sense to teach some git stuff as follows:
    (a) clone the code from our repository
    (b) make a new repository under your name on redmine ( or elsewhere)
    (c) push all of the code to their repository
    (d) commit changes as needed

    I don't see how to deal with updates.
38. Tell people about =delete and =default. We choose a style in which we do not routinely use =default. Your experiment may or may not have a policy.
39. Items added during the brainstorming session June 19, 2014
    - tour of the art supplied services; timer, memory, fpu, scheduler
    - How to modify exception behaviour
    - n of m prescaler module

- Open event-data root file and navigate it
- Need more excercises as targets of opportunity.
- One algorithm with intermediate complexity: clustering, helix fitting
- Provide a test suite for the above bullet.
- Exercise: producer to add noise hits - number controlable from fcl Modify the event display to display them. This can go as an exercise in the write a producer.
- Commit to their own local repository - on branch.
- Loop Gen particle - write it as loop that calls a function. foreach with lambda.
- use distance function instead of address arithmetic.
- message logger - Log verbatim? How to define a category or destination that does not prescale, ever.

40. how to use -e and –estart command line option
41. Clean up the getting your C++ up to speed tarball. It has binaries and editor backups. It might also be missing a sample log file that it is supposed to have
42. Consider doing the following. Figure out a way so that the build system can be told to run everything and generate all of the sample output and autoinject it.
43. Can we make the text robust against line numbers that change in listings?
44. Anne wanted to automatically include text from source files. Marc also wants to. In both cases I objected because I don't want people to learn to format source code in strange ways that exist solely for the purpose of fitting on an 8.5x11 piece of paper. Marc suggested that we look into the LaTeX bashful package. I suggested that we have draft and production modes. Production mode will run the code, redirect output to file and include the files into the text. Draft mode will just use the existing files from the previous run. To make this work all of the tooling (makefiles and the files to be included, must be part of the git repository).
45. Check email from Rob Plunkett and fix the bugs he found.
46. Document the temporary file names for the output files and the tfileservice
47. Document how to not store events or subruns, just runs; in order to make summaries of run only run and subrun only.
48. Examples of producing the new time tracker and memory tracker databases and code to read them.

49. Find a place to reference the description of the e and s qualifiers
https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/AboutQualifiers

# 37 Classes in the toyExperiment

This is meant to be a manifest of the toyExperiment data product. Fill in any details that are not well covered in the above.

In DataProducts:

**EnumToString** Def

**EnumToStringSparse**

**PDGCode**

In MCDataProducts

**GenParticle**

**GenParticleCollection**

**Intersection**

**IntersectionCollection**

**MCRunSummary**

**TrkHitMatch**

In RecoDataProducts

**DetectorStatus**

**DetectorStatusRecord**

**FittedHelixData**

**FittedHelixDataCollection**

**Helix**

**RecoTrk**

**RecoTrkCollection**

**TrkHit**

**TrkHitCollection**

Producer modules:

**DetectorSimulation_module**

**EventGenerator_module**

**HitMaker_module**

**FindAndFitHelix_module**

Analyzer modules:

**EventDisplay_module**

**HelloWorld_module**

**InspectFittedHelix_module**

**InspectGenParticles_module**

**InspectIntersections_module**

**InspectTrkHits_module**

**MassPlot_module**

Analyzer modules:

**Conditions/Conditions_service**

**Geometry/Geometry_service.**

**Conditions/Conditions**

**Geometry/Geometry**

Other source files:

**Analyzers/PlotOrientation**

**Conditions/ParticleInfo**

**Conditions/PDT**

**Conditions/ShellConditions**

**Geometry/IntersectionFinder**

**Geometry/Shell**

**Geometry/Tracker**

**Geometry/TrackerComponent**

**Reconstruction/FittedHelix**

**Utilities/Decay2Body**

**Utilities/dphi**

**Utilities/eventIDToString**

**Utilities/ParameterSetFromFile**

**Utilities/phi_norm**

**Utilities/phi_small**

**Utilities/polar3Vector**

**Utilities/RandomUnitSphere**

**Utilities/safeSqrt**

**Utilities/sqrtOrThrow**

**Utilities/TwoBodyKinematics**

Directories:

**Analyzers**

**Conditions**

**DataProducts**

**Geometry**

**HelloWorldScripts**

**MCDataProducts**

**RecoDataProducts**

**Reconstruction**

**Simulations**

**Utilities**

**bin**

**databaseFiles**

**fcl**

**inputFiles**