Daya Bay Offline Software User Manual

December 17, 2015

Contents

Chapter 1

Introduction

1.1 Intended Audience

This manual describes how Daya Bay collaborators can run offine software jobs, extend existing functionality and write novel software components. Despite also being programmers, such individuals are considered "users" of the software. What is not described are internal details of how the offline software works which are not directly pertinent to users.

This document covers the software written to work with the Gaudi framework[1]. Some earlier software was used during the Daya Bay design stage and is documented elsewhere[1].

1.2 Document Organization

The following chapter contains a one to two page summary or "quick start" for each major element of the offline. You can try to use this chapter to quickly understand the most important aspects of a major offline element or refer back to them later to remind you how to do something.

Each subsequent chapter gives advanced details, describes less used aspects or expand on items for which there is not room in the "quick start" section.

1.3 Contributing

Experts and users are welcome to contribute corrections or additions to this documentation by commiting `.tex` or `.rst` sources. **However:**

> *Ensure latex compiles before committing into dybsvn*

1.4 Building Documentation

To build the plain latex documentation:

```
cd $SITEROOT/dybgaudi/Documentation/OfflineUserManual/tex
make plain     ## alternatively: pdflatex main
```

To build the Sphinx derived latex and html renderings of the documentation some non-standard python packages must first be installed, as described docs[2]. After this the Sphinx documentation can be build with:

```
. ~/v/docs/bin/activate    # ~/v/docs path points to where the "docs" virtualpython is created
cd $SITEROOT/dybgaudi/Documentation/OfflineUserManual/tex
make
```

[1] See chapter 6.

[2] http:///dayabay.bnl.gov/oum/docs

1.5 Typographical Conventions

This is bold text.

Chapter 2

Quick Start

This chapter holds brief "quick start" information about each major offline software element.

2.1 Offline Infrastructure

2.2 Installation and Working with the Source Code

2.2.1 Installing a Release

1. Download `dybinst`[1].
2. Run it: `./dybinst RELEASE all`

The `RELEASE` string is `trunk` to get the latest software or `X.Y.Z` for a numbered release. The wiki topic Category:Offline_Software_Releases[2] documents avilable releases.

2.2.2 Using an existing release

The easiest way to get started is to use a release of the software that someone else has compiled for you. Each cluster maintains a prebuilt release that you can just use. See the wiki topic Getting_Started_With_Offline_Software[3] for details.

2.2.3 Projects

A project is a directory with a `cmt/project.cmt` file. Projects are located by the `CMTPROJECTPATH` environment variable. This variable is initialized to point at a released set of projects by running:

```
shell> cd /path/to/NuWa-RELEASE
bash> source setup.sh
tcsh> source setup.csh
```

Any directories holding your own projects should then be **prepended** to this colon (":") separated `CMTPROJECTPATH` variable.

2.2.4 Packages

A package is a directory with a `cmt/requirements` file. Packages are located by the `CMTPATH` environment variable which is **automatically** set for you based on `CMTPROJECTPATH`. You should **not** set it by hand.

2.2.5 Environment

Every package has a setup script that will modify your environment as needed. For example:

```
shell> cd /path/to/NuWa-RELEASE/dybgaudi/DybRelease/cmt/
shell> cmt config  # needed only if no setup.* scripts exist
bash> source setup.sh
tcsh> source setup.csh
```

[1] http://dayabay.ihep.ac.cn/svn/dybsvn/installation/trunk/dybinst/dybinst
[2] https://wiki.bnl.gov/dayabay/index.php?title=Category:Offline_Software_Releases
[3] https://wiki.bnl.gov/dayabay/index.php?title=Getting_Started_With_Offline_Software

2.3 Offline Framework

2.4 Data Model

2.5 Detector Description

2.6 Kinematic Generators

2.7 Detector Simulation

2.8 Quick Start with Truth Information

Besides hits, `DetSim`, through the `Historian` package can provide detailed truth information in the form of *particle histories* and *unobservable statistics*. These are briefly described next and in detail later in this chapter.

2.8.1 Particle History

As particles are tracked through the simulation information on where they traveled and what they encountered can be recorded. The particle history is constructed with tracks (`SimTrack` objects) and vertices (`SimVertex` objects). Conceptually, these may mean slightly different things than what one may expect. A vertex is a 4-location when something "interesting" happened. This could be an interaction, a scatter or a boundary crossing. Tracks are then the connection between two vertices.

Because saving all particle history would often produce unmanageably large results rules are applied by the user to specify some fraction of the total to save. This means the track/vertex hierarchy is, in general, truncated.

2.8.2 Unobservable Statistics

One can also collect statistics on unobservable values such as number of photons created, number of photon backscatters, and energy deposited in different ADs. The sum, the square of the sum and the number of times the value is recorded are stored to allow mean and RMS to be calculated. The same type of rules that limit the particle histories can be used to control how these statistics are collected.

2.8.3 Configuring Truth Information

The rules that govern how the particle histories and unobservable statistics are collected are simple logical statements using a C++ like operators and some predefined variables.

Configuring Particle Histories

The hierarchy of the history is built by specifying selection rules for the tracks and the vertices. Only those that pass the rules will be included. By default, only primary tracks are saved. Here are some examples of a track selection:

```
# Make tracks for everything that's not an optical photon:
trackSelection = "pdg != 20022"
# Or, make tracks only for things that start
# in the GD scintillator and have an energy > 1Mev
trackSelection =
  "(MaterialName == '/dd/Materials/GdDopedLS') and (E > 1 MeV)"
```

And, here are some examples of a vertex selection:

```
# Make all vertices.. one vertex per Step.
vertexSelection = "any"
# Make vertices only when a particle crosses a volume boundary:
vertexSelection = "VolumeChanged == 1"
```

As an aside, one particular application of the Particle Histories is to draw a graphical representation of the particles using a package called GraphViz[4]. To do this, put the `DrawHistoryAlg` algorithm in your sequence. This will generate files in your current directory named `tracks_N.dot` and `tracks_and_vertices_N.dot`, where `N` is the event number. These files can be converted to displayable files with GraphViz's dot program.

Configuring Unobservable Statistics

What statistics are collected and when they are collected is controlled by a collection of triples:

1. A name for the statistics for later reference.
2. An algebraic formula of predefined variables defining the value to collect.
3. A rule stating what conditions must be true to allow the collection.

An example of some statistic definitions:

```
stats = [
    ["PhotonsCreated" , "E" , "StepNumber==1 and pdg==20022" ]
   ,["Photon_bounce_radius" , "r" , "pdg==20022 and dAngle > 90" ]
   ,["edep-ad1"  ,"dE"  ,"pdg!=20022 and
      ((MaterialName == '/dd/Materials/LiquidScintillator' or
        MaterialName == '/dd/Materials/GdDopedLS') and AD==1)"  ]
   ]
```

2.8.4 Accessing the resulting truth information

The resulting Truth information is stored in the `SimHeader` object which is typically found at `/Event/Sim/SimHeader` in the event store. It can be retrieved by your algorithm like so:

```
DayaBay::SimHeader* header = 0;
if (exist<DayaBay::SimHeader>(evtSvc(),m_location)) {
   header = get<DayaBay::SimHeader>(m_location);
}
const SimParticleHistory* h = header->particleHistory();
const SimUnobservableStatisticsHeader* h = header->unobservableStatistics();
```

[4] http://graphviz.org

2.9 Electronics Simulation

2.10 Trigger Simulation

The main algorithm in TrigSim, `TsTriggerAlg` has 3 properties which can be specified by the user.

TrigTools `Default:` "TsMultTriggerTool" List of Tools to run.

TrigName `Default:` "TriggerAlg" Name of the main trigger algorithm for bookkeeping.

ElecLocation `Default:` "/Event/Electroincs/ElecHeader" Path of ElecSimHeader in the TES, currently the default is picked up from ElecSimHeader.h

The user can change the properties through the TrigSimConf module as follows:

```
import TrigSim
trigsim = TrigSim.Configure()
import TrigSim.TrigSimConf as TsConf
TsConf.TsTriggerAlg().TrigTools = [ "TsExternalTriggerTool" ]
```

The `TrigTools` property takes a list as an argument allowing multiple triggers to be specified. Once implemented, the user could apply multiple triggers as follows:

```
import TrigSim
trigsim = TrigSim.Configure()
import TrigSim.TrigSimConf as TsConf
TsConf.TsTriggerAlg().TrigTools = [ "TsMultTriggerTool" ,
                                    "TsEsumTriggerTool" ,
                                    "TsCrossTriggerTool" ]
```

2.11 Readout

The default setup for Readout Sim used the `ROsFecReadoutTool` and `ROsFeeReadoutTool` tools to do the FEC and FEE readouts respectivly. The default setup is as follows

```
import ReadoutSim
rosim = ReadoutSim.Configure()
import ReadoutSim.ReadoutSimConf as ROsConf
ROsConf.ROsReadoutAlg().RoTools=["ROsFecReadoutTool","ROsFeeReadoutTool"]
ROsConf.ROsFeeReadoutTool().AdcTool="ROsFeeAdcPeakOnlyTool"
ROsConf.ROsFeeReadoutTool().TdcTool="ROsFeeTdcTool"
```

where the Fee will be read out using the tools specified via the `TdcTool` and `AdcTool` properties. Currently the only alternate readout tool is the `ROsFeeAdcMultiTool` which readout the cycles specified in the `ReadoutCycles` relative to the readout window start. The selection and configuration of this alternate tool is

```
ROsConf.ROsFeeReadoutTool().AdcTool="ROsFeeAdcMultiTool"
ROsConf.ROsFeeAdcMultiTool().ReadoutCycles=[0,4,8]
```

2.12 Event Display

2.12.1 A Plain Event Display: EvtDsp

A plain event display module, EvtDsp, is available for users. It makes use of the basic graphic features of the "ROOT" package to show the charge and time distributions of an event within one plot. One example is shown in Fig. 2.1. A lot of features of ROOT are immediately available, like "save as" a postscript file. All PMTs are projected to a 2-D plain. Each PMT is represented by a filled circle. The radii of them characterize the relative charge differences. The colors of them show the times of them, i.e. the red indicates the smallest time and the blue indicates the largest time.

Simple Mode

One can use a default simple algorithm to invoke the EvtDsp module. The charge and time of the first hit of each channel will be shown. Once setting up the nuwa environment, the following commands can be used to show events.

```
shell> nuwa.py -n -1 -m EvtDsp DayaBayDataFile.data
shell> nuwa.py --dbconf "offline_db" -n -1 -m "EvtDsp -C" DayaBayDataFile.data
shell> nuwa.py -n -1 -m "EvtDsp -S" DayaBaySimulatedFile.root
```

where the first one, by default, will show the raw information, i.e. delta ADC (ADC-preADC) and TDC distributions from ReadoutHeader, the second one will show calibrated result, CalibReadoutHeader, in PE and ns, as seen in Fig. 2.1 and the last line is for SimHeader, i.e. information is directly extracted from MC truth.

A simple readouts grouping was implemented. Readouts with delta trigger times within $2\mu s$ are considered as one event and shown together. But an event only allows one readout for one detector. For example a very close retrigger after an energetic muon in the same AD will start a new event. This algorithm also works for calibReadout and simHeader.

Advance Mode

One can also directly call the Gaudi Tool, EvtDsp, and plot the charges and times calculated in a different manner. In the simple mode, no selection is applied to select hits, however this is not the best choice in some cases, for example, some hits' times are out of the physically allowed window, like the blue hit in the inner water shield in Fig. 2.1 seems like a noise hit. One can also make a selection in an analysis algorithm to show only a fraction of interesting events or have a different event grouping algorithm. To use this feature one need to follow the standard Gaudi procedure to locate a tool "EvtDsp" first, i.e., add use EvtDsp module in cmt requirements file

```
use EvtDsp v*  Visualization
```

then get access to this tool

```
#include "EvtDsp/IEvtDsp.h"

IEvtDsp* m_evtDsp
StatusCode sc = toolSvc()->retrieveTool("EvtDsp","EvtDsp",m_evtDsp);
```

After this three simple interfaces are available and they can be plugged into anywhere of a user code.

```
/// Plot AD
virtual StatusCode plotAD(DayaBay::Detector det,
                          double chrg[8][24],       double time[8][24],
                          const char* chrgunit = 0, const char* timeunit = 0,
                          const char* info = 0 ) = 0;

/// Plot pool
virtual StatusCode plotPool(DayaBay::Detector det,
```

```
9                               double chrg[9][24][2],  double time[9][24][2],
10                              const char* chrgunit = 0, const char* timeunit = 0,
11                              const char* info = 0 ) =0;
12
13  /// A pause method for user. After this all displayed stuff will be flushed.
14  virtual StatusCode pause() = 0;
```

where for AD, chrg and time are arrays indexed by ring-1 and column-1, while for water pool, chrg and time arrays are indexed by wall-1,spot-1 and inward.

2.13 Reconstruction

2.14 Database

The content of this quickstart has been migrated to sop/[5]

[5] http:///dayabay.bnl.gov/oum/sop/

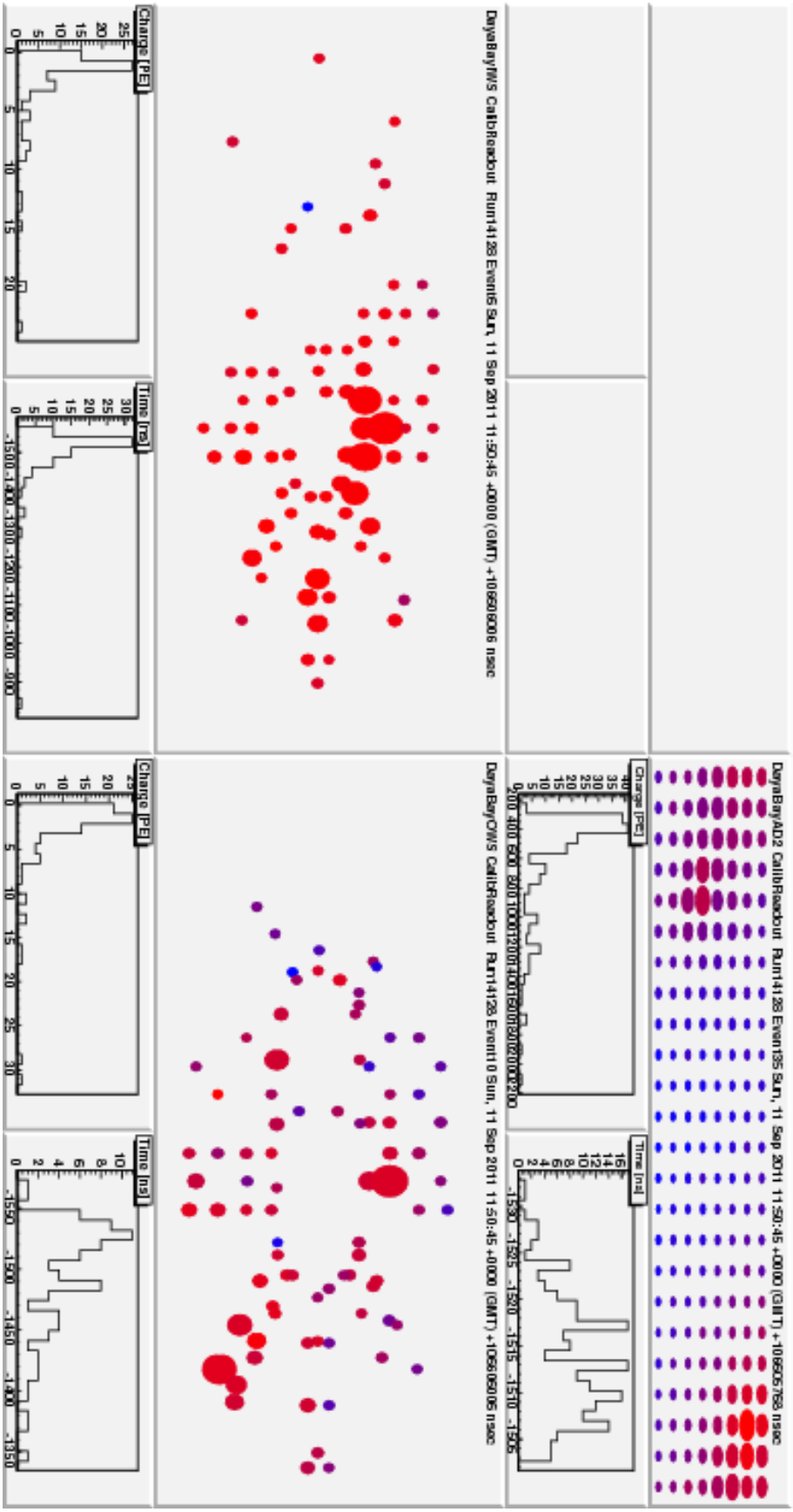

Figure 2.1: A snapshot for EvtDsp for a muon event which passed outer and inner water pool and struck AD No. 2, while AD No. 1 was quiet. The time and charge patterns of the AD and water pool hits are clearly seen.

Chapter 3

Analysis Basics

3.1 Introduction

This guide will help you analyze Daya Bay data. It contains a short description of the Daya Bay data and analysis software, called NuWa. It is not a detailed technical manual. In this document you can learn how to:

- Open a data file and see what it contains [Sec. 3.2.1]
- Draw histograms of the data in the file [Sec. 3.2.2]
- Use NuWa to do more detailed calculations with the data [Sec. 3.3]
- Write your own NuWa analysis module [Sec. 3.4.7]
- Write your own NuWa analysis algorithm [Sec. 3.4.8]
- Select events using *tags* [Sec. 3.4.2]
- Add your own data variables to the data file [Sec. 3.4.3]
- Filter data based on data path or tag [Sec. 3.4.5]

A set of cheat-sheets are included. These give short descriptions of the data and other NuWa features.

3.2 Daya Bay Data Files

Daya Bay uses ROOT files for data analysis. Basic analysis can be done with these files using only the ROOT program (http://root.cern.ch). For more complex analysis, see the Section 3.3 on using NuWa. If you do not have ROOT installed on your computer, you can access it on the computer clusters as part of the NuWa software (Sec. 3.5.1).

3.2.1 Opening data files

Daya Bay data files can be opened using the ROOT program,

```
shell> root
root[0] TFile f("recon.NoTag.0002049.Physics.DayaBay.SFO-1._0001.root");
root[1] TBrowser b;
root[1] b.BrowseObject(&f);
```

The ROOT browser window will display the contents of the file, as shown in Fig. 3.1. Event data is found under the path `/Event`, as summarized in Table 3.1. A section on each data type is included in this document. Simulated data files may include additional data paths containing "truth" information. A complete list of data paths are given in Sec. 3.5.5.

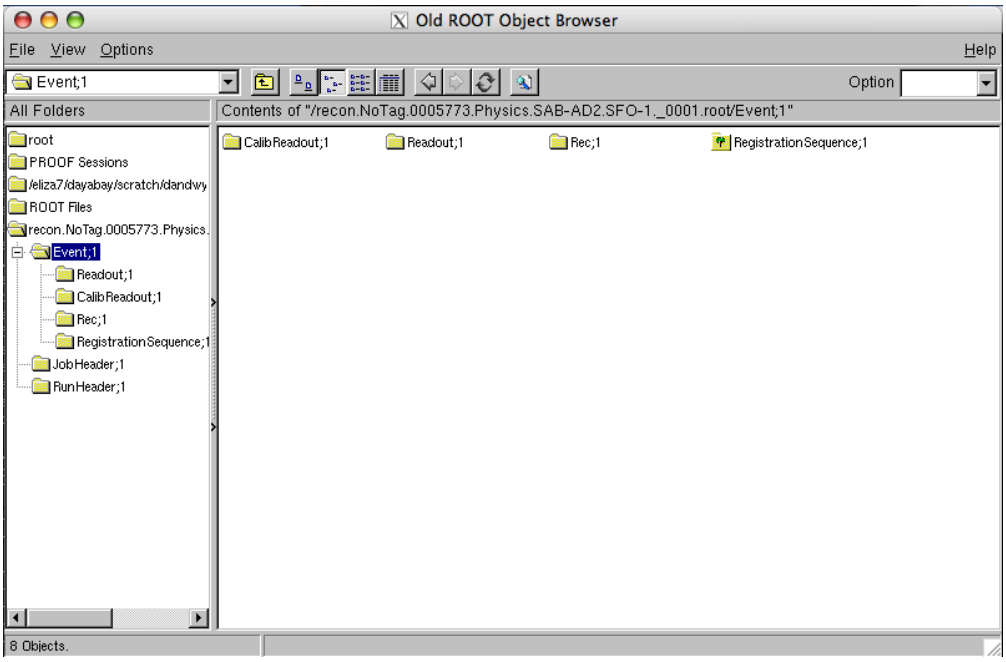

Figure 3.1: Data File Contents

A set of standard data ROOT files will be maintained on the clusters. The file prefix is used to identify the contents of the file, as shown in Table 3.2. The location of these files on each cluster are listed in Section 3.5.4.

Each data paths in the ROOT file contains ROOT trees. You can directly access a ROOT tree,

```
root[0] TFile f("recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root");
root[1] TTree* AdSimple = (TTree*)f.Get("/Event/Rec/AdSimple");
```

Table 3.1: Standard paths for Event Data

	Real and Simulated Data	
`/Event/Readout`	Raw data produced by the experiment	Sec. 3.5.8
`/Event/CalibReadout`	Calibrated times and charges of PMT and RPC hits	Sec. 3.5.9
`/Event/Rec`	Reconstructed vertex and track data	Sec. 3.5.11
	Simulated Data Only	
`/Event/Gen`	True initial position and momenta of simulated particles	
`/Event/Sim`	Simulated track, interactions, and PMT/RPC hits (Geant)	
`/Event/Elec`	Simulated signals in the electronics system	
`/Event/Trig`	Simulated signals in the trigger system	
`/Event/SimReadout`	Simulated raw data	

Table 3.2: Standard NuWa Event Data files

File Prefix	Readout	CalibReadout	Rec	Coinc	Spall	Simulation Truth (Gen,Sim)
daq.	yes					optional
calib.	optional	yes				optional
recon.	some events	some events	yes			optional
coinc.	some events	some events	some events	yes		optional
spall.	some events	some events	some events		yes	optional

The next section gives examples of working with these ROOT Trees. See the ROOT User's Guide for more details on working with Trees, http://root.cern.ch/download/doc/12Trees.pdf.

3.2.2 Histogramming data

Data can be histogrammed by selecting items in the TBrowser, or by using the Draw() function of the tree. For example, Figure 3.2 shows the data contained in a reconstructed event.

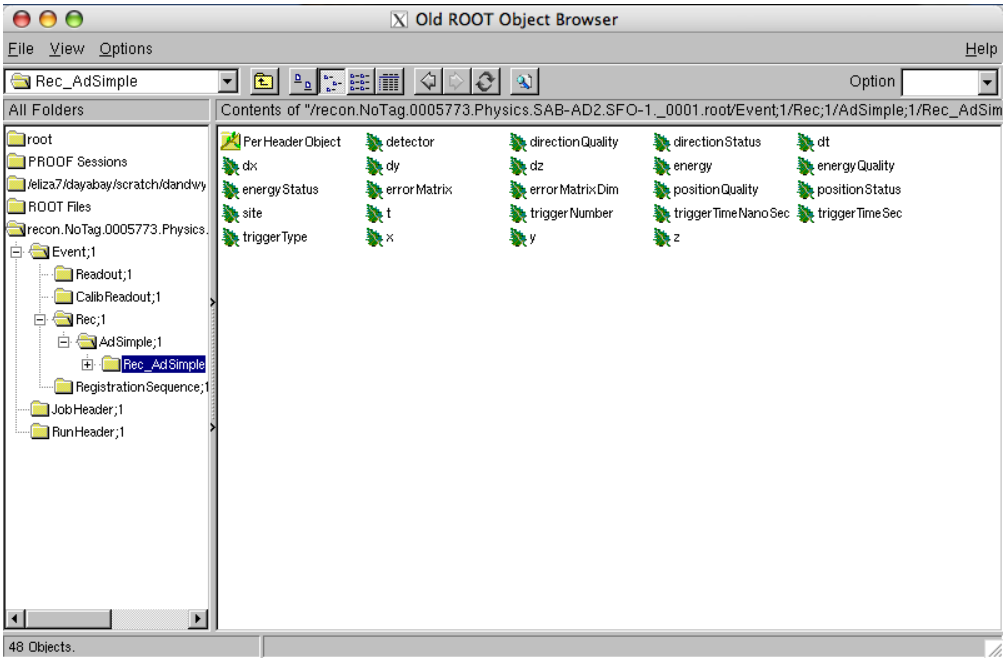

Figure 3.2: Example Reconstructed Data

The Draw() function allows the addition of selection cuts. For example, we can draw the reconstructed energy for all events where the reconstruction was successful by selecting events with energyStatus==1 and energy < 15 MeV,

```
root[2] AdSimple->Draw("energy","energyStatus==1 && energy<15");
```

Two- and three-dimensional histograms can be drawn by separating the variables with a colon. The third colz argument will use a color scale for a two-dimensional histogram. Fig. 3.3 shows the resulting histograms.

```
root[3] AdSimple->Draw("z:sqrt(x*x+y*y)","positionStatus==1","colz");
```

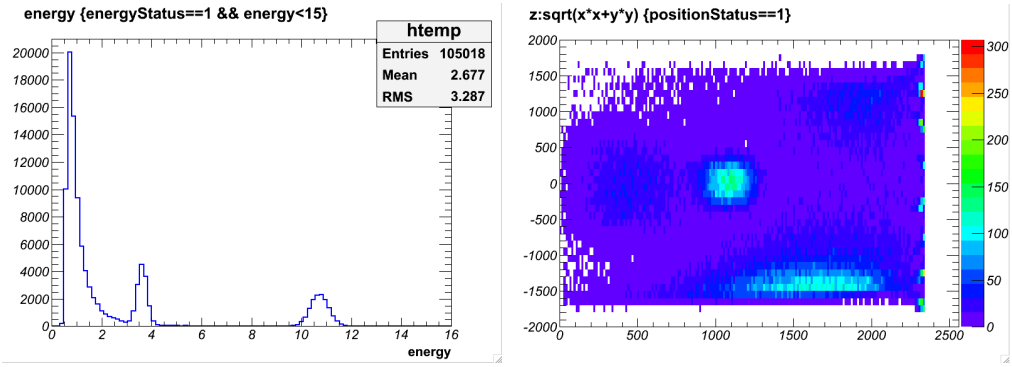

Figure 3.3: Example Histograms

A weighting can be added to each entry in histogram by multiplying your selection by the weighting factor (i.e. `weight*(selection)`. This can be used to draw the calibrated PMT charge distribution in `AD2` (Fig. **??**.) The charge distribution for a specfic event can be selected using the event number.

```
root[1] TTree* CalibReadoutHeader = (TTree*)f.Get("/Event/CalibReadout/CalibReadoutHeader");
root[2] CalibReadoutHeader->Draw("ring:column",
                                 "chargeAD*(detector==2)","colz")
root[3] CalibReadoutHeader->Draw("ring:column",
                                 "chargeAD*(detector==2 && eventNumber==12345)","colz")
```

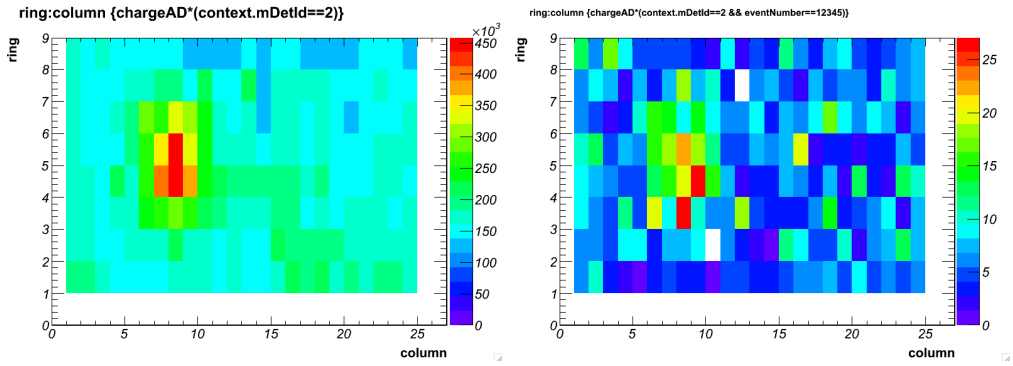

Figure 3.4: The calibrated PMT charge (in photoelectrons) for all events and for an individual event.

The trigger time is divided into two parts; a count of seconds from January 1970 (i.e. unixtime), and a precise count of nanoseconds from the last second. To draw the absolute trigger time, you must add these two counts. Figure 3.5 shows a histogram of the calibrated PMT hit charges versus trigger time[1]. The ROOT `Sum$()` function will histogram the sum of a quantity for each event; it can be used to histogram the sum of charge over all AD PMTs.

```
root[2] CalibReadoutHeader->Draw("chargeAD:triggerTimeSec+triggerTimeNanoSec*1e-9",
        "(detector==2 && ring==4 && column==15 && chargeAD>-3 && chargeAD<7)",
        "colz");
```

[1]The trigger time can be converted to a readable Beijing local time format using the lines described in Sec. 3.5.16

```
root[3] CalibReadoutHeader->Draw("Sum$(chargeAD):triggerTimeSec+triggerTimeNanoSec*1e-9",
                                 "detector==2 && Sum$(chargeAD)<1500","colz");
```

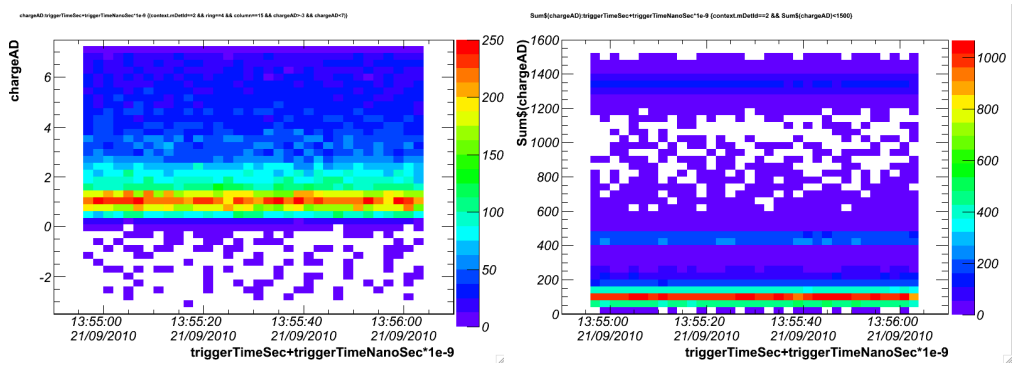

Figure 3.5: The calibrated charge (in photoelectrons) for one PMT and for the sum of all PMTs versus trigger time.

3.2.3 Histogramming Raw DAQ data

To properly histogram raw DAQ data from `/Event/Readout`, you will need to use part of the Daya Bay software in addition to ROOT. You must load the NuWa software, as described in Sec. 3.5.1. Running `load.C` will allow you to call functions in your `Draw()` command. For example, you can call the function to draw the raw fine-range ADC and TDC distributions for PMT electronics board 6, connector 5 (Fig. 3.6.) The selection on `context.mDetId==2` selects the detector `AD2`; Sec. 3.5.7 lists the allowed detector and site IDs. If you have a raw `.data` file produced by the DAQ, see section 3.5.8 to wrap it in a ROOT tree so that you can directly histogram the raw data.

```
root[0] .x $ROOTIOTESTROOT/share/load.C
root[1] TFile f("daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root");
root[2] TTree* ReadoutHeader = (TTree*)f.Get("/Event/Readout/ReadoutHeader");
root[3] ReadoutHeader->Draw("daqPmtCrate().adcs(6,5,1).value()","context.mDetId==2");
root[4] ReadoutHeader->Draw("daqPmtCrate().tdcs(6,5,1).value()","context.mDetId==2");
```

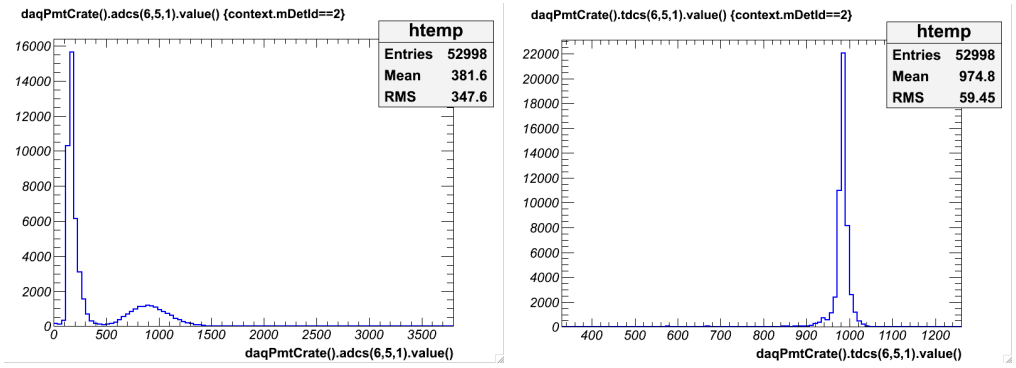

Figure 3.6: Histograms of Raw fine-range ADC and TDC values from PMT FEE board 6, connector 5.

3.2.4 Some ROOT Tree Tricks

A ROOT `TChain` can be used to combine the trees of the same path from multiple files into one large tree. For example, if a data run produced two files, you can combine the trees from these files:

```
root[0] TChain AdSimple("/Event/Rec/AdSimple");
root[1] AdSimple.Add("recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root");
root[2] AdSimple.Add("recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0002.root");
root[3] AdSimple.Draw("energy","energyStatus==1 && detector==2");
```

To combine all the variables from trees at different data paths into a single tree, you can use the `TTree::AddFriend()` function. This can be used to histogram or select using variables from both trees. This should only be done for trees that are synchronized. The raw, calibrated, and reconstructed data are generally synchronized, as long as the data has not been filtered. The simulated truth trees at `/Event/Gen` and `/Event/Sim` are generally not synchronized with the data trees since one simulated event may produce an arbitary number of triggered readouts.

```
root[1] TTree* CalibReadoutHeader = (TTree*)f.Get("/Event/CalibReadout/CalibReadoutHeader");
root[2] TTree* AdSimple = (TTree*)f.Get("/Event/Rec/AdSimple");
root[3] AdSimple->AddFriend( CalibReadoutHeader );
root[4] AdSimple->Draw("energy:nHitsAD","detector==2","colz");
```

See the ROOT User's Guide for more details on working with Trees, http://root.cern.ch/download/doc/12Trees.pdf.

3.2.5 Analysis Examples (or A Treatise on Cat-skinning)

> *What is the best / simplest / fastest way for me to examine event data and generate my histograms?*

If this is your question, then please read this section. As discussed in the preceding sections, you can directly use ROOT to inspect NuWa event data files. Within ROOT, there are a few different methods to process event data. Alternatively, you can use the full power NuWa to process data. To demonstrate these different methods, a set of example scripts will be discussed in this section. Each example script generates the exact same histogram of number of hit PMTs versus reconstructed energy in the AD, but uses a different methods. Each ROOT script shows how to "chain" trees from multiple files, and how to "friend" data trees from the same file. All example scripts can be found in the Tutorial/Quickstart[2] software package.

- `dybTreeDraw.C`: ROOT script using `TTree::Draw()`
- `dybTreeGetLeaf.C`: ROOT script using `TTree::GetLeaf()`
- `dybTreeSetBranch.C`: ROOT script using `TTree::SetBranchAddress()`
- `dybNuWaHist.py`: NuWa algorithm using the complete data classes

The example `dybTreeDraw.C` is the simplest approach; it is recommended that you try this method first when generating your histograms. If you plan to include your algorithm as part of standard data production, you will eventually need to use a NuWa algorithm such as `dybNuWaHist.py`. The other two methods are only recommended for special circumstances. A detailed description of the advantages and disadvantages of each approach are provided in the following sections.

[2] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tutorial/Quickstart

dybTreeDraw.C

This is the easiest approach and usually requires the least programming. Please consider using this approach first if possible.

Advantages:

- Simple to run
- Requires the least programming
- Easy for others to understand and reproduce
- Allows chaining and friending of data files

Disadvantages:

- Slower when you need to make many histograms
- Some cuts or variables cannot be expressed in a draw command
- No access to geometry, database, other external data
- Cannot be integrated with production analysis job

To run this example, use the following approach:

```
root [0] .L dybTreeDraw.C+
root [1] dybTreeDraw("recon*.root")
```

The key lines from the script are:

```
// Fill histograms
// AD#1
reconT.Draw("calibStats.nHit:energy>>nhitVsEnergyAD1H",
            "context.mDetId==1 && energyStatus==1");
// AD#2
reconT.Draw("calibStats.nHit:energy>>nhitVsEnergyAD2H",
            "context.mDetId==2 && energyStatus==1");
```

dybGetLeaf.C

There are some cases where the variables and cuts cannot be expressed in a simple `TTree::Draw()` command. Is this case, using `TTree::GetLeaf()` is an alternative. This is also a better alternative for those familiar with `TSelector` or `TTree::MakeClass`, since it allows chaining and friending of data files.

Advantages:

- Fairly simple to run
- Requires some minimal programming
- Allows chaining and friending of data files

Disadvantages:

- No access to geometry, database, other external data
- Cannot be integrated with production analysis job

To run this example, use the following approach:

```
root [0] .L dybTreeGetLeaf.C+
root [1] dybTreeGetLeaf("recon*.root")
```

The key lines from the script are:

```
// Process each event
int maxEntries=reconT.GetEntries();
for(int entry=0;entry<maxEntries;entry++){

  // Get next event
  reconT.GetEntry(entry);

  // Get event data
  int detector = (int) reconT.GetLeaf("context.mDetId")->GetValue();
  int energyStatus = (int) reconT.GetLeaf("energyStatus")->GetValue();
  double energy = reconT.GetLeaf("energy")->GetValue();
  int nHit = (int)reconT.GetLeaf("calibStats.nHit")->GetValue();

  // Fill histograms
  if(energyStatus==1){ // Reconstruction was successful
    if(detector==1){
      // AD#1
      nhitVsEnergyAD1H->Fill(energy,nHit);
    }else if(detector==2){
      // AD#2
      nhitVsEnergyAD2H->Fill(energy,nHit);
    }
  }
}
```

dybTreeSetBranch.C

Use this approach only if you really need the fastest speed for generating your histograms, and cuts cannot be expressed in a simple `TTree::Draw()` command. The example script relies on `TTree::SetBranchAddress()` to explicitly manage the event data location in memory. By avoiding reading data unnecessary data from the file, it also demonstrates how to achieve the highest speed.

Advantages:

- Fastest method to histogram data
- Allows chaining and friending of data

Disadvantages:

- Requires some careful programming
- No access to geometry, database, other external data
- Cannot be integrated with production analysis job

To run this example, use the following approach:

```
root [0] .L dybTreeSetBranch.C+
root [1] dybTreeSetBranch("recon*.root")
```

The key lines from the script are:

```
// Enable only necessary data branches
reconT.SetBranchStatus("*",0); // Disable all
calibStatsT.SetBranchStatus("*",0); // Disable all

// Must reenable execNumber since the tree indexing requires it
reconT.SetBranchStatus("execNumber",kTRUE);
reconT.SetBranchStatus("calibStats.execNumber",kTRUE);

int detector = 0;
```

```
10  reconT.SetBranchStatus("context.mDetId",kTRUE);
11  reconT.SetBranchAddress("context.mDetId",&detector);
12
13  int energyStatus = 0;
14  reconT.SetBranchStatus("energyStatus",kTRUE);
15  reconT.SetBranchAddress("energyStatus",&energyStatus);
16
17  float energy = -1;
18  reconT.SetBranchStatus("energy",kTRUE);
19  reconT.SetBranchAddress("energy",&energy);
20
21  int nHit = -1;
22  reconT.SetBranchStatus("calibStats.nHit",kTRUE);
23  reconT.SetBranchAddress("calibStats.nHit",&nHit);
24
25  // Process each event
26  int maxEntries=reconT.GetEntries();
27  for(int entry=0;entry<maxEntries;entry++){
28
29    // Get next event
30    reconT.GetEntry(entry);
31
32    // Fill histograms
33    if(energyStatus==1){ // Reconstruction was successful
34      if(detector==1){
35        // AD#1
36        nhitVsEnergyAD1H->Fill(energy,nHit);
37      }else if(detector==2){
38        // AD#2
39        nhitVsEnergyAD2H->Fill(energy,nHit);
40      }
41    }
42  }
```

dybNuWaHist.py

This example uses a full NuWa algorithm to generate the histogram. Use this approach when you need complete access to the event data object, class methods, geometry information, database, and any other external data. You must also use this approach if you want your algorithm to be included in the standard production analysis job. It is the most powerful approach to analysis of the data, but it is also the slowest. Although it is the slowest method, it may still be fast enough for your specific needs.

Advantages:

- Full data classes and methods are available
- Full access to geometry, database, other external data
- Can be integrated with production analysis job

Disadvantages:

- Slowest method to histogram data
- Requires some careful programming
- Requires a NuWa software installation

To run this example, use the following approach:

```
1  shell> nuwa.py -n -1 -m"Quickstart.dybNuWaHist" recon*.root
```

The key lines from the script are:

```
    def execute(self):
        """Process each event"""
        evt = self.evtSvc()

        # Access the reconstructed data
        reconHdr = evt["/Event/Rec/AdSimple"]
        if reconHdr == None:
            self.error("Failed to get current recon header")
            return FAILURE
        # Access the calibrated data statistics
        calibStatsHdr = evt["/Event/Data/CalibStats"]
        if reconHdr == None:
            self.error("Failed to get current calib stats header")
            return FAILURE

        # Check for antineutrino detector
        detector = reconHdr.context().GetDetId()
        if detector == DetectorId.kAD1 or detector == DetectorId.kAD2:

            # Found an AD.  Get reconstructed trigger
            recTrigger = reconHdr.recTrigger()
            if not recTrigger:
                # No Reconstructed information
                self.warning("No reconstructed data for AD event!?")
                return FAILURE

            # Get reconstructed values
            energyStatus = recTrigger.energyStatus()
            energy = recTrigger.energy()
            nHit = calibStatsHdr.getInt("nHit")

            # Fill the histograms
            if energyStatus == ReconStatus.kGood:
                if detector == DetectorId.kAD1:
                    self.nhitVsEnergyAD1H.Fill( energy/units.MeV, nHit )
                elif detector == DetectorId.kAD2:
                    self.nhitVsEnergyAD2H.Fill( energy/units.MeV, nHit )

        return SUCCESS
```

The next section provides more information on data analysis using NuWa (Sec. 3.3).

3.2.6 Advanced Examples

The following section presents advanced examples of working with Daya Bay data files. All example scripts can be found in the Tutorial/Quickstart[3] software package.

Combining 'Unfriendly' Trees

The examples in the previous section show how to histogram data by 'friending' trees. Trees can only be 'friended' if there is a natural relationship between the trees. The Coincidence and Spallation trees collect data from multiple triggers into one entry. As a consequence, you cannot 'friend' these trees with the trees which contain data with one trigger per entry (e.g. `CalibStats`, `AdSimple`, etc.). For example, you may want to histogram data in the `Coincidence` tree, but you want to apply a cut on a variable that is only present in `CalibStats`.

It is possible to combine data from these 'unfriendly' trees. The approach is to manually look up the data for the corresponding entries between the 'unfriendly' trees. By building on the example `dybTreeGetLeaf.C`, the advanced example `dybTreeGetLeafUnfriendly.C` generates a histogram with data from both the `Coincidence`

[3] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tutorial/Quickstart

and `CalibStats` data. The first step in this process is to create an index to allow a unique look-up of an entry from the `CalibStats` tree:

```
// Disable pre-existing index in the calib stats trees
//  (Another reason ROOT is frustrating; we must manually do this)
calibStatsT.GetEntries();
Long64_t* firstEntry = calibStatsT.GetTreeOffset();
for(int treeIdx=0; treeIdx<calibStatsT.GetNtrees(); treeIdx++){
  calibStatsT.LoadTree(firstEntry[treeIdx]);
  calibStatsT.GetTree()->SetTreeIndex(0);
}

// Build a new look-up index for the 'unfriendly' tree
//  (Trigger number and detector id uniquely identify an entry)
calibStatsT.BuildIndex("triggerNumber","context.mDetId");
```

Once this index is available, we can manually load a specific `CalibStats` entry with the call:

```
    // Look up corresponding entry in calib stats
    int status = calibStatsT.GetEntryWithIndex(triggerNumber, detector);
```

Now that we are prepared, we can step through each entry in the `Coincidence` tree. For each Coincidence multiplet we can look up all of the corresponding entries from the `CalibStats` tree. Here is the main loop over Coincidence entries from the example script, demonstrating how to fill a histogram with data from these unfriendly trees:

```
// Process each coincidence set
int maxEntries=adCoincT.GetEntries();
for(int entry=0;entry<maxEntries;entry++){

  // Get next coincidence set
  adCoincT.GetEntry(entry);

  // Get multiplet data
  int multiplicity = (int) adCoincT.GetLeaf("multiplicity")->GetValue();
  int detector = (int) adCoincT.GetLeaf("context.mDetId")->GetValue();
  std::vector<int>& triggerNumberV = getLeafVectorI("triggerNumber",&adCoincT);
  std::vector<int>& energyStatusV = getLeafVectorI("energyStatus",&adCoincT);
  std::vector<float>& energyV = getLeafVectorF("e",&adCoincT);

  // Loop over AD events in multiplet
  for(int multIdx=0; multIdx<multiplicity; multIdx++){

    // Get data for each AD trigger in the multiplet
    int triggerNumber = triggerNumberV[multIdx];
    int energyStatus = energyStatusV[multIdx];
    float energy = energyV[multIdx];

    // Look up corresponding entry in calib stats
    int status = calibStatsT.GetEntryWithIndex(triggerNumber, detector);
    if(status<=0){
      std::cout << "Failed to find calib stats for trigger number "
                << triggerNumber << " and detector ID " << detector
                << std::endl;
      continue;
    }
    // Get data from matching calib stats entry
    double nominalCharge = calibStatsT.GetLeaf("NominalCharge")->GetValue();

    // Fill histograms
    if(energyStatus==1 && energy>0){ // Reconstruction was successful
      if(detector==1){
        // AD#1
        chargeVsEnergyAD1H->Fill(energy,nominalCharge/energy);
      }else if(detector==2){
```

```
40          // AD#2
41          chargeVsEnergyAD2H->Fill(energy,nominalCharge/energy);
42        }
43      }
44
45    }  // End loop over AD triggers in the multiplet
46  } // End loop over AD coincidence multiplets
```

Using `TTree::Draw()` with 'Unfriendly' Trees

The previous example script allowed us to correlate and histogram data between the 'unfriendly' `Coincidence` and `CalibStats` trees. This example required that we manually loop on the individual entries in the `Coincidence` tree, and fill the histograms entry-by-entry. An alternate approach is to reformat the data from the 'unfriendly' `CalibStats` tree into a 'friendly' format. Once in this 'friendly' format, we can return to simple calls to `TTree::Draw()` to place cuts and histogram data. This approach is more technical to setup, but can be useful if you want to continue to use `TCuts`, or if you want to repeatedly histogram the data to explore the variations of cuts.

As discussed, this approach relies on reformatting the data from an 'unfriendly' tree into a 'friendly' format. The example script `dybTreeDrawUnfriendly.C` generates the same histograms as the previous example `dybTreeGetLeafUnfriendly.C`, but uses this alternate approach. The following lines shows this in practice:

```
1  // Create 'friendly' version of data from CalibStats
2  std::string mainEntriesName = "multiplicity";
3  std::vector<string> calibVarNames; //variable names to copy from CalibStats
4  calibVarNames.push_back("MaxQ");
5  calibVarNames.push_back("NominalCharge");
6  std::string indexMajorName = "triggerNumber";
7  std::string indexMinorName = "context.mDetId";
8  TTree* calibStatsFriendlyT = makeFriendTree(&adCoincT,
9                                              &calibStatsT,
10                                             mainEntriesName,
11                                             calibVarNames,
12                                             indexMajorName,
13                                             indexMinorName);
14 if(!calibStatsFriendlyT){
15   std::cout << "Failed to create friendly tree" << std::endl;
16   return;
17 }
18 // Add new friendly tree to coincidence tree
19 adCoincT.AddFriend(calibStatsFriendlyT,"calibStats");
```

Once this 'friendly' tree has been generated, we can use `TTree::Draw()` with the `CalibStats` variables:

```
1  // Fill histograms
2  // AD#1
3  adCoincT.Draw("calibStats.NominalCharge/e:e>>chargeVsEnergyAD1H",
4                "context.mDetId==1 && energyStatus==1 && e>0","colz");
5  // AD#2
6  adCoincT.Draw("calibStats.NominalCharge/e:e>>chargeVsEnergyAD2H",
7                "context.mDetId==2 && energyStatus==1 && e>0","colz");
```

The reformatted `CalibStats` data is available in the newly created tree `calibStatsFriendlyT`, which is dynamically created and kept in memory. Once you close your ROOT session, this tree will be deleted. If you wish to keep this 'friendly' tree around for later reuse, then you should write it to a file:

```
1  TFile outputFile("friendlyCalibStats.root","RECREATE");
2  calibStatsFriendlyT.SetDirectory(&outputFile);
3  calibStatsFriendlyT.Write();
```

The generation of this reformatted 'friendly' tree relies on the fairly complex helper function `makeFriendTree`:

```
TTree* makeFriendTree(TChain* mainT,
                      TChain* unfriendlyT,
                      const string& mainEntriesName,
                      const std::vector<string>& friendVarNames,
                      const string& indexMajorName,
                      const string& indexMinorName)
```

One entry in the tree `mainT` corresponds to multiple entries in the `unfriendlyT` tree; these are the `Coincidence` and `CalibStats` trees respectively in our example. `mainEntriesName` is the name of the branch in `mainT` that tells us the count of `unfriendlyT` entries that correspond to the current `mainT` entry. This is the variable `multiplicity` in our example, which tells us how many AD triggers are in the current coincidence multiplet. The variables names given in `friendVarNames` are reformatted from single numbers (i.e. `float friendVar`) in the `unfriendlyT` tree to arrays (i.e. `float friendVar[multiplicity]`) in the new 'friendly' tree returned by the function. For our example, these are the `CalibStat` variables `MaxQ` and `NominalCharge`. The `indexMajorName` and `indexMinorName` variables are present in both trees, and are used to correlate one entry in the `mainT` with multiple entries in the `unfriendlyT` tree. These are the variables `triggerNumber` and `context.mDetId`. Note that one or both of these index variables must be an array in the `mainT` tree to properly describe the 'unfriendly' one-to-many relationship between entries in `mainT` and `unfriendlyT`.

This helper function may require some slight modification for your specific case. It assumes that the branches have the following types:

- `mainEntriesName`: `integer` in `mainT`
- `friendVarNames`: `float` in `unfriendlyT`
- `indexMajorName`: `vector<int>` in `mainT` and `int` in `unfriendlyT`
- `indexMinorName`: `int` in both `mainT` and `unfriendlyT`

This helper function could be extended to dynamically check these variable types (eg. `float`, `int`, `vector<float>`, `vector<int>`, etc), and then respond accordingly. This is left as an exercise for the analyzer.

3.3 NuWa Basics

If you wish to do more analysis than histogramming data from files, you must use NuWa. NuWa is the name given to the analysis software written for the Daya Day experiment. It is installed and available on the computer clusters. To load the software on one of the clusters, see Sec. 3.5.1. To install NuWa on another computer, see Sec. 3.5.2.

NuWa analysis allows you to:

- Access all event data
- Relate data at different paths (ie. `/Event/Rec` to `/Event/Readout`)
- Access non-event data (ie. PMT positions, cable mapping, etc)
- Do more complex calculations
- Write NuWa data files

This section provides a short description of the `nuwa.py` program, Job Modules, and analysis algorithms. This is followed by a series of *recipes* for common analysis tasks.

3.3.1 The `nuwa.py` Command

The `nuwa.py` command is the main command to use the Daya Bay analysis software. A command has a structure similar to,

```
shell> nuwa.py -n <numberOfEntries> -m"<Module>" <inputFile>
```

A complete list of options is given in Sec **??**. An example is,

```
shell> nuwa.py -n 100 -m"Quickstart.PrintRawData" daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

In this simple example, the first 100 triggered readouts are read from the input file, and their data is printed to the screen. The `-n` option specifies the number of entries to process. The `-n -1` option will process all events in the input file(s). The `-m` option specifies how the job should be configured. Sec. 3.3.2 discusses job configuration using Job Modules.

An arbitrary number of input files can be given, and will be processed in sequence.

```
shell> nuwa.py -n <numberOfEntries> -m"<Module>" <inputFile1> <inputFile2>
```

The `-o` option can be used to write the event data to a NuWa output file,

```
shell> nuwa.py -n <numberOfEntries> -m"<Module>" -o <outputFile> <inputFile>
```

Some other useful options are,

- `--no-history`: Do not print out job configuration information to the screen
- `-l` *n*: Set the minimum level of logging output printed to the screen (1: VERBOSE, 2: DEBUG, 3: INFO, 4: WARNING, 5: ERROR)
- `-A` *n*`*s`: Keep events for the past *n* seconds available for correlation studies with the current event.
- `--help`: Print nuwa.py usage, including descriptions of all options.

3.3.2 NuWa Job Modules

Job modules are used to configure simulation and analysis tasks. Specifically, Job modules are scripts which do the following:

- Add analysis Algorithms and Tools to the job
- Configure Algorithms, Tools, and Services used by the job

Job Modules are used with the nuwa.py command as follows,

```
shell> nuwa.py -n 100 -m"<Module1>" -m"<Module2>" <inputFile>
```

You can put as many modules as you like on the command line. Some modules can take arguments; these should be placed inside the quotes immediately after the module name,

```
shell> nuwa.py -n 100 -m"<Module1> -a argA -b argB" <inputFile>
```

3.4 NuWa Recipes

Many NuWa analysis tasks rely on a standard or familiar approach. This section provides a list of *recipes* for common analysis tasks such as,

- See the history of a NuWa file [Sec. 3.4.1]
- Tag a set of events in a NuWa file [Sec. 3.4.2]
- Add your own variables to the NuWa file [Sec. 3.4.3]
- Copy all the data at a path to a new file [Sec. 3.4.5]
- Write tagged data to a new file [Sec. 3.4.6]
- Change the configuration of an existing Job Module [Sec. 3.4.7]
- Write your own analysis Algorithm [Python] [Sec. 3.4.8]
- Write your own analysis Algorithm [C++] [Sec. 3.4.9]
- Modify an existing part of NuWa [C++] [Sec. 3.4.10]

3.4.1 See the history of a NuWa File

Before using a NuWa data file, you may want to see what processing has already been done on the file. The following command will print the history of all NuWa jobs that have been run to produce this file:

```
shell> nuwa.py -n 0 --no-history -m"JobInfoSvc.Dump"
           recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

You will see much information printed to the screen, including the following sections which summarize the NuWa jobs that have been run on this file:

```
Cached Job Information:
{  jobId : daf3a684-6190-11e0-82f7-003048c51482
 cmtConfig : x86_64-slc4-gcc34-opt
 command : /eliza7/dayabay/scratch/dandwyer/NuWa-trunk-opt/dybgaudi/InstallArea/scripts/nuwa.py
           -n 0 --no-history -mJobInfoSvc.Dump
           recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
 hostid : 931167014
 jobTime : Fri, 08 Apr 2011 03:32:40 +0000
 nuwaPath : /eliza16/dayabay/users/dandwyer/installs/trunk_2011_03_30_opt/NuWa-trunk
 revision : 11307:11331
 username : dandwyer
}


Cached Job Information:
{  jobId : 6f5c02f4-6190-11e0-897b-003048c51482
 cmtConfig : x86_64-slc4-gcc34-opt
 command : /eliza7/dayabay/scratch/dandwyer/NuWa-trunk-opt/dybgaudi/InstallArea/scripts/nuwa.py
           -A None -n -1 --no-history --random=off -mQuickstart.DryRunTables
           -mQuickstart.Calibrate -mQuickstart.Reconstruct
           -o recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
           daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
 hostid : 931167014
 jobTime : Fri, 08 Apr 2011 03:29:39 +0000
 nuwaPath : /eliza16/dayabay/users/dandwyer/installs/trunk_2011_03_30_opt/NuWa-trunk
 revision : 11307:11331
 username : dandwyer
}
```

```
29
30
31 Cached Job Information:
32 {  jobId : 22c6620e-6190-11e0-84ac-003048c51482
33  cmtConfig : x86_64-slc4-gcc34-opt
34  command : /eliza7/dayabay/scratch/dandwyer/NuWa-trunk-opt/dybgaudi/InstallArea/scripts/nuwa.py
35            -A None -n -1 --no-history --random=off -mProcessTools.LoadReadout
36            -o daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
37            /eliza7/dayabay/data/exp/dayabay/2010/TestDAQ/NoTag/0922/daq.NoTag.0005773.Physics.SAB-AD2.SFO-
38  hostid : 931167014
39  jobTime : Fri, 08 Apr 2011 03:27:31 +0000
40  nuwaPath : /eliza16/dayabay/users/dandwyer/installs/trunk_2011_03_30_opt/NuWa-trunk
41  revision : 11307:11331
42  username : dandwyer
43 }
```

The jobs are displayed in reverse-chronological order. The first job converted the raw daq `.data` file to a NuWa `.root` file. The second job ran an example calibration and reconstruction of the raw data. The final job (the current running job) is printing the job information to the screen.

3.4.2 Tag Events in a NuWa File

Event tags are used to identify a subset of events. These can be used to separate events into classes such as *muons*, *inverse-beta decay*, *noise*, etc. In general, tags be used to identify any set of events of interest.

The job module Tagging/UserTagging/python/UserTagging/UserTag/DetectorTag.py[4] is a simple example of tagging readouts by detector type. The tag can be applied by adding the module to a NuWa job:

```
1 shell> nuwa.py -n -1 --no-history -m"UserTagging.UserTag.DetectorTag"
2            daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

To add your own tag, follow the steps for modifing an existing python module (section 3.4.8.) Use Tagging/UserTagging/python/UserTagging/UserTag/DetectorTag.py[5] as a starting point. You should add your own tag in the `initTagList` function:

```
1 self.addTag('MySpecialEvent' , '/Event/UserTag/MySpecialEvent')
```

In the `check` function, you should retrieve event data and decide if you want to tag it:

```
1 # Get reconstructed data
2 recHdr = evt["/Event/Rec/AdSimple"]
3 # Add your calculation / decision here
4 # ...
5 #
6 if tagThisEvent:
7     # Keep track of the reconstructed data you are tagging
8     self.getTag('MySpecialEvent').setInputHeaders( [recHdr] )
9     self.tagIt('MySpecialEvent')
```

Once a tag has been set, it can be used by later analysis algorithms in the current job, or saved to the output file and used at a later time. Here is a Python example of checking the tag:

```
1 # Check tag
2 tag = evt["/Event/UserTag/MySpecialEvent"]
3 if tag:
4     # This event is tagged.  Do something.
5     # ...
```

Tags can also be used to produce filtered data sets, as shown in section 3.4.6.

[4] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tagging/UserTagging/python/UserTagging/UserTag/DetectorTag.py

[5] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tagging/UserTagging/python/UserTagging/UserTag/DetectorTag.py

3.4.3 Add Variables to a NuWa File

A common task is to add a new user-defined variable for each event. For example, the time since the previous trigger can be calculated and added to each event. This is a task for `UserData`.

The example job module Tutorial/Quickstart/python/Quickstart/DtData.py[6] shows the example of adding the time since the previous trigger to each event. This example can be run:

```
shell> nuwa.py -n -1 --no-history -m"Quickstart.DtData"
          -o daqPlus.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

After completion, the output file can be opened in ROOT and the new data variables can be viewed and histogrammed (Fig 3.7.) The file can also be read back into another NuWa job, and the user data will still be accessible.

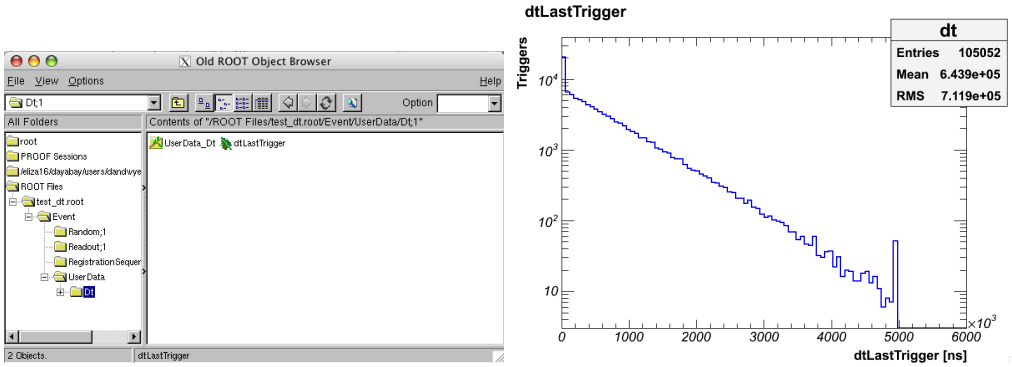

Figure 3.7: Example of browsing and histogramming user-defined data in ROOT.

To add your own variables, copy and modify the module Tutorial/Quickstart/python/Quickstart/DtData.py[7]. See section 3.4.8 for general advice on modifying an existing job module. Currently single integers, single floating-point decimal numbers, and arrays of each can be added as user-defined variables.

3.4.4 Adding User-defined Variables to Tagged Events

The Tagging/UserTagging[8] package provides some convenient tools for simultaneously applying tags and adding user data for those tagged events. Following the example described in section 3.4.2, user data can be added in parallel to an event tag. In the `initTagList` function, you can define user data associated with the tag:

```
myTag = self.addTag('MySpecialEvent' , '/Event/UserTag/MySpecialEvent')
myData = myTag.addData('MySpecialData','/Event/UserData/MySpecialData')
myData.addInt('myInt')
```

In the `check` function, you should set the variable value before calling `tagIt`:

```
if tagThisEvent:
    # Keep track of the reconstructed data you are tagging
    self.getTag('MySpecialEvent').setInputHeaders( [recHdr] )
    myData = self.getTag('MySpecialEvent').getData('MySpecialData')
```

[6] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tutorial/Quickstart/python/Quickstart/DtData.py

[7] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tutorial/Quickstart/python/Quickstart/DtData.py

[8] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Tagging/UserTagging

```
5     myData.set('myInt',12345)
6     self.tagIt('MySpecialEvent')
```

3.4.5 Copy Data Paths to a New File

There may be situations where you would like to filter only some paths of data to a smaller file. The job module `SimpleFilter.Keep` can be used for this purpose. The following example shows how to create an output file which contains only the `AdSimple` reconstructed data:

```
1 shell> nuwa.py -n -1 -m"SimpleFilter.Keep /Event/Rec/AdSimple"
2             -o adSimple.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
3             recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

This module can take multiple arguments to save more paths to the same file:

```
1 shell> nuwa.py -n -1 -m"SimpleFilter.Keep /Event/Rec/AdSimple /Event/Rec/AdQmlf"
2             -o myRecData.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
3             recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

3.4.6 Write Tagged Data to a New File

There may be situations where you would like to filter only some events to a smaller data file. The `SmartFilter` package provides some tools for this purpose. The first step is to define your own tag for the events you wish to keep, as discussed in section 3.4.2. The following example shows how to create an output file which contains only the events you have tagged as `MySpecialEvents`:

```
1 shell> nuwa.py -n -1 -m"MySpecialTagger" -m"SmartFilter.Keep /Event/UserTag/MySpecialEvents"
2             -o mySpecialEvents.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
3             recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

The output file will contain your tag `/Event/UserTag/MySpecialEvents`, plus any data that your tag refers to such as `/Event/Rec/AdSimple`, `/Event/Readout/ReadoutHeader`, etc.

To create more advanced data filters, copy and modify the job module Filtering/SmartFilter/python/SmartFilter/Example.py[9].

3.4.7 Change an Existing Job Module

This section describes how to change an existing module with name *PACKAGE.MODULE*. First copy this Job Module to your local directory. You can locate a module using the environment variable `$ PACKAGE ROOT`,

```
1 shell> mkdir mywork
2 shell> cd mywork
3 shell> cp $<PACKAGE>ROOT/python/<PACKAGE>/<MODULE>.py myModule.py
```

Once you have a copy of the Job Module, open it with your favorite text editor. The module is written in the Python language (http://www.python.org); see the Python website for a good tutorial on this language. Job Modules are composed of two functions: `configure()` and `run()`,

```
1 def configure( argv=[] ):
2     """A description of your module here
3     """
4     # Most job configuration commands here
5     return
6
7 def run(app):
8     """Specific run-time configuration"""
```

[9] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Filtering/SmartFilter/python/SmartFilter/Example.py

```
    # Some specific items must go here (Python algorithms, add libraries, etc.)
    pass
```

For advice on what lines to modify in the module, send your request to the offline software mailing list: `theta13-offline@dayabay.lbl.gov`.

To run your modified version of the module, call it in the `nuwa.py` command without the *PACKAGE.* prefix in the module name. With no prefix, modules from the current directory will be used.

```
shell> ls
myModule.py
shell> nuwa.py -n -1 -m"myModule" recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

3.4.8 Write a Python analysis Algorithm

If you wish to add your own algorithm to NuWa, a good place to start is by writing a prototype algorithm in Python. Writing your algorithm in Python is much easier than C++, and does not require you to compile.

To get started, copy the example template Python algorithm to your local directory:

```
shell> mkdir mywork
shell> cd mywork
shell> cp $QUICKSTARTROOT/python/Quickstart/Template.py myAlg.py
```

Alternatively, you can copy `PrintRawData.py`, `PrintCalibData.py`, or `PrintReconData.py` if you want to specifically process the readout, calibrated, or reconstructed data. Each of these files is a combination of a Python algorithm and a nuwa Python Job Module. To run this module and algorithm, you can call it in the following way:

```
shell> nuwa.py -n -1 -m"myAlg" recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

Inside this file, you can find a Python algorithm. It is a Python class that defines three key functions:

- `initialize()`: Called once at job start
- `execute()`: Called once for each event
- `finalize()`: Called once at job end

You should edit these functions so that the algorithm will do the task you want. There are a few common tasks for algorithms. One is to print to the screen some data from the event:

```
def execute(self):
    evt = self.evtSvc()
    reconHdr = evt["/Event/Rec/RecHeader"]
    print "Energy [MeV] = ", reconHdr.recResult().energy() / units.MeV
```

Another common task is to histogram some data from the event:

```
def initialize(self):
    # Define the histogram
    self.stats["/file1/myhists/energy"] = TH1F("energy",
                                               "Reconstructed energy for each trigger",
                                                100,0,10)

def execute(self):
    evt = self.evtSvc()
    reconHdr = evt["/Event/Rec/RecHeader"]
    if reconHdr.recResult().energyStatus() == ReconStatus.kGood:
        #Fill the histogram
        self.stats["/file1/myhists/energy"].Fill(reconHdr.recResult().energy() / units.MeV)
```

Although these examples are simple, algorithms can perform complex calculations on the data that are not possible directly from ROOT. For cheat-sheets of the data available in NuWa, see the following sections: Readout data [3.5.8], Calibrated hit data [3.5.9], Reconstructed data [3.5.11].

Remember to commit your new algorithm to SVN! The wiki section SVN_Repository#Guidelines[10] provides some tips on committing new software to SVN.

3.4.9 Write a C++ analysis Algorithm

A drawback of using Python algorithms is that they will usually run slower than an algorithm written in C++. If you wish to run your algorithm as part of data production, or if you just want it to run faster, then you should convert it to C++.

Adding a C++ algorithm to Gaudi is a more complex task. The first step is to create your own Project. Your own Project allows you to write and run your own C++ analysis software with NuWa. See section 3.5.3 for how to prepare this.

Once you have your own project, you should prepare your own package for your new algorithm. A tool has been provided to help you with this. The following commands will set up your own package:

```
shell> cd myNuWa
shell> svn export http:/ /dayabay.ihep.ac.cn/svn/dybsvn/people/wangzhe/Start
shell> svn export http:/ /dayabay.ihep.ac.cn/svn/dybsvn/people/wangzhe/ProjRename
shell> ProjRename Start MyNewAlg
shell> ls
MyNewAlg  ProjRename
shell> emacs MyNewAlg/src/components/MyNewAlg.cc &
```

At this point you should edit the empty algorithm in `MyNewAlg/src/components/MyNewAlg.cc`. In particular, you should add your analysis code into the `initialize()`, `execute()`, and `finalize()` functions.

To compile your new algorithm, you should do the following in a new clean shell:

```
shell> pushd NuWa-trunk
shell> source setup.sh
shell> export CMTPROJECTPATH=/path/to/myProjects:${CMTPROJECTPATH}
shell> popd
shell> cd myNuWa/MyNewAlg/cmt
shell> cmt config; cmt make;
```

Now you should setup a separate 'running' shell for you to run and test your new algorithm. Staring with a clean shell, run the following:

```
shell> pushd NuWa-trunk
shell> source setup.sh
shell> export CMTPROJECTPATH=/path/to/myProjects:${CMTPROJECTPATH}
shell> cd dybgaudi/DybRelease/cmt
shell> source setup.sh
shell> popd
shell> pushd myNuWa/MyNewAlg/cmt
shell> source setup.sh; source setup.sh;
```

Now you should be set up and ready to run your new NuWa algorithm in this shell:

```
shell> nuwa.py -n -1 -m"MyNewAlg.run" recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

Remember to commit your new algorithm to SVN!

3.4.10 Modify Part of NuWa

Sometimes you may want to modify an existing part of NuWa and test the changes you have made. First, you must setup your own Project as shown in section 3.5.3.

Next, you should checkout the package into your Project:

[10] https://wiki.bnl.gov/dayabay/index.php?title=SVN_Repository#Guidelines

Table 3.3: Some Common Services

ICableSvc	Electronics cable connection maps and hardware serial numbers
ICalibDataSvc	PMT and RPC calibration parameters
ISimDataSvc	PMT/Electronics input parameters for simulation
IJobInfoSvc	NuWa Job History Information (command line, software version, etc)
IRunDataSvc	DAQ Run information (run number, configuration, etc.)
IPmtGeomInfoSvc	Nominal PMT positions
IStatisticsSvc	Saving user-defined histograms, ntuples, trees, etc. to output files

```
shell> cd myNuWa
shell> svn checkout http:/ /dayabay.ihep.ac.cn/svn/dybsvn/dybgaudi/trunk/Reconstruction/CenterOfChargePos
shell> ls
CenterOfChargePos
shell> emacs CenterOfChargePos/src/components/CenterOfChargePosTool.cc &
```

After you have made your changes, you should compile and test your modifications. To compile the modified package, you should run the following commands in a clean shell:

```
shell> pushd NuWa-trunk
shell> source setup.sh
shell> export CMTPROJECTPATH=/path/to/myProjects:${CMTPROJECTPATH}
shell> popd
shell> cd myNuWa/CenterOfChargePos/cmt
shell> cmt config; cmt make;
```

To make NuWa use your modified package, run the following commands in a new clean shell:

```
shell> pushd NuWa-trunk
shell> source setup.sh
shell> export CMTPROJECTPATH=/path/to/myProjects:${CMTPROJECTPATH}
shell> cd dybgaudi/DybRelease/cmt
shell> source setup.sh
shell> popd
shell> pushd myNuWa/CenterOfChargePos/cmt
shell> source setup.sh; source setup.sh;
```

This shell will now use your modified code instead of the original version in NuWa:

```
shell> nuwa.py -n -1 -m"Quickstart.Calibrate" -m"Quickstart.Reconstruct"
          -o recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

After you have verified that your changes are correct, you can commit your changes:

```
shell> cd CenterOfChargePos
shell> svn diff
(Review the changes you have made.)
shell> svn commit -m"I fixed a bug!"
```

3.4.11 Using Services

Another advantage of using NuWa is that it provides a set of useful *Services*. Services give you access to other data in addition to the event data, such as cable mappings, calibration parameters, geometry information, etc. Services can also provide other useful tasks. Table 3.3 gives lists some common services. Section 3.5.14 gives detailed descriptions of the common services.

Multiple versions of the same service can exists. For example, `StaticCalibDataSvc` loads the PMT calibration parameters from a text table, while `DbiCalibDataSvc` loads the PMT calibration parameters from the database. To access a Service from a Python algorithm, you should load the service in the `initialize()` function:

```
self.calibDataSvc = self.svc('ICalibDataSvc','StaticCalibDataSvc')
if self.calibDataSvc == None:
    self.error("Failed to get ICalibDataSvc: StaticCalibDataSvc")
    return FAILURE
```

When requesting a service, you provide the type of the service (`ICalibDataSvc`) followed by the specific version you wish to use (`StaticCalibDataSvc`).

Loading the service in C++ is similar:

```
ICalibDataSvc* calibDataSvc = svc<ICalibDataSvc>("StaticCalibDataSvc", true);
if( !calibDataSvc ) {
  error() << "Failed to get ICalibDataSvc: StaticCalibDataSvc" << endreq;
  return StatusCode::FAILURE;
}
```

3.5 Cheat Sheets

3.5.1 Loading the NuWa software

On the computer clusters you must load the software each time you log on. You can load the NuWa software using the `nuwaenv` command,

```
shell> nuwaenv -r trunk -O
```

The `nuwaenv` command can incorporate both shared releases and personal projects. For more information on using and configuring nuwaenv see: https://wiki.bnl.gov/dayabay/index.php?title=Environment_Management_with_nuwaenv.

In the end, nuwaenv is a way of automating the sourcing of the following shell commands. The examples given are for the `pdsf` cluster.

```
# bash shell
shell> cd /common/dayabay/releases/NuWa/trunk-opt/NuWa-trunk/
shell> source setup.sh
shell> cd dybgaudi/DybRelease/cmt/
shell> source setup.sh
```

```
# c-shell
shell> cd /common/dayabay/releases/NuWa/trunk-opt/NuWa-trunk/
shell> source setup.csh
shell> cd dybgaudi/DybRelease/cmt/
shell> source setup.csh
```

3.5.2 Installing the NuWa software

For the brave, you can attempt to install NuWa on your own computer. Try the following:

```
shell> mkdir nuwa
shell> cd nuwa
shell> svn export http:/ /dayabay.ihep.ac.cn/svn/dybsvn/installation/trunk/dybinst/dybinst
shell> ./dybinst trunk all
```

If you are very lucky, it will work. Otherwise, send questions to `theta13-offline@dayabay.lbl.gov`. Your chance of success will be much greater if your try to install NuWa on a computer running Scientific Linux or OS X.

3.5.3 Making your own Project

If you want add or modify a part of NuWa, you should create your own Project. This will allow you to create your own packages to add or replace those in NuWa. The first step is to create a subdirectory for your packages in some directory `/path/to/myProjects`:

```
shell> mkdir -p /path/to/myProjects/myNuWa/cmt
```

Create two files under `myNuWa/cmt` with the following content:

```
shell> more project.cmt
project myNuWa

use dybgaudi

build_strategy with_installarea
structure_strategy without_version_directory
setup_strategy root
```

```
shell> more version.cmt
v0
```

Now you can create new packages under the directory `myNuWa/`, and use them in addition to an existing NuWa installation. See section 3.4.9 for more details.

You can also replace an existing NuWa package with you own modified version in `myNuWa/`. See section 3.4.10 for more details.

3.5.4 Standard Data Files

A set of standard Daya Bay data files are available on the computer clusters. The following table provides the location of these files on each cluster:

Type	Location
	Onsite Farm
daq. (.data)	/dyb/spade/rawdata
daq.	??
	PDSF
daq. (.data)	(In HPSS Archive)
daq.	/eliza16/dayabay/nuwaData/*exp,sim*/*dataTag*/daq
calib.	/eliza16/dayabay/nuwaData/*exp,sim*/*dataTag*/calib
recon.	/eliza16/dayabay/nuwaData/*exp,sim*/*dataTag*/recon
coinc.	/eliza16/dayabay/nuwaData/*exp,sim*/*dataTag*/coinc
spall.	/eliza16/dayabay/nuwaData/*exp,sim*/*dataTag*/spall
	IHEP
daq. (.data)	
daq.	
recon.	
coinc.	
spall.	
	BNL
daq. (.data)	
daq.	
recon.	
coinc.	
spall.	

Using the Catalog

A Catalog tool is provided to locate the raw data files. Be sure to load NuWa before running this example (see section 3.5.1). Here is a simple example to locate the raw data files for a run:

```
shell> python
Python 2.7 (r27:82500, Jan  6 2011, 05:00:16)
[GCC 3.4.6 20060404 (Red Hat 3.4.6-8)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import DybPython.Catalog
>>> DybPython.Catalog.runs[8000]
['/eliza16/dayabay/data/exp/dayabay/2011/TestDAQ/NoTag/0430/daq.NoTag.0008000.Physics.EH1-Merged.SFO-1._000
>>> DybPython.Catalog.runs[8001]
['/eliza16/dayabay/data/exp/dayabay/2011/TestDAQ/NoTag/0430/daq.NoTag.0008001.Physics.EH1-Merged.SFO-1._000
>>> DybPython.Catalog.runs[8002]
['/eliza16/dayabay/data/exp/dayabay/2011/TestDAQ/NoTag/0430/daq.NoTag.0008002.Pedestal.EH1-WPI.SFO-1._0001
```

For more information, refer to the Catalog description https://wiki.bnl.gov/dayabay/index.php?title=Accessing_Data_in_a

[11] https://wiki.bnl.gov/dayabay/index.php?title=https://wiki.bnl.gov/dayabay/index.php?title=Accessing_Data_in_a_Warehouse

3.5.5 Data File Contents

The table below lists the known data paths and provides a short description of their contents.

Path	Name	Description
	Real and Simulated Data	
/Event/Readout	ReadoutHeader	Raw data produced by the experiment
/Event/CalibReadout	CalibReadoutHeader	Calibrated times and charges of PMT and RPC hits
/Event/Rec	AdSimple	Toy AD energy and position reconstruction
	AdQmlf	AD Maximum-likelihood light model reconstruction
/Event/Tags		Standard tags for event identification
/Event/Tags/Coinc	ADCoinc	Tagged set of AD time-coincident events
/Event/Tags/Muon	MuonAny	Single muon trigger from any detector
	Muon/FirstMuonTrigger	First trigger from a prompt set of muon triggers
	Retrigger	Possible retriggering due to muon event
/Event/Data	CalibStats	Extra statistics calculated from calibrated data
/Event/Data/Coinc	ADCoinc	Summary data for sets of AD time-coincident events
/Event/Data/Muon	Spallation	Summary data for muon events and subsequent AD events
/Event/UserTags		User-defined event tags
/Event/UserData		User-defined data variables
	Simulated Data Only	
/Event/Gen	GenHeader	True initial position and momenta of simulated particles
/Event/Sim	SimHeader	Simulated track, interactions, and PMT/RPC hits (Geant)
/Event/Elec	ElecHeader	Simulated signals in the electronics system
/Event/Trig	TrigHeader	Simulated signals in the trigger system
/Event/SimReadout	SimHeader	Simulated raw data

3.5.6 Common NuWa Commands

This section provides a list of common `nuwa.py` commands. You must load the NuWa software before you can run these commands (see section 3.5.1).

```
# Wrap raw DAQ files in ROOT tree:
shell> nuwa.py -n -1 -m"ProcessTools.LoadReadout"
          -o daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.data

# Generate Calibration Data
shell> nuwa.py -n -1 -m"Quickstart.Calibrate" -m"Tagger.CalibStats"
          -o calib.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root

# Generate Reconstruction-only data files
shell> nuwa.py -n -1 -A"0.2s" -m"Quickstart.Calibrate" -m"Tagger.CalibStats"
          -m"Quickstart.Reconstruct"
          -m"SmartFilter.Clear" -m"SmartFilter.KeepRecon"
          -o recon.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root

# Generate Spallation-only data files
shell> nuwa.py -n -1 -A"0.2s" -m"Quickstart.Calibrate" -m"Tagger.CalibStats"
          -m"Quickstart.Reconstruct"
          -m"Tagger.MuonTagger.MuonTag" -m"Tagger.MuonTagger.SpallData"
          -m"SimpleFilter.Keep /Event/Data/Muon/Spallation"
          -o spall.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

```
# Generate ADCoincidence-only data files
shell> nuwa.py -n -1 -m"Quickstart.Calibrate" -m"Tagger.CalibStats"
          -m"Quickstart.Reconstruct"
          -m"Tagger.CoincTagger.ADCoincTag" -m"Tagger.CoincTagger.ADCoincData"
          -m"SimpleFilter.Keep /Event/Data/Coinc/AD1CoincData /Event/Data/Coinc/AD2CoincData"
          -o coinc.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

```
# Generate ODM figures
shell> nuwa.py -n -1 --output-stats="{'file1':'odmHistograms.root'}"
          -m"AdBasicFigs.MakeFigs"
          -m"Quickstart.Calibrate" -m"Tagger.CalibStats"
          -m"AdBasicFigs.MakeCalibFigs"
          -m"MuonBasicFigs.MakeCalibFigs"
          -m"Quickstart.Reconstruct"
          -m"AdBasicFigs.MakeReconFigs"
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
```

3.5.7 Conventions and Context

The following sections summarizes the conventions for sites, detectors, and other items used in the analysis software.

Sites

The site ID identifies the site location within the experiment.

Site	C++/Python Name	Number	Description
Unknown	kUnknown	0x00	Undefined Site
Daya Bay	kDayaBay	0x01	Daya Bay Near Hall (EH-1)
Ling Ao	kLingAo	0x02	Ling Ao Near Hall (EH-2)
Far	kFar	0x04	Far Hall (EH-3)
Mid	kMid	0x08	Mid Hall (Doesn't exist)
Aberdeen	kAberdeen	0x10	Aberdeen tunnel
SAB	kSAB	0x20	Surface Assembly Building
PMT Bench Test	kPMTBenchTest	0x40	PMT Bench Test at Dong Guan
All	kAll	(Logical OR of all sites)	All sites

To access the site labels from Python, you can use the commands,

```
from GaudiPython import gbl
gbl.DayaBay.Detector # Access any class in library, then ENUMs are available
Site = gbl.Site
print Site.kDayaBay
```

For C++, the site labels can be accessed,

```
#include "Conventions/Site.h"
std::cout << Site::kDayaBay << std::endl;
```

The Site convention is defined in DataModel/Conventions/Conventions/Site.h[12].

Detectors

The detector ID identifies the detector location within the site.

[12] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/DataModel/Conventions/Conventions/Site.h

Detector	C++/Python Name	Number	Description
Unknown	kUnknown	0	Undefined Detector
AD stand 1	kAD1	1	Anti-neutrino detector on stand #1
AD stand 2	kAD2	2	Anti-neutrino detector on stand #2
AD stand 3	kAD3	3	Anti-neutrino detector on stand #3
AD stand 4	kAD4	4	Anti-neutrino detector on stand #4
Inner water pool	kIWS	5	Inner water pool
Outer water pool	kOWS	6	Outer water pool
RPC	kRPC	7	Complete RPC assembly
All	kAll	8	All detectors

To access the detector labels from Python, you can use the commands,

```
from GaudiPython import gbl
gbl.DayaBay.Detector # Access any class in library, then ENUMs are available
DetectorId = gbl.DetectorId
print DetectorId.kAD1
```

For C++, the detector labels can be accessed,

```
#include "Conventions/DetectorId.h"
std::cout << DetectorId::kAD1 << std::endl;
```

The Detector convention is defined in DataModel/Conventions/Conventions/DetectorId.h[13].

3.5.8 Raw DAQ Data

Conversion from `.data`

The raw DAQ file can be wrapped in a ROOT tree. This allows you to histogram the raw data directly from ROOT, as shown in section 3.2.3. The following command will wrap the data. In addition, ROOT will compress the raw data by almost half the original size. The file still contains the raw binary data; no event data conversion is performed.

```
shell> nuwa.py -n -1 -m"ProcessTools.LoadReadout"
          -o daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
          daq.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.data
```

Raw data in ROOT

The following table summarizes the raw data that is accessible directly from ROOT. All ROOT variables must be preceded by `daqPmtCrate()`..

Item	ROOT Variable	Description
site	detector().site()	Site ID number
detector	detector().detectorId()	Detector ID number
trigger type	triggerType()	All active triggers, logically OR'd
trigger time	triggerTime().GetSeconds()	Complete trigger time [seconds]
TDC time	tdcs(*board*,*connector*,*adcGain*).values()	Channel TDC values
ADC charge	adcs(*board*,*connector*,*adcGain*).values()	Channel ADC values
	gains(*board*,*connector*).values()	Channel ADC Gain (1: Fine ADC, 2: Coarse ADC)
	preAdcRaws(*board*,*connector*,*adcGain*).values()	Channel pre-ADC raw values
	peaks(*board*,*connector*,*adcGain*).values()	Clock cycle (in 25ns) of ADC peak relative to TDC hit

[13] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/DataModel/Conventions/Conventions/DetectorId.h

Readout data in NuWa

Here is a cheat-sheet for processing raw data in Python. These lines can be used in the `execute()` function of a Python algorithm.

```
        evt = self.evtSvc()

        # Access the Readout Header.  This is a container for the readout data
        readoutHdr = evt["/Event/Readout/ReadoutHeader"]
        if readoutHdr == None:
            self.error("Failed to get current readout header")
            return FAILURE

        # Access the Readout.  This is the data from one trigger.
        readout = readoutHdr.daqCrate().asPmtCrate()
        if readout == None:
            self.info("No readout this cycle")
            return SUCCESS

        # Get the detector ID for this trigger
        detector = readout.detector()
        detector.detName()

        # Trigger Type: This is an integer of the type for this trigger
        readout.triggerType()
        # Event Number: A count of the trigger, according to the DAQ
        readout.eventNumber()

        # Trigger Time: Absolute time of trigger for this raw data
        triggerTime = readout.triggerTime()

        # Loop over each channel data in this trigger
        for channel in readout.channelReadouts():
            channelId = channel.channelId()

            # The channel ID contains the detector ID, electronics board number,
            # and the connector number on the board.
            channelId.detName()
            channelId.board()
            channelId.connector()

            # Loop over hits for this channel
            for hitIdx in range( channel.hitCount() ):
                # TDC data for this channel
                #
                # The TDC is an integer count of the time between the time
                # the PMT pulse arrived at the channel, and the time the
                # trigger reads out the data.  Therefore, a larger TDC =
                # earlier time.  One TDC count ~= 1.5625 nanoseconds.
                #
                tdc = channel.tdc( hitIdx )

                # ADC data for this channel
                #
                # The ADC is an integer count of the charge of the PMT
                # pulse.  It is 12 bits (0 to 4095).  There are two ADCs
                # for every PMT channel (High gain and Low gain).  Only
                # the high gain ADC is recorded by default.  If the high
                # gain ADC is saturated (near 4095), then the low gain ADC
                # is recorded instead.
                #
                # For the Mini Dry Run data, one PMT photoelectron makes
                # about 20 high gain ADC counts and about 1 low gain ADC
                # count.  There is an offset (Pedestal) for each ADC of
```

```
60                # ~70 ADC counts (ie. no signal = ~70 ADC, 1 photoelectron
61                # = ~90 ADC, 2 p.e. = ~110 ADC, etc.)
62                #
63                # The ADC peal cycle is a record of the clock cycle which had
64                # the 'peak' ADC.
65                #
66                # ADC Gain: Here is a description of ADC gain for these values
67                #  Unknown = 0
68                #  High = 1
69                #  Low = 2
70                #
71                adc = channel.adc( hitIdx )
72                preAdc = channel.preAdcAvg( hitIdx )
73                peakCycle = channel.peakCycle( hitIdx )
74                isHighGain = channel.isHighGainAdc( hitIdx )
```

3.5.9 Calibrated Data

Calibrated data in ROOT

The following table summarizes the calibrated data visible directly in ROOT. Array items have their length given in the brackets (i.e. *name[length]*). ROOT will automatically draw all entries in the array given the array name. See the ROOT User's Guide for more details on working with Trees, http://root.cern.ch/download/doc/12Trees.pdf.

Item	ROOT Variable	Description
site	site	Site ID number
detector	detector	Detector ID number
event number	eventNumber	Unique ID number for each triggered event in a run
trigger type	triggerType	All active triggers, logically OR'd
trigger time	triggerTimeSec	Trigger time: seconds from Jan. 1970 (unixtime)
	triggerTimeNanoSec	Trigger time: nanoseconds from last second
AD PMT hits	nHitsAD	Number of AD PMT hits
	timeAD[nHitsAD]	Calibrated time [ns] of PMT hit relative to trigger time
	chargeAD[nHitsAD]	Calibrated charge [photoelectrons] of PMT hit
	hitCountAD[nHitsAD]	Index of this hit for this PMT (0, 1, 2, ...)
	ring[nHitsAD]	PMT ring in AD (counts 1 to 8 from AD bottom)
	column[nHitsAD]	PMT column in AD (counts 1 to 24 counterclockwise)
Calib. PMT hits	nHitsAD_calib	Number of AD calibration PMT (2-inch) hits
	timeAD_calib[nHitsAD_calib]	Calibrated time [ns] of PMT hit relative to trigger time
	chargeAD_calib[nHitsAD_calib]	Calibrated charge [photoelectrons] of PMT hit
	hitCountAD_calib[nHitsAD_calib]	Index of this hit for this PMT (0, 1, 2, ...)
	topOrBottom[nHitsAD_calib]	PMT vertical position (1: AD top, 2: AD bottom)
	acuColumn[nHitsAD_calib]	PMT radial position (ACU axis: A=1, B=2, C=3)
Water Pool PMT hits	nHitsPool	Number of Water Pool PMT hits
	timePool[nHitsPool]	Calibrated time [ns] of PMT hit relative to trigger time
	chargePool[nHitsPool]	Calibrated charge [photoelectrons] of PMT hit
	hitCountPool[nHitsPool]	Index of this hit for this PMT (0, 1, 2, ...)
	wallNumber[nHitsPool]	PMT wall number
	wallSpot[nHitsPool]	PMT spot number in wall
	inwardFacing[nHitsPool]	PMT direction (0: outward, 1: inward)

Calibrated data in NuWa

Here is a cheat-sheet for processing calibrated data in Python. These lines can be used in the `execute()` function of a Python algorithm.

```
        evt = self.evtSvc()

        # Access the Calib Readout Header.
        # This is a container for calibrated data
        calibHdr = evt["/Event/CalibReadout/CalibReadoutHeader"]
        if calibHdr == None:
            self.error("Failed to get current calib readout header")
            return FAILURE

        # Access the Readout.  This is the calibrated data from one trigger.
        calibReadout = calibHdr.calibReadout()
        if calibReadout == None:
            self.error("Failed to get calibrated readout from header")
            return FAILURE

        # Get the detector ID for this trigger
        detector = calibReadout.detector()
        detector.detName()

        # Trigger Type: This is an integer of the type for this trigger
        calibReadout.triggerType()
        # Trigger Number: A count of the trigger, according to the DAQ
        calibReadout.triggerNumber()

        # Trigger Time: Absolute time of trigger for this calibrated data
        triggerTime = calibReadout.triggerTime()

        # Loop over each channel data in this trigger
        for channel in calibReadout.channelReadout():
            sensorId = channel.pmtSensorId()
            if detector.isAD():
               pmtId = AdPmtSensor( sensorId.fullPackedData() )
               pmtId.detName()
               pmtId.ring()
               pmtId.column()
            elif detector.isWaterShield():
               pmtId = PoolPmtSensor( sensorId.fullPackedData() )
               pmtId.detName()
               pmtId.wallNumber()
               pmtId.wallSpot()
               pmtId.inwardFacing()

            # Calibrated hit data for this channel
            for hitIdx in range( channel.size() ):
                # Hit time is in units of ns, and is relative to trigger time
                hitTime = channel.time( hitIdx )
                # Hit charge is in units of photoelectrons
                hitCharge = channel.charge( hitIdx )
```

3.5.10 Calibrated Statistics Data

Calibrated statistics data in ROOT

The following table summarizes the calibrated statistics data for each event visible directly in ROOT. Array items have their length given in the brackets (i.e. *name[length]*). ROOT will automatically draw all entries in the array given the array name. See the ROOT User's Guide for more details on working with Trees, http://root.cern.ch/download/doc/12Trees.pdf.

ROOT Variable	Description
dtLastAD1_ms	Time since previous AD1 trigger [ms]
dtLastAD2_ms	Time since previous AD2 trigger [ms]
dtLastIWS_ms	Time since previous Inner water pool trigger [ms]
dtLastOWS_ms	Time since previous Outer water pool trigger [ms]
dtLast_ADMuon_ms	Time since previous AD event with greater than 20 MeV [ms]
dtLast_ADShower_ms	Time since previous AD event with greater than 1 GeV [ms]
ELast_ADShower_pe	Energy of last AD event with greater than 1 GeV [pe]
nHit	Total number of hit 8-inch PMTS
nPEMedian	Median charge (number of photoelectrons) on PMTs
nPERMS	RMS of charge (number of photoelectrons) on PMTs
nPESum	Total sum of charge (number of photoelectrons) on all PMTs
nPulseMedian	Median number of hits on PMTs
nPulseRMS	Median number of hits on PMTs
nPulseSum	Total Sum of number of hits on all PMTs
tEarliest	Earliest hit time on all PMTs [ns]
tLatest	Latest hit time on all PMTS [ns]
tMean	Mean hit time on all PMTS [ns]
tMedian	Median hit time on all PMTS [ns]
tRMS	RMS of hit time on all PMTS [ns]
charge_sum_flasher_max	The maxima total charge collected for one PMT in one readout [PE] (sum over all
time_PSD	For hits in each AD, for time window between -1650 and -1250 ns, $\frac{Nhit_{-1650,-1450}}{Nhit_{-1650,-1250}}$.
time_PSD1	For hits in each AD, for time window between -1650 and -1250 ns, $\frac{Nhit_{-1650,-1500}}{Nhit_{-1650,-1250}}$.
time_PSD_local_RMS	The RMS of the time of the first hit (also must be within -1650 and -1250) for 5x5
Q1	The total charge (within -1650 and -1250) of nearby ± 3 columns PMTs (total 7 co
Q2	The total charge (within -1650 and -1250) of $4 \rightarrow 9$ and $-4 \rightarrow -9$ columns PMTs (
Q3	The total charge (within -1650 and -1250) of PMTs for the rest of columns (other t
flasher_flag	"1-time_PSD + 1- time_PSD1 + Q3/Q2*2 + nPEMax/nPESum + time_PSD_local
EarlyCharge	The charge sum in time window t¡-1650ns
LateCharge	The charge sum in time window t¿-1250ns
NominalCharge	The charge sum in time window -1650ns¡t¡-1250ns, See Doc6926
MaxQ	The largest charge fraction of PMTs
maxqRing	The ring number of the MaxQ PMT
maxqCol	The column number of the MaxQ PMT
QuadrantQ1	Total charge of PMTs with column number in [maxqCol-2, maxqCol+3]). For the v
QuadrantQ2	Total charge of PMTs with column number in [(maxqCol+6)-2,(maxqCol+6)+3])
QuadrantQ3	Total Charge of PMTs with column number in [(maxq+12)-2, (maxqCol+12)+3])
QuadrantQ4	Total Charge of PMTs with column number in [(maxq+18)-2, (maxqCol+18)+3])
Quadrant	The ratio of QuadrantQ3/(QuadrantQ2 + QuadrantQ4)
MainPeakRMS	According to the location of MaxQ PMT, divide 24 columns into two clusters. Mai
SecondPeakRMS	See description in MainPeakRMS.
PeakRMS	The sum of MainPeakRMS and SecondPeakRMS
RingKurtosis	Kurtosis of charge weighted distance in the Ring dimension for the MainPeak clust
ColumnKurtosis	Kurtosis of charge weighted distance in the Column dimension for the MainPeak cl
Kurtosis	Sum of RingKurtosis and ColumnKurtosis
MiddleTimeRMS	RMS of PMT first hit time in the time window (-1650ns, -1250ns). This time windo
integralRunTime_ms	'DAQ Running time' from the start of the file up to the current trigger
integralLiveTime_buffer_full_ms	'DAQ Livetime' from the start of the file up to the current trigger. The 'DAQ Live
integralLiveTime_blocked_trigger_ms	'DAQ Livetime', using an alternate correction for 'blocked trigger' periods
blocked_trigger	A count of the 'blocked triggers' immediately preceding the current trigger. When
buffer_full_flag	This flag is true if the electronics memory buffers filled immediately preceding this

Calibrated statistics data in NuWa

Here is a cheat-sheet for processing calibrated statistics data in Python. These lines can be used in the `execute()` function of a Python algorithm.

```
        evt = self.evtSvc()

        # Access the Calibrated Statistics Data Header.
        # This is a container for calibrated statistics data
        calibStats = evt["/Event/Data/CalibStats"]
        if calibStats == None:
            self.debug("No calibrated statistics!")
            return FAILURE

        # Access the Calibrated statistics data
        nPESum = calibStats.get('nPESum').value()
```

3.5.11 Reconstructed Data

Reconstructed data in ROOT

The following table summarizes the reconstructed data visible directly in ROOT. Reconstruction can optionally estimate an energy, a position, and/or a track direction. The `status` variables should be checked to determine whether reconstruction has successfully set any of these quantities.

Item	ROOT Variable	Description
site	site	Site ID number
detector	detector	Detector ID number
trigger type	triggerType	All active triggers, logically added
trigger time	triggerTimeSec	Trigger time count in seconds from Jan. 1970 (unixtime)
	triggerTimeNanoSec	Trigger time count of nanoseconds from last second
energy	energyStatus	Status of energy reconstruction (0: unknown, 1: good, >1: failures)
	energy	reconstructed energy [MeV]
	energyQuality	Measure of fit quality (χ^2, likelihood, etc.)
position	positionStatus	Status of position reconstruction (0: unknown, 1: good, >1: failures)
	x	reconstructed x position [mm] in AD, Water Pool, or RPC coordinates
	y	reconstructed y position [mm] in AD, Water Pool, or RPC coordinates
	z	reconstructed z position [mm] in AD, Water Pool, or RPC coordinates
	positionQuality	Measure of fit quality (χ^2, likelihood, etc.)
direction	directionStatus	Status of track reconstruction (0: unknown, 1: good, >1: failures)
	dx	reconstructed dx track direction in AD, Water Pool, or RPC coordinates
	dy	reconstructed dy track direction in AD, Water Pool, or RPC coordinates
	dz	reconstructed dz track direction in AD, Water Pool, or RPC coordinates
	directionQuality	Measure of fit quality (χ^2, likelihood, etc.)
error matrix	errorMatrixDim	Dimension of error matrix (0 if not set)
	errorMatrix	Array of error matrix elements

Reconstructed data in NuWa

Here is a cheat-sheet for processing reconstructed data in Python. These lines can be used in the `execute()` function of a Python algorithm.

```
        evt = self.evtSvc()

        # Access the Recon Header.  This is a container for the reconstructed data
        reconHdr = evt["/Event/Rec/AdSimple"]
        if reconHdr == None:
            self.error("Failed to get current recon header")
```

```
7             return FAILURE
8
9         result = reconHdr.recTrigger()
10
11        # Get the detector ID for this trigger
12        detector = result.detector()
13        detector.detName()
14
15        # Trigger Type: This is an integer of the type for this trigger
16        result.triggerType()
17        # Trigger Number: A count of the trigger, according to the DAQ
18        result.triggerNumber()
19
20        # Trigger Time: Absolute time of trigger for this raw data
21        triggerTime = result.triggerTime()
22
23        # Energy information
24        result.energyStatus()
25        result.energy()
26        result.energyQuality()
27
28        # Position information
29        result.positionStatus()
30        result.position().x()
31        result.position().y()
32        result.position().z()
33        result.positionQuality()
34
35        # Direction information, for tracks
36        result.directionStatus()
37        result.direction().x()
38        result.direction().y()
39        result.direction().z()
40        result.directionQuality()
41
42        # Covariance Matrix, if one is generated
43        result.errorMatrix()
```

3.5.12 Spallation Data

Spallation data in ROOT

The following table summarizes the spallation data visible directly in ROOT. Array items have their length given in the brackets (i.e. *name[length]*). ROOT will automatically draw all entries in the array given the array name. See the ROOT User's Guide for more details on working with Trees, http://root.cern.ch/download/doc/12Trees.pdf.

ROOT Variable	Description
tMu_s	Timestamp of this muon event (seconds part)
tMu_ns	Timestamp of this muon event (nanoseconds part)
dtLastMu_ms	Time since previous muon event [ms]
dtNextMu_ms	Time to next muon event [ms]
hitAD1	Did AD1 have a prompt trigger for this muon?
hitAD2	Did AD2 have a prompt trigger for this muon?
hitAD3	Did AD3 have a prompt trigger for this muon?
hitAD4	Did AD4 have a prompt trigger for this muon?
hitIWS	Did the Inner water pool have a prompt trigger for this muon?
hitOWS	Did the Outer water pool have a prompt trigger for this muon?
hitRPC	Did the RPC have a prompt trigger for this muon?
triggerNumber_AD1	Trigger number of prompt AD1 muon trigger (if exists)
triggerNumber_AD2	Trigger number of prompt AD2 muon trigger (if exists)
triggerNumber_AD3	Trigger number of prompt AD3 muon trigger (if exists)
triggerNumber_AD4	Trigger number of prompt AD4 muon trigger (if exists)
triggerNumber_IWS	Trigger number of prompt IWS muon trigger (if exists)
triggerNumber_OWS	Trigger number of prompt OWS muon trigger (if exists)
triggerNumber_RPC	Trigger number of prompt RPC muon trigger (if exists)
triggerType_AD1	Trigger type of prompt AD1 muon trigger (if exists)
triggerType_AD2	Trigger type of prompt AD2 muon trigger (if exists)
triggerType_AD3	Trigger type of prompt AD3 muon trigger (if exists)
triggerType_AD4	Trigger type of prompt AD4 muon trigger (if exists)
triggerType_IWS	Trigger type of prompt IWS muon trigger (if exists)
triggerType_OWS	Trigger type of prompt IWS muon trigger (if exists)
triggerType_RPC	Trigger type of prompt IWS muon trigger (if exists)
dtAD1_ms	Time since first prompt muon trigger [ms]
dtAD2_ms	Time since first prompt muon trigger [ms]
dtAD3_ms	Time since first prompt muon trigger [ms]
dtAD4_ms	Time since first prompt muon trigger [ms]
dtIWS_ms	Time since first prompt muon trigger [ms]
dtOWS_ms	Time since first prompt muon trigger [ms]
dtRPC_ms	Time since first prompt muon trigger [ms]
calib_nPESum_AD1	CalibStats charge sum from prompt muon trigger
calib_nPESum_AD2	CalibStats charge sum from prompt muon trigger
calib_nPESum_AD3	CalibStats charge sum from prompt muon trigger
calib_nPESum_AD4	CalibStats charge sum from prompt muon trigger
calib_nPESum_IWS	CalibStats charge sum from prompt muon trigger
calib_nPESum_OWS	CalibStats charge sum from prompt muon trigger
nRetriggers	Total number of possible retriggers
detectorId_rt[nRetriggers]	Possible retrigger detector ID
dtRetrigger_ms[nRetriggers]	Time of retrigger relative to first prompt muon trigger
triggerNumber_rt[nRetriggers]	Trigger number of retrigger
triggerType_rt[nRetriggers]	Trigger type of retrigger
calib_nPESum_rt[nRetriggers]	Total charge sum of retrigger
nSpall	Number of AD triggers between this muon and next muon
detectorId_sp[nSpall]	Detector ID of AD trigger
triggerNumber_sp[nSpall]	Trigger number of AD trigger
triggerType_sp[nSpall]	Trigger type of AD trigger
dtSpall_ms[nSpall]	Time between AD trigger and first prompt muon trigger [ms]
energyStatus_sp[nSpall]	AD energy reconstruction status
energy_sp[nSpall]	AD reconstructed energy [MeV]
positionStatus_sp[nSpall]	AD position reconstruction status
x_sp[nSpall]	AD reconstructed X position [mm]
y_sp[nSpall]	AD reconstructed Y position [mm]
z_sp[nSpall]	AD reconstructed Z position [mm]

Spallation data in NuWa

Here is a cheat-sheet for processing spallation data in Python. These lines can be used in the `execute()` function of a Python algorithm.

```
        evt = self.evtSvc()

        # Access the Spallation Data Header.
        # This is a container for muon spallation data
        spallData = evt["/Event/Data/Muon/Spallation"]
        if spallData == None:
            self.debug("No spallation data this cycle")
            return SUCCESS

        # Access the spallation data
        nSpall = spall.get('nSpall').value()
```

3.5.13 Coincidence Data

Coincidence data in ROOT

The following table summarizes the coincidence data visible directly in ROOT. Array items have their length given in the brackets (i.e. *name[length]*). ROOT will automatically draw all entries in the array given the array name. See the ROOT User's Guide for more details on working with Trees, http://root.cern.ch/download/doc/12Trees.pdf.

ROOT Variable	Description
multiplicity	Number of AD events within coincidence window
triggerNumber[multiplicity]	Trigger number of event
triggerType[multiplicity]	Trigger type of event
t_s[multiplicity]	Timestamp of event (seconds part)
t_ns[multiplicity]	Timestamp of event (nanoseconds part)
dt_ns[multiplicity]	Time relative to first event in multiplet
energyStatus[multiplicity]	Status of AD energy reconstruction
e[multiplicity]	Reconstructed energy [MeV]
positionStatus[multiplicity]	Status of AD position reconstruction
x[multiplicity]	AD Reconstructed X position [mm]
y[multiplicity]	AD Reconstructed Y position [mm]
z[multiplicity]	AD Reconstructed Z position [mm]
I[mult*(mult-1)/2]	Prompt helper array for ROOT histogramming
J[mult*(mult-1)/2]	Delayed helper array for ROOT histogramming
dtLastAD1_ms[multiplicity]	Time since last muon in AD1 [ms]
dtLastAD2_ms[multiplicity]	Time since last muon in AD2 [ms]
dtLastIWS_ms[multiplicity]	Time since last muon in Inner water pool [ms]
dtLastOWS_ms[multiplicity]	Time since last muon in Outer water pool [ms]
dtLast_ADMuon_ms	Time since previous AD event above 3200 pe (20 MeV) [ms]
dtLast_ADShower_ms	Time since previous AD event above 160000 pe (1 GeV) [ms]
ELast_ADShower_pe	Energy of last AD event with greater than 160000 pe [pe]
calib_nHit[multiplicity]	CalibStats data
calib_nPEMedian[multiplicity]	CalibStats data
calib_nPERMS[multiplicity]	CalibStats data
calib_nPESum[multiplicity]	CalibStats data
calib_nPulseMedian[multiplicity]	CalibStats data
calib_nPulseRMS[multiplicity]	CalibStats data
calib_nPulseSum[multiplicity]	CalibStats data
calib_tEarliest[multiplicity]	CalibStats data
calib_tLatest[multiplicity]	CalibStats data
calib_tMean[multiplicity]	CalibStats data
calib_tMedian[multiplicity]	CalibStats data
calib_tRMS[multiplicity]	CalibStats data
gen_count[multiplicity]	Monte-Carlo truth generator data
gen_e[multiplicity]	Monte-Carlo truth generator data
gen_execNumber[multiplicity]	Monte-Carlo truth generator data
gen_lastDaughterPid[multiplicity]	Monte-Carlo truth generator data
gen_pid[multiplicity]	Monte-Carlo truth generator data
gen_px[multiplicity]	Monte-Carlo truth generator data
gen_py[multiplicity]	Monte-Carlo truth generator data
gen_pz[multiplicity]	Monte-Carlo truth generator data
gen_type[multiplicity]	Monte-Carlo truth generator data

Coincidence data in NuWa

Here is a cheat-sheet for processing coincidence data in Python. These lines can be used in the `execute()` function of a Python algorithm.

```
1         evt = self.evtSvc()
2
3         # Access the Coincidence Data Header.
```

```
4         # This is a container for AD coincidence data
5         coincHdr = evt["/Event/Data/Coinc/AD1Coinc"]
6         if coincHdr == None:
7             self.debug("No coincidence header this cycle")
8             return SUCCESS
9
10        # Access the Coincidence Data
11        dt_ms = coinc.get('dt_ms').value()
```

3.5.14 NuWa Services

(Add documentation for common services here.)

3.5.15 Computer Clusters

(Add details for each computer cluster here.)

3.5.16 Miscellaneous

Time Axes in ROOT

The following lines will display a time axis in a human-readable format using Beijing local time.

```
1root [3] htemp->GetXaxis()->SetTimeDisplay(1);
2root [4] htemp->GetXaxis()->SetTimeFormat("#splitline{%H:%M:%S}{%d\/%m\/%Y}");
3root [5] htemp->GetXaxis()->SetNdivisions(505);
4root [6] htemp->GetXaxis()->SetTimeOffset(8*60*60);
5root [7] htemp->Draw("colz");
```

3.6 Hands-on Exercises

- Find the AD Dry Run data files from run 5773 on PDSF.
- Convert the first file of this run from `.data` to `.root`.
- Generate a calibrated data file from this data.
- Plot the AD charge map figures shown in Fig. 3.4
- Generate a reconstructed data file from this data.
- Plot the calibrated AD charge sum vs. the AD reconstructed energy.
- From the first simulation file from run 29000, generate a spallation file and plot the time from each AD event to the last muon.
- From the first simulation file from run 29000, generate an AD coincidence file and plot the prompt vs. delayed reconstructed energy.

Chapter 4

Offline Infrastructure

4.1 Mailing lists

- existing lists, their purposes
- offline list - expected topics
- subscribing
- archives
- how to get help

4.2 DocDB

- Content - what should go in DocDB
- how to access
- Major features
- Basic instructions
- how to get help

4.3 Wikis

- Content - what should go in DocDB
- How to access
- Basic markup help
- Conventions, types of topics
- Using categories

4.4 Trac bug tracker

- when to use it
- roles and responsibilities

Chapter 5

Installation and Working with the Source Code

5.1 Using pre-installed release

All major clusters should have existing releases installed and ready to use. Specific information on different clusters is available in the wiki topic "Cluster Account Setup" [1]. The key piece of information to know is where the release is installed.

Configuring your environment to use an installed release progresses through several steps.

5.1.1 Basic setup

Move to the top level release directory and source the main setup script.

```
shell> cd /path/to/NuWa-RELEASE
bash> source setup.sh
tcsh> source setup.csh
```

Replace "`RELEASE`" with "`trunk`" or the release label of a frozen release.

5.1.2 Setup the `dybgaudi` project

Projects are described more below. To set up your environment to use our software project, "`dybgaudi`" and the other projects on which it depends to must enter a, so called, "release package" and source its setup script.

```
shell> cd /path/to/NuWa-RELEASE
bash> source setup.sh
tcsh> source setup.csh
```

You are now ready to run some software. Try:

```
shell> cd $HOME
shell> nuwa.py --help
```

[1] https://wiki.bnl.gov/dayabay/index.php?title=Cluster_Account_Setup

5.2 Instalation of a Release

If you work on a cluster, it is best to use a previously existing release. If you do want to install your own copy it is time and disk consuming but relatively easy. A script called "`dybinst`" takes care of everything.

First, you must download the script. It is best to get a fresh copy whenever you start an installation. The following examples show how to install the "trunk" branch which holds the most recent development.

```
shell> svn export http://dayabay.ihep.ac.cn/svn/dybsvn/installation/trunk/dybinst/dybinst
```

Now, let it do its work:

```
shell> ./dybinst trunk all
```

Expect it to take about 3-4 hours depending on your computer's disk, CPU and network speed. It will also use several GBs of storage, some of which can be reclaimed when the install is over.

5.3 Anatomy of a Release

`external/` holds 3^{rd} party binary libraries and header files under `PACKAGE/VERSION/` sub directories.

`NuWa-RELEASE/` holds the projects and their packages that make up a release.

`lcgcmt` build information for using 3^{rd} party external packages

`gaudi` the Gaudi framework

`lhcb` packages adopted from the LHCb experiment

`dybgaudi` packages specific to Daya Bay offline software

`relax` packages providing dictionaries for CLHEP and other HEP libraries.

5.3.1 Release, Projects and Packages

- What is a release. For now see https://wiki.bnl.gov/dayabay/index.php?title=Category:Offline_Software_Releases
- What is a package. For now see https://wiki.bnl.gov/dayabay/index.php?title=CMT_Packages
- What is a project. For now see https://wiki.bnl.gov/dayabay/index.php?title=CMT_Projects.

5.3.2 Personal Projects

- Using a personal project with projects from a NuWa release.
- CMTPROJECTPATH

For now see https://wiki.bnl.gov/dayabay/index.php?title=CMT_Projects.

5.4 Version Control Your Code

5.4.1 Using SVN to Contribute to a Release

5.4.2 Using GIT with SVN

Advanced developers may consider using `git`[2] to interface with the SVN repository. Reasons to do this include being able to queue commits, advanced branching and merging, sharing code with other git users or with yourself on other computers with the need to commit to SVN. In particular, git is used to track the projects (gaudi, etc) while retaining the changes Daya Bay makes. For more information see https://wiki.bnl.gov/dayabay/index.php?title=Synchronizing_Repositories.

5.5 Technical Details of the Installation

5.5.1 LCGCMT

The LCGCMT package is for defining platform tags, basic CMT macros, building external packages and "glueing" them into CMT.

Builders

The builders are CMT packages that handle downloading, configuring, compiling and installing external packages in a consistent manner. They are used by `dybinst` or can be run directly. For details see the `README.org` file under `lcgcmt/LCG_builders/` directory.

Some details are given for specific builders:

data: A select sampling of data files are installed under the "data" external package. These are intended for input to unit tests or for files that are needed as input but are too large to be conveniently placed in SVN. For the conventions that must be followed to add new files see the comments in the `data/cmt/requirements/` file under the builder area.

[2] http://git.or.cz/

Chapter 6

Offline Framework

6.1 Introduction

When writing software it is important to manage complexity. One way to do that is to organize the software based on functionality that is generic to many specific, although maybe similar applications. The goal is to develop software which "does everything" except those specific things that make the application unique. If done well, this allows unique applications to be implemented quickly, and in a way that is robust against future development but still flexible to allow the application to be taken in novel directions.

This can be contrasted with the inverted design of a toolkit. Here one focuses on units of functionality with no initial regards of integration. One builds libraries of functions or objects that solve small parts of the whole design and, after they are developed, find ways to glue them all together. This is a useful design, particularly when there are ways to glue disparate toolkits together, but can lead to redundant development and inter-operational problems.

Finally there is the middle ground where a single, monolithic application is built from the ground up. When unforeseen requirements are found their solution is bolted on in whatever the most expedient way can be found. This can be useful for quick initial results but eventually will not be maintainable without growing levels of effort.

6.2 Framework Components and Interfaces

Gaudi components are special classes that can be used by other code without explicitly compiling against them. They can do this because they inherit from and implement one or more special classes called "interface classes" or just interfaces. These are light weight and your code compiles against them. Which actual implementation that is used is determined at run time by looking them up by name. Gaudi Interfaces are special for a few reasons:

Pure-virtual: all methods are declared `=0` so that implementations are required to provide them. This is the definition of an "interface class". Being pure-virtual also allows for an implementation class to inherit from multiple interfaces without problem.

References counted: all interfaces must implement reference counting memory management.

ID number: all interface implementations must have a unique identifying number.

Fast casting: all interfaces must implement the fast `queryInterface()` dynamic cast mechanism.

Part of a components implementation involves registering a "factory" class with Gaudi that knows how to produce instances of the component given the name of the class. This registration happens when the

component library is linked and this linking can be done dynamically given the class name and the magic of generated `rootmap` files.

As a result, C++ (or Python) code can request a component (or Python shadow class) given its class name. At the same time as the request, the resulting instance is registered with Gaudi using a nick-name[1]. This nick-name lets you configure multiple instances of one component class in different ways. For example one might want to have a job with two competing instances of the same algorithm class run on the same data but configured with two different sets of properties.

6.3 Common types of Components

The main three types of Gaudi components are Algorithms, Tools and Services.

6.3.1 Algorithms

- Inherit from `GaudiAlgorithm` or if you will produce data from `DybAlgorithm`.
- execute(), initialize(), finalize() and associated requirements (eg. calling GaudiAlgorithm::initialize()).
- TES access with `get()` and `put()` or `getTes()` and `putTES` if implementing `DybAlgorithm`. There is also `getAES` to access the archive event store.
- Logging with `info()`, etc.
- required boilerplate (_entries & _load files, cpp macros)
- some special ones: sequencer (others?)

Algorithms contain code that should be run once per execution cycle. They may take input from the TES and may produce output. They are meant to encapsulate complexity in a way that allows them to be combined in a high-level manner. They can be combined in a serial chain to run one-by-one or they can run other algorithms as sub-algorithms. It is also possible to set up high-level branch decisions that govern whether or not sub-chains run.

6.3.2 Tools

Tools contain utility code or parts of algorithm code that can be shared. Tool instances can be public, in which case any other code may use it, or they may be private. Multiple instances of a private tool may be created. A tool may be created at any time during a job and will be deleted once no other code references it.

6.3.3 Services

Service is very much like a public tool of which there is a single instance created. Services are meant to be created at the beginning of the job and live for its entire life. They typically manage major parts of the framework or some external service (such as a database).

6.4 Writing your own component

6.4.1 Algorithms

One of the primary goals of Gaudi is to provide the concept of an Algorithm which is the main entry point for user code. All other parts of the framework exist to allow users to focus on writing algorithms.

An algorithm provide three places for users to add their own code:

[1] Nick-names default to the class name.

`initialize()` This method is called once, at the beginning of the job. It is optional but can be used to apply any properties that the algorithm supports or to look up and cache pointers to services, tools or other components or any other initializations that require the Gaudi framework.

`execute()` This method is called once every execution cycle ("event"). Here is where user code does implements whatever algorithm the user creates.

`finalize()` This method is called once, at the end of the job. It is optional but can be used to `release()` any cached pointers to services or tools, or do any other cleaning up that requires the Gaudi framework.

When writing an algorithm class the user has three possible classes to use as a basis:

`Algorithm` is a low level class that does not provide many useful features and is probably best to ignore.

`GaudiAlgorithm` inherits from Algorithm and provide many useful general features such as access to the message service via `info()` and related methods as well as methods providing easy access to the TES and TDS (eg, `get()` and `getDet()`). This is a good choice for many types of algorithms.

`DybAlgorithm` inherits from GaudiAlgorithm and adds Daya Bay specific features related to producing objects from the DataModel. It should only be considered for algorithms that need to add new data to the TES. An algorithm may be based on GaudiAlgorithm and still add data to the TES but some object bookkeeping will need to be done manually.

Subclasses of `DybAlgorithm` should provide `initialize`, `execute` and `finalize` methods as they would if they use the other two algorithm base classes. `DybAlgorithm` is templated by the DataModel data type that it will produce and this type is specified when a subclass inherits from it. Instances of the object should be created using the `MakeHeaderObject()` method. Any input objects that are needed should be retrieved from the data store using `getTES()` or `getAES()`. Finally, the resulting data object is automatically put into the TES at the location specified by the "`Location`" property which defaults to that specified by the DataModel class being used. This will assure bookkeeping such as the list of input headers, the random state and other things are properly set.

6.4.2 Tools

- examples
- Implementing existing tool interface,
- writing new interface.
- required boilerplate (_entries & _load files, cpp macros)

6.4.3 Services

- common ones provided, how to access in C++
- Implementing existing service interface,
- writing new interface.
- Include difference between tools and service.
- required boilerplate (_entries & _load files, cpp macros)

6.4.4 Generalized Components

6.5 Properties and Configuration

Just about every component that Gaudi provides, or those that Daya Bay programmers will write, one or more *properties*. A property has a name and a value and is associated with a component. Users can set properties that will then get applied by the framework to the component.

Gaudi has two main ways of setting such configuration. Initially a text based C++-like language was used. Daya Bay does not use this but instead uses the more modern Python based configuration. With this, it is possible to write a main Python program to configure everything and start the Gaudi main loop to run some number of executions of the top-level algorithm chain.

The configuration mechanism described below was introduced after release 0.5.0.

6.5.1 Overview of configuration mechanism

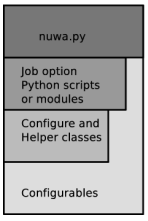

Figure 6.1: Cartoon of the layers of configuration code.

The configuration mechanism is a layer of Python code. As one goes up the layer one goes from basic Gaudi configuration up to user interaction. The layers are pictured in Fig. 6.1. The four layers are described from lowest to highest in the next sections.

6.5.2 Configurables

All higher layers may make use of Configurables. They are Python classes that are automatically generated for all components (Algorithms, Tools, Services, etc). They hold all the properties that the component defines and include their default values and any documentation strings. They are named the same as the component that they represent and are available in Python using this pattern:

```
from PackageName.PackageNameConf import MyComponent
mc = MyComponent()
mc.SomeProperty = 42
```

You can find out what properties any component has using the `properties.py` script which should be installed in your `PATH`.

```
shell> properties.py
GtGenerator :
```

```
3         GenName:  Name of this generator for book keeping purposes.
4         GenTools:  Tools to generate HepMC::GenEvents
5         GlobalTimeOffset: None
6         Location:  TES path location for the HeaderObject this algorithm produces.
7         ...
```

A special configurable is the `ApplicationMgr`. Most users will need to use this to include their algorithms into the "`TopAlg`" list. Here is an example:

```
1
2from Gaudi.Configuration import ApplicationMgr
3theApp = ApplicationMgr()
4
5from MyPackage.MyPackageConf import MyAlgorithm
6ma = MyAlgorithm()
7ma.SomeProperty = "harder, faster, stronger"
8theApp.TopAlg.append(ma)
```

Configurables and Their Names

It is important to understand how configurables eventually pass properties to instantiated C++ objects. Behind the scenes, Gaudi maintains a catalog that maps a key name to a set of properties. Normally, no special attention need be given to the name. If none is given, the configurable will take a name based on its class:

```
1# gets name 'MyAlgorithm'
2generic = MyAlgorithm()
3# gets name 'alg1'
4specific = MyAlgorithm('alg1')
5
6theApp.TopAlg.append(generic)
7theApp.TopAlg.append(specific)
8# TopAlg now holds ['MyAlgorithm/MyAlgorithm', 'MyAlgorithm/alg1']
```

Naming Gaudi Tool Configurables

In the case of Gaudi Tools, things become more complex. Tools themselves can (and should) be configured through configurables. But, there are a few things to be aware of or else one can become easily tricked:

- Tool configurables can be public or private. A public tool configurable is "owned" by ToolSvc and shared by all parents, a private one is "owned" by a single parent and not shared.
- By default, a tool configurable is public.
- "Ownership" is indicated by prepending the parent's name, plus a dot (".") to the a simple name.
- Ownership is set, either when creating the tool configurable by prepending the parent's name, or during assignment of it to the parent configurable.
- During assignment to the parent **a copy** will be made if the tool configurable name is not consistent with the parent name plus a dot prepended to a simple name.

What this means is that you may end up with different final configurations depending on:

- the initial name you give the tool configurable
- when you assign it to the parent
- if the parent uses the tool as a private or a public one

- when you assign the tool's properties

To best understand how things work some examples are given. An example of how public tools work:

```
mt = MyTool("foo")
mt.getName()              # -> "ToolSvc.foo"

mt.Cut = 1
alg1.pubtool = mt
mt.Cut = 2
alg2.pubtool = mt
mt.Cut = 3
# alg1 and alg2 will have same tool, both with cut == 3
```

Here a single "`MyTool`" configurable is created with a simple name. In the constructor a "`ToolSvc.`" is appended (since there was no "." in the name). Since the tool is public the final value (3) will be used by both `alg1` and `alg2`.

An example of how private tools work:

```
mt = MyTool("foo")
mt.getName()              # -> "ToolSvc.foo"

mt.Cut = 1
alg1.privtool = mt
# alg1 gets "alg1.foo" configured with Cut==1
mt.Cut = 2
alg2.privtool = mt
# (for now) alg2 gets "alg2.foo" configured with Cut==2

# after assignment, can get renamed copy
from Gaudi.Configuration import Configurable
mt2 = Configurable.allConfigurables["alg2.foo"]
mt2.Cut = 3
# (now, really) alg2 gets "alg2.foo" configured with Cut==3
```

Again, the same tool configurable is created and implicitly renamed. An initial cut of 1 is set and the tool configurable is given to alg1. Guadi makes a copy and the "`ToolSvc.foo`" name of the original is changed to "`alg1.foo`" in the copy. The original then as the cut changed to 2 and given to alg2. Alg1's tool's cut is still 1. Finally, the copied `MyTool` configurable is looked up using the name "`alg2.foo`". This can be used if you need to configure the tool after it has been assigned to alg2.

6.5.3 The Package `Configure` Class and Optional Helper Classes

Every package that needs any but the most trivial configuration should provide a `Configure` class. By convention this class should be available from the module named after the package. When it is instantiated it should:

- Upon construction (in `__init__()`), provide a sensible, if maybe incomplete, default configuration for the general features the package provides.
- Store any and all configurables it creates in the instance (Python's `self` variable) for the user to later access.

In addition, the package author is encouraged to provide one or more "helper" classes that can be used to simplify non-default configuration. Helper objects can either operate on the `Configure` object or can be passed in to `Configure` or both.

To see an example of helpers are written look at:

```
$SITEROOT/dybgaudi/InstallArea/python/GenTools/Helpers.py
```

Package authors should write these classes and all higher layers may make use of these classes.

6.5.4 User Job Option Scripts

The next layer consists of job option scripts. These are short Python scripts that use the lower layers to provide non-default configuration that makes the user's job unique. However, these are not "main program" files and do not execute on their own (see next section).

Users can configure an entire job in one file or spread parts of the configuration among multiple files. The former case is useful for bookkeeping and the latter is if the user wants to run multiple jobs that differ in only a small part of their configuration. In this second case, they can separate invariant configuration from that which changes from run to run.

An example of a job script using the `GenTools` helpers described above is:

```
1 from GenTools.Helpers import Gun
2 gunner = Gun()
3
4 import GaudiKernel.SystemOfUnits as units
5 gunner.timerator.LifeTime = int(60*units.second)
6 # ...
7 import GenTools
8 gt = GenTools.Configure("gun","Particle Gun",helper=gunner)
9 gt.helper.positioner.Position = [0,0,0]
```

In the first two lines a "`Gun`" helper class is imported and constructed with defaults. This helper will set up the tools needed to implement a particle gun based generator. It chooses a bunch of defaults such as particle type, momentum, etc, which you probably don't want so you can change them later. For example the mean life time is set in line 5. Finally, the package is configured and this helper is passed in. The configuration creates a `GtGenerator` algorithm that will drive the GenTools implementing the gun based kinematics generation. After the `Configure` object is made, it can be used to make more configuration changes.

This specific example was for GenTools. Other package will do different things that make sense for them. To learn what each package does you can read the `Configure` and/or helper code or you can read its inlined documentation via the `pydoc` program. Some related examples of this latter method:

```
1 shell> pydoc GenTools.Helpers
2 Help on module GenTools.Helpers in GenTools:
3
4 NAME
5     GenTools.Helpers
6
7 FILE
8     /path/to/NuWa-trunk/dybgaudi/InstallArea/python/GenTools/Helpers.py
9
10 DESCRIPTION
11     Several helper classes to assist in configuring GenTools.  They
12     assume geometry has already been setup.  The helper classes that
13     produce tools need to define a "tools()" method that returns an
14     ordered list of what tools it created.  Users of these helper classes
15     should use them like:
16
17 CLASSES
18     Gun
19     HepEVT
20 ...
21
22 shell> pydoc GenTools.Helpers.Gun
23 Help on class Gun in GenTools.Helpers:
24
25 GenTools.Helpers.Gun = class Gun
26  |  Configure a particle gun based kinematics
27  |
28  |  Methods defined here:
29  |
```

```
30 |  __init__(self, ...)
31 |      Construct the configuration.  Coustom configured tools can
32 |      be passed in or customization can be done after construction
33 |      using the data members:
34 |
35 |      .gun
36 |      .positioner
37 |      .timerator
38 |      .transformer
39 |
40 |      The GtGenerator alg is available from the .generatorAlg member.
41 |
42 |      They can be accessed for additional, direct configuration.
43 ...
```

6.5.5 User Job Option Modules

A second, complimentary high-level configuration method is to collect lower level code into a user job module. These are normal Python modules and as such are defined in a file that exist in the users current working, in the packages `python/` sub directory or otherwise in a location in the user's `PYTHONPATH`.

Any top level code will be evaluated as the module is `import`ed in the context of configuration (same as job option scripts). But, these modules can supply some methods, named by convention, that can allow additional functionality.

`configure(argv=[])` This method can hold all the same type of configuration code that the job option scripts do. This method will be called just after the module is `import`ed. Any command line options given to the module will be available in `argv` list.

`run(appMgr)` This method can hold code that is to be executed after the configuration stage has finished and all configuration has been applied to the actual underlying C++ objects. In particular, you can define pure-Python algorithms and add them to the `TopAlg` list.

There are many examples Job Option Modules in the code. Here are some specific ones.

`GenTools.Test` this module[2] gives an example of a `configure(argv=[])` function that parses command line options. Following it will allow users to access the command line usage by simply running `nuwa.py -m 'GenTools.Test --help'`.

`DivingIn.Example` this module[3] gives an example of a Job Option Module that takes no command line arguments and configures a Python Algorithm class into the job.

6.5.6 The `nuwa.py` main script

Finally, there is the layer on top of it all. This is a main Python script called `nuwa.py` which collects all the layers below. This script provides the following features:

- A single, main script everyone uses.
- Configures framework level things
  - Python, interactive vs. batch
  - Logging level and color
  - File I/O, specify input or output files on the command line

[2] Code is at `dybgaudi/Simulation/GenTools/python/GenTools/Test.py`.
[3] Code is at `tutorial/DivingIn/python/DivingIn/Example.py`

- Geometry
- Use or not of the archive event store
- Access to visualization
- Running of user job option scripts and/or loading of modules

After setting up your environment in the usual way the `nuwa.py` script should be in your execution `PATH`. You can get a short help screen by just typing[4]:

```
shell> nuwa.py --help
Usage:
    This is the main program to run NuWa offline jobs.

    It provides a job with a minimal, standard setup.  Non standard
    behavior can made using command line options or providing additional
    configuration in the form of python files or modules to load.

    Usage:

      nuwa.py [options] [-m|--module "mod.ule --mod-arg ..."] \
              [config1.py config2.py ...] \
              [mod.ule1 mod.ule2 ...] \
              [input1.root input2.root ...]

    Python modules can be specified with -m|--module options and may
    include any per-module arguments by enclosing them in shell quotes
    as in the above usage.  Modules that do not take arguments may
    also be listed as non-option arguments.  Modules may supply the
    following functions:

    configure(argv=[]) - if exists, executed at configuration time

    run(theApp) - if exists, executed at run time with theApp set to
    the AppMgr.

    Additionally, python job scripts may be specified.

    Modules and scripts are loaded in the order they are specified on
    the command line.

    Finally, input ROOT files may be specified.  These will be read in
    the order they are specified and will be assigned to supplying
    streams not specificially specified in any input-stream map.

    The listing of modules, job scripts and/or ROOT files may be
    interspersed but must follow all options.


Options:
  -h, --help            show this help message and exit
  -A, --no-aes          Do not use the Archive Event Store.
  -l LOG_LEVEL, --log-level=LOG_LEVEL
                        Set output log level.
  -C COLOR, --color=COLOR
                        Use colored logs assuming given background ('light' or
                        'dark')
  -i, --interactive     Enter interactive ipython shell after the run
                        completes (def is batch).
  -s, --show-includes   Show printout of included files.
  -m MODULE, --module=MODULE
```

[4] Actual output may differ slightly.

```
53                        Load given module and pass optional argument list
54  -n EXECUTIONS, --executions=EXECUTIONS
55                        Number of times to execute list of top level
56                        algorithms.
57  -o OUTPUT, --output=OUTPUT
58                        Output filename
59  -O OUTPUT_STREAMS, --output-streams=OUTPUT_STREAMS
60                        Output file map
61  -I INPUT_STREAMS, --input-streams=INPUT_STREAMS
62                        Input file map
63  -H HOSTID, --hostid=HOSTID
64                        Force given hostid
65  -R RUN, --run=RUN     Set run number
66  -N EXECUTION, --execution=EXECUTION
67                        Set the starting execution number
68  -V, --visualize       Run in visualize mode
69  -G DETECTOR, --detector=DETECTOR
70                        Specify a non-default, top-level geometry file
```

Each job option `.py` file that you pass on the command line will be `eval`uated in turn and the list of `.root` files will be appended to the "`default`" input stream. Any non-option argument that does not end in `.py` or `.root` is assumed to be a Python module which will be loaded as described in the previous section.

If you would like to pass command line arguments to your module, instead of simply listing them on the command line you must `-m` or `--module`. The module name and arguments must be surrounded by shell quotes. For example:

```
1 shell> nuwa.py -n1 -m "DybPython.TestMod1 -a foo bar" \
2                    -m DybPython.TestMod2 \
3                    DybPython.TestMod3
```

In this example, only `DybPython.TestMod1` takes arguments. `TestMod2` does not but can still be specified with "`-m`". As the help output states, modules and job script files are all loaded in the order in which they are listed on the command line. All non-option arguments must follow options.

6.5.7 Example: Configuring `DetSimValidation`

During the move from the legacy G4dyb simulation to the Gaudi based one an extensive validation process was done. The code to do this is in the package `DetSimValidation` in the `Validation` area. It is provides a full-featured configuration example. Like `GenTools`, the configuration is split up into modules providing helper classes. In this case, there is a module for each detector and a class for each type of validation run. For example, test of uniformly distributed positrons can be configured like:

```
from DetSimValidation.AD import UniformPositron
up = UniformPositron()
```

Chapter 7

Data Model

- Over all structure of data
  - One package per processing stage
  - Single "header object" as direct TES DataObject
  - Providence
- Tour of DataModel packages

7.1 Overview

The "data model" is the suite of classes used to describe almost all of the information used in our analysis of the experimental results. This includes simulated truth, real and simulated DAQ data, calibrated data, reconstructed events or other quantities. Just about anything that an algorithm might produce is a candidate for using existing or requiring new classes in the *data model*. It does not include some information that will be stored in a database (reactor power, calibration constants) nor any analysis ntuples. In this last case, it is important to strive to keep results in the form of *data model* classes as this will allow interoperability between different algorithms and a common language that we can use to discuss our analysis.

The classes making up the *data model* are found in the `DataModel` area of a release. There is one package for each related collection of classes that a particular analysis stage produces.

7.1.1 HeaderObject

There is one special class in each package which inherits from `HeaderObject`. All other objects that a processing stage produces will be held, directly or indirectly by the `HeaderObject` for the stage. `HeaderObjects` also hold a some book-keeping items such as:

TimeStamp giving a single reference time for this object and any subobjects it may hold. See below for details on what kind of times the *data model* makes use of.

Execution Number counts the number of times the algorithm's execution method has been called, starting at 1. This can be thought of as an "event" number in more traditional experiments.

Random State holds the stage of the random number generator engine just before the algorithm that produced the `HeaderObject` was run. It can be used to re-run the algorithm in order to reproduce and arbitrary output.

Input `HeaderObjects` that were used to produce this one are referenced in order to determine providence.

Time Extent records the time this data spans. It is actually stored in the `TemporalDataObject` base class.

7.2 Times

There are various times recorded in the data. Some are absolute but imprecise (integral number of ns) and others are relative but precise (sub ns).

7.2.1 Absolute Time

Absolute time is stored in `TimeStamp` objects from the `Conventions` package under `DataModel`. They store time as seconds from the Unix Epoch (Jan 1, 1970, UTC) and nanoseconds w/in a second. A 32 bit integer is currently given to store each time scale[1]. While providing absolute time, they are not suitable for recording times to a precision less than 1 ns. `TimeStamp` objects can be implicitly converted to a `double` but will suffer a loss of precision of 100s of μsec when holding modern times.

7.2.2 Relative Time

Relative times simply count seconds from some absolute time and are stored as a `double`.

7.2.3 Reference times

Each `HeaderObject` holds an absolute reference time as a `TimeStamp`. How each is defined depends on the algorithms that produced the `HeaderObject`.

Sub-object precision times

Some `HeaderObjects`, such as `SimHeader`, hold sub-objects that need precision times (eg `SimHits`). These are stored as doubles and are measured from the reference time of the `HeaderObject` holding the sub-objects.

7.2.4 Time Extents

Each `TemporalObject` (and thus each `HeaderObject`) has a time extent represented by an earliest TimeStamp followed by a latest one. These are used by the window-based analysis window implemented by the Archive Event Store**??** to determine when objects fall outside the window and can be purged. How each earliest/latest pair is defined depends on the algorithm that produced the object but are typically chosen to just contain the times of all sub-objects held by the `HeaderObject`.

7.2.5 How Some Times are Defined

This list how some commonly used times are defined. The list is organized by the top-level `DataObject` where you may find the times.

`GenHeader` Generator level information.

- **Reference Time** Defined by the generator output. It is the first or primary signal event interaction time.
- **Time Extent** Defined to encompass all primary vertices. Will typically be infinitesimally small.
- **Precision Times** Currently, there no precision times in the conventional sense. Each primary vertex in an event may have a unique time which is absolute and stored as a `double`.

SimHeader Detector Simulation output.

- **Reference Time** This is identical to the reference time for the `GenHeader` that was used to as input to the simulation.

[1] Before 2038 someone had better increase the size what stores the seconds!

Time Extent Defined to contain the times of all `SimHits` from all detectors.

Precision Times Each RPC/PMT `SimHit` has a time measured from the reference time.**FIXME: Need to check on times used in the** `Historian`.

ElecHeader

TrigHeader

Readout

...

7.3 Examples of using the Data Model objects

Please write more about me!

7.3.1 Tutorial examples

Good examples are provided by the tutorial project which is located under `NuWa-RELEASE/tutorial/`. Each package shoudl provide a simple, self contained example but note that sometimes they get out of step with the rest of the code or may show less than ideal (older) ways of doing things.

Some good examples to look at are available in the `DivingIn` tutorial package. It shows how to do almost all things one will want to do to write analysis. It includes, accessing the data, making histograms, reading/writing files. Look at the Python modules under `python/DivingIn/`. Most provide instructions on how to run them in comments at the top of the file. There is a companion presentation available as DocDB #3131[2].

[2] http://dayabay.ihep.ac.cn/cgi-bin/DocDB/ShowDocument?docid=3131

Chapter 8

Data I/O

Gaudi clearly separates transient data representations in memory from those that persist on disk. The transient representations are described in the previous section. Here the persistency mechanism is described from the point of view of configuring jobs to read and write input/output (I/O) files and how to extend it to new data.

8.1 Goal

The goal of the I/O subsystem is to persist or preserve the state of the event store memory beyond the life time of the job that produced it and to allow this state to be restored to memory in subsequent jobs.

As a consequence, any algorithms that operate on any particular state of memory should not depend, nor even be able to recognize, that this state was restored from persistent files or was generated "on the fly" by other, upstream algorithms.

Another consequence of this is that users should **not** need to understand much about the file I/O subsystem except basics such as deciding what to name the files. This is described in the section on configuration below. Of course, experts who want to add new data types to the subsystem must learn some things which are described in the section below on adding new data classes.

8.2 Features

The I/O subsystem supports these features:

Streams: Streams are time ordered data of a particular type and are named. In memory this name is the location in the Transient Event Store (TES) where the data will be accessed. On disk this name is the directory in the ROOT `TFile` where the `TTree` that stores the stream of data is located.

Serial Files: A single stream can be broken up into sequential files. On input an ordered list of files can be given and they will be navigated in order, transparently. On output, files closed and new ones opened based on certain criteria. **FIXME: This is not yet implemented! But, it is easy to do so, the hooks are there.**

Parallel Files: Different streams from one job need not be stored all together in the same file. Rather, they can be spread among one or more files. The mapping from stream name to file is user configurable (more on this below).

Navigation: Input streams can be navigated forward, backward and random access. The key is the "entry" number which simply counts the objects in the stream, independent of any potential file breaks.[1]

[1] Correct filling of the Archive Event Service is only guaranteed when using simple forward navigation.

Policy: The I/O subsystem allows for various I/O policies to be enforced by specializing some of its classes and through the converter classes.

8.3 Packages

The I/O mechanism is provided by the packages in the `RootIO` area of the repository. The primary package is `RootIOSvc` which provides the low level Gaudi classes. In particular it provides an event selector for navigating input as well as a conversion service to facilitate converting between transient and persistent representations. It also provides the file and stream manipulation classes and the base classes for the data converters. The concrete converters and persistent data classes are found in packages with a prefix "`Per`" under `RootIO/`. There is a one-to-one correspondence between these packages and those in `DataModel` holding the transient data classes.

The `RootIOSvc` is generic in the sense that it does not enforce any policy regarding how data is sent through I/O. In order to support Daya Bay's unique needs there are additional classes in `DybSvc/DybIO`. In particular `DybEvtSelector` and `DybStorageSvc`. The first enforces the policy that the "next event" means to advance to the next `RegistrationSequence`[2] and read in the objects that it references. The second also enforces this same policy but for the output.

8.4 I/O Related Job Configuration

I/O related configuration is handled by `nuwa.py`. You can set the input and output files on the command line. See section 6.5.6 for details.

8.5 How the I/O Subsystem Works

This section describes how the bits flow from memory to file and back again. It isn't strictly needed but will help understand the big picture.

8.5.1 Execution Cycle vs. Event

Daya Bay does not have a well defined concept of "event". Some physics interactions can lead overlapping collections of hits and others can trigger multiple detectors. To correctly simulate this reality it is required to allow for multiple results from an algorithm in any given run through the chain of algorithms. This run is called a "top level execution cycle" which might simplify to an "event" in other experiments.

8.5.2 Registration Sequence

In order to record this additional dimension to our data we use a class called `RegistrationSequence` (RS). There is one RS created for each execution cycle. Each time new data is added to the event store it is also recorded to the current RS along with a unique and monotonically increasing sequence number or index.

The RS also hold flags that can be interpreted later. In particular it holds a flag saying whether or not any of its data should be saved to file. These flags can be manipulated by algorithms in order to implement a filtering mechanism.

Finally, the RS, like all data in the analysis time window, has a time span. It is set to encompass the time spans of all data that it contains. Thus, RS captures the results of one run through the top level algorithms.

[2] **FIXME: This needs to be described in the Data Model chapter and a reference added here**

8.5.3 Writing data out

Data is written out using a `DybStorageSvc`. The service is given a RS and will write it out through the converter for the RS. This conversion will also trigger writing out all data that the RS points to.

When to write out

In principle, one can write a simple algorithm that uses `DybStorageSvc` and is placed at the end of the chain of top-level algorithms[3]. As a consequence, data will be forced to be written out at the end of each execution cycle. This is okay for simple analysis but if one wants to filter out records from the recent past (and still in the AES) based on the current record it will be too late as they will be already written to file.

Instead, to be completely correct, data must not be written out until every chance to use it (and thus filter it) has been exhausted. This is done by giving the job of using `DybStorageSvc` to the agent that is responsible for clearing out data from the AES after they have fallen outside the analysis window.

8.5.4 Reading data in

Just as with output, input is controlled by the RS objects. In Gaudi it is the jobs of the "event selector" to navigate input. When the application says "go to the next event" it is the job of the event selector to interpret that command. In the Daya Bay software this is done by `DybIO/DybEvtSelector` which is a specialization of the generic `RootIOSvc/RootIOEvtSelector`. This selector will interpret "next event" as "next RegistrationSequence". Loading the next RS from file to memory triggers loading all the data it referenced. The TES and thus AES are now back to the state they were in when the RS was written to file in the first place.

8.6 Adding New Data Classes

For the I/O subsystem to support new data classes one needs to write a persistent version of the transient class and a converter class that can copy information between the two.

8.6.1 Class Locations and Naming Conventions

The persistent data and converters classes are placed in a package under `RootIO/` named with the prefix "`Per`" plus the name of the corresponding `DataModel` package. For example:

`DataModel/GenEvent/` $\longleftrightarrow$ `RootIO/PerGenEvent/`

Likewise, the persistent class names themselves should be formed by adding "`Per`" to the their transient counterparts. For example, `GenEvent`'s `GenVertex` transient class has a persistent counterpart in `PerGenEvent` with the name `PerGenVertex`.

Finally, one writes a converter for each top level data class (that is a subclass of DataObject with a unique Class ID number) and the converters name is formed by the transient class name with "`Cnv`" appended. For example the class that converts between `GenHeader` and `PerGenHeader` is called `GenHeaderCnv`.

The "`Per`" package should produce both a linker library (holding data classes) and a component library (holding converters). As such the data classes header (`.h`) files should go in the usual `PerXxx/PerXxx/` subdirectory and the implementation (`.cc`) files should go in `PerXxx/src/lib/`. All converter files should go in `PerXxx/src/components/`. See the `PerGenHeader` package for example.

8.6.2 Guidelines for Writing Persistent Data Classes

In writing such classes, follow these guidelines which differ from normal best practices:

[3] This is actually done in `RootIOTest/DybStorageAlg`

- Do not include any methods beyond constructors/destructors.
- Make a default constructor (no arguments) as well as one that can set the data members to non-default values
- Use public, and not private, data members.
- Name them with simple, but descriptive names. Don't decorate them with "`m_`", "`f`" or other prefixes traditionally used in normal classes.

8.6.3 Steps to Follow

1. Your header class should inherit from `PerHeaderObject`, all sub-object should, in general, not inherit from anything special.
2. Must provide a default constructor, convenient to define a constructor that passes in initial values.
3. Must initialize **all** data members in any constructor.
4. Must add each header file into `dict/headers.h` file (file name must match what is in `requirements` file below.
5. Must add a line in `dict/classes.xml` for every class and any STL containers or other required instantiated templates of these classes. If the code crashes inside low-level ROOT I/O related "T" classes it is likely because you forgot to declare a class or template in `classes.xml`.
6. Run a `RootIOTest` script to generate trial output.
7. Read the file with bare root + the `load.C` script.
8. Look for ROOT reporting any undefined objects or missing streamers. This indicates missing entries in `dict/classes.xml`.
9. Browse the tree using a `TBrowser`. You should be able to drill down through the data structure. Anything missing or causes a crash means missing `dict/classes.xml` entries or incorrect/incomplete conversion.
10. Read the file back in using the `RootIOTest` script.
11. Check for any crash (search for "Break") or error in the logs.
12. Use the `diff_out.py` script to diff the output and intput logs and check for unexplained differences (this may require you to improve `fillStream()` methods in the DataModel classes.

8.6.4 Difficulties with Persistent Data Classes

Due to limitations in serializing transient objects into persistent ones care must be taken in how the persistent class is designed. The issues of concern are:

Redundancy: Avoid storing redundant transient information that is either immaterial or that can be reconstructed by other saved information when the object is read back in.

Referencing: One can not directly store pointers to other objects and expect them to be correct when the data is read back in.

The Referencing problem is particularly difficult. Pointers can refer to other objects across different "boundaries" in memory. For example:

- Pointers to subobjects within the same object.
- Pointers to objects within the same HeaderObject hierarchy.
- Pointers to objects in a different HeaderObject hierarchy.
- Pointers to objects in a different execution cycle.
- Pointers to isolated objects or to those stored in a collection.

The `PerBaseEvent` package provides some persistent classes than can assist the converter in resolving references:

PerRef Holds a TES/TFile path and an entry number

PerRefInd Same as above but also an array index

In many cases the transient objects form a hierarchy of references. The best strategy to store such a structure is to collect all the objects into like-class arrays and then store the relationships as indices into these arrays. The `PerGenHeader` classes give an example of this in how the hierarchy made up of vertices and tracks are stored.

8.6.5 Writing Converters

The converter is responsible for copying information between transient and persistent representations. This copy happens in two steps. The first allows the converter to copy information that does not depend on the conversion of other top-level objects. The second step lets the converter fill in anything that required the other objects to be copied such as filling in references.

A Converter operates on a top level `DataObject` subclass and any subobjects it may contain. In Daya Bay software, almost all such classes will inherit from `HeaderObject`. The converter needs to directly copy only the data in the subclass of `HeaderObject` and can delegate the copying of parent class to its converter.

The rest of this section walks through writing a converter using the `GenHeaderCnv` as an example.

Converter Header File

First the header file:

```
1#include "RootIOSvc/RootIOTypedCnv.h"
2#include "PerGenEvent/PerGenHeader.h"
3#include "Event/GenHeader.h"
4
5class GenHeaderCnv : public RootIOTypedCnv<PerGenHeader,
6                                           DayaBay::GenHeader>
```

The converter inherits from a base class that is templated on the persistent and transient class types. This base class hides away much of Gaudi the machinery. Next, some required Gaudi boilerplate:

```
1public:
2    static const CLID& classID() {
3        return DayaBay::CLID_GenHeader;
4    }
5
6    GenHeaderCnv(ISvcLocator* svc);
7    virtual ~GenHeaderCnv();
```

The transient class ID number is made available and constructors and destructors are defined. Next, the initial copy methods are defined. Note that they take the same types as given in the templated base class.

```
StatusCode PerToTran(const PerGenHeader& per_obj,
                     DayaBay::GenHeader& tran_obj);

StatusCode TranToPer(const DayaBay::GenHeader& per_obj,
                     PerGenHeader& tran_obj);
```

Finally, the fill methods can be defined. These are only needed if your classes make reference to objects that are not subobjects of your header class:

```
    //StatusCode fillRepRefs(IOpaqueAddress* addr, DataObject* dobj);
    //StatusCode fillObjRefs(IOpaqueAddress* addr, DataObject* dobj);
```

FIXME: This is a low level method. We should clean it up so that, at least, the needed `dynamic_cast<>` on the `DataObject*` is done in the base class.

Converter Implementation File

This section describes what boilerplate each converter needs to implement. It doesn't go through the actual copying code. Look to the actual code (such as `GenHeaderCnv.cc`) for examples.

First the initial boilerplate and constructors/destructors.

```
#include "GenHeaderCnv.h"
#include "PerBaseEvent/HeaderObjectCnv.h"

using namespace DayaBay;
using namespace std;

GenHeaderCnv::GenHeaderCnv(ISvcLocator* svc)
    : RootIOTypedCnv<PerGenHeader,GenHeader>("PerGenHeader",
                                             classID(),svc)
{ }
GenHeaderCnv::~GenHeaderCnv()
{ }
```

Note that the name of the persistent class, the class ID number and the `ISvcLocator` all must be passed to the parent class constructor. One must get the persistent class name correct as it is used by ROOT to locate this class's dictionary.

When doing the direct copies, first delegate copying the `HeaderObject` part to its converter:

```
// From Persistent to Transient
StatusCode GenHeaderCnv::PerToTran(const PerGenHeader& perobj,
                                   DayaBay::GenHeader& tranobj)
{
    StatusCode sc = HeaderObjectCnv::toTran(perobj,tranobj);
    if (sc.isFailure()) return sc;

    // ... rest of specific p->t copying ...

    return StatusCode::SUCCESS;
}

// From Transient to Persistent
StatusCode GenHeaderCnv::TranToPer(const DayaBay::GenHeader& tranobj,
                                   PerGenHeader& perobj)
{
    StatusCode sc = HeaderObjectCnv::toPer(tranobj,perobj);
    if (sc.isFailure()) return sc;

    // ... rest of specific t->p copying ...

    return StatusCode::SUCCESS;
}
```

For filling references to other object you implement the low level Gaudi methods `fillRepRefs` to fill references in the persistent object and `fillObjRefs` for the transient. Like above, you should first delegate the filling of the `HeaderObject` part to `HeaderObjectCnv`.

```
StatusCode GenHeaderCnv::fillRepRefs(IOpaqueAddress*, DataObject* dobj)
{
    GenHeader* gh = dynamic_cast<GenHeader*>(dobj);
    StatusCode sc = HeaderObjectCnv::fillPer(m_rioSvc,*gh,*m_perobj);
    if (sc.isFailure()) { ... handle error ... }

    // ... fill GenHeader references, if there were any, here ...

    return sc;
}

StatusCode GenHeaderCnv::fillObjRefs(IOpaqueAddress*, DataObject* dobj)
{
    HeaderObject* hobj = dynamic_cast<HeaderObject*>(dobj);
    StatusCode sc = HeaderObjectCnv::fillTran(m_rioSvc,*m_perobj,*hobj);
    if (sc.isFailure()) { ... handle error ... }

    // ... fill GenHeader references, if there were any, here ...

    return sc;
}
```

Register Converter with Gaudi

One must tell Gaudi about your converter by adding two files. Both are named after the package and with "`_entries.cc`" and "`_load.cc`" suffixes. First the "load" file is very short:

```
#include "GaudiKernel/LoadFactoryEntries.h"
LOAD_FACTORY_ENTRIES(PerGenEvent)
```

Note one must use the package name in the CPP macro. Next the "entries" file has an entry for each converter (or other Gaudi component) defined in the package:

```
#include "GaudiKernel/DeclareFactoryEntries.h"
#include "GenHeaderCnv.h"
DECLARE_CONVERTER_FACTORY(GenHeaderCnv);
```

Resolving references

The Data Model allows for object references and the I/O code needs to support persisting and restoring them. In general the Data Model will reference an object by pointer while the persistent class must reference an object by an index into some container. To convert pointers to indices and back, the converter must have access to the transient data and the persistent container.

Converting references can be additionally complicated when an object held by one `HeaderObject` references an object held by another `HeaderObject`. In this case the converter of the first must be able to look up the converter of the second and obtain its persistent object. This can be done as illustrated in the following example:

```
#include "Event/SimHeader.h"
#include "PerSimEvent/PerSimHeader.h"
StatusCode ElecHeaderCnv::initialize()
{
    MsgStream log(msgSvc(), "ElecHeaderCnv::initialize");

    StatusCode sc = RootIOBaseCnv::initialize();
    if (sc.isFailure()) return sc;

```

```
10    if (m_perSimHeader) return StatusCode::SUCCESS;
11
12    RootIOBaseCnv* other = this->otherConverter(SimHeader::classID());
13    if (!other) return StatusCode::FAILURE;
14
15    const RootIOBaseObject* base = other->getBaseObject();
16    if (!base) return StatusCode::FAILURE;
17
18    const PerSimHeader* pgh = dynamic_cast<const PerSimHeader*>(base);
19    if (!pgh) return StatusCode::FAILURE;
20
21    m_perSimHeader = pgh;
22
23    return StatusCode::SUCCESS;
24 }
```

A few points:

- This done in `initialize()` as the pointer to the persistent object we get in the end will not change throughout the life of the job so it can be cached by the converter.
- It is important to call the base class's `initialize()` method as on line 7.
- Next, get the other converter is looked up by class ID number on line 12.
- Its persistent object, as a `RootIOBaseObj` is found and `dynamic_cast` to the concrete class on lines 15 and 18.
- Finally it is stored in a data member for later use during conversion at line 21.

8.6.6 CMT `requirements` File

The CMT `requirements` file needs:

- Usual list of `use` lines
- Define the headers and linker library for the public data classes
- Define the component library
- Define the dictionary for the public data classes

Here is the example for `PerGenEvent`:

```
1 package PerGenEvent
2 version v0
3
4 use Context    v*     DataModel
5 use BaseEvent    v*     DataModel
6 use GenEvent     v*     DataModel
7 use ROOT         v*     LCG_Interfaces
8 use CLHEP        v*     LCG_Interfaces
9 use PerBaseEvent v*     RootIO
10
11 # public code
12 include_dirs $(PERGENEVENTROOT)
13 apply_pattern install_more_includes more="PerGenEvent"
14 library PerGenEventLib lib/*.cc
15 apply_pattern linker_library library=PerGenEventLib
16
17 # component code
18 library PerGenEvent components/*.cc
```

```
19 apply_pattern component_library library=PerGenEvent
20
21
22 # dictionary for persistent classes
23 apply_pattern reflex_dictionary dictionary=PerGenEvent \
24               headerfiles=$(PERGENEVENTROOT)/dict/headers.h \
25               selectionfile=../dict/classes.xml
```

Chapter 9

Detector Description

9.1 Introduction

The Detector Description, or "DetDesc" for short, provides multiple, partially redundant hierarchies of information about the detectors, reactors and other physical parts of the experiment.

The description has three main sections:

Materials defines the elements, isotopes and materials and their optical properties that make up the detectors and the reactors.

Geometry describes the volumes, along with their solid shape, relative positioning, materials and sensitivity and any surface properties, making up the detectors and reactors. The geometry, like that of Geant4, consists of logical volumes containing other placed (or physical) logical volumes. Logical volumes only know of their children.

Structure describes a hierarchy of distinct, placed "touchable" volumes (Geant4 nomenclature) also known as Detector Elements (Gaudi nomenclature). Not all volumes are directly referenced in this hiearchy, only those that are considered important.

The data making up the description exists in a variety of forms:

XML files The definitive source of ideal geometry is stored in XML files following a well defined DTD schema.

DetDesc TDS objects In memory, the description is accessed as objects from the DetDesc package stored in the Transient Detector Store. These objects are largely built from the XML files but can have additional information added, such as offsets from ideal locations.

Geant4 geometry Objects in the Materials and Geometry sections can be converted into Geant4 geometry objects for simulation purposes.

9.1.1 Volumes

There are three types of volumes in the description. Figure 9.1 describes the objects that store logical, physical and touchable volume information.

Logical

XML `<logvol>`

C++ `ILVolume`

Description: The logical volume is the basic building block. It combines a shape and a material and zero or more daughter logical volumes fully contained inside the shape.

Example: The single PMT logical volume placed as a daughter in the AD oil and Pool inner/outer water shields[1].

9.1.2 Physical

XML `<physvol>`

C++ `IPVolume`

Description: Daughters are placed inside a mother with a transformation matrix giving the daughters translation and rotation with respect to the mother's coordinate system. The combination of a transformation and a logical volume is called a physical volume.

Example: The 192 placed PMTs in the AD oil logical volume.

9.1.3 Touchable

XML `<detelem>`

C++ `DetectorElement`

Description: Logical volumes can be reused by placing them multiple times. Any physical daughter volumes are also reused when their mother is placed multiple times. A touchable volume is the trail from the top level "world" volume down the logical/physical hiearchy to a specific volume. In Geant4 this trail is stored as a vector of physical volumes (G4TouchableHistory). On the other hand in Gaudi only local information is stored. Each DetectorElement holds a pointer to the mother DetectorElement that "supports" it as well as pointers to all child DetectorElements that it supports.

Example: The $8 \times 192 = 1536$ AD PMTs in the whole experiment

Scope of Detector Description, basics of geometry, structure and materials. Include diagrams showing geometry containment and structure's detector element / geometry info relationships.

9.2 Conventions

The numbering conventions reserve 0 to signify an error. PMTs and RPCs are addressed using a single bit-packed integer that also records the site and detector ID. The packing is completely managed by classes in `Conventions/Detectors.h`. The site ID is in `Conventions/Site.h` and the detector ID (type) is in `Conventions/DetectorId.h`. These are all in the `DataModel` area.

9.2.1 AD PMTs

The primary PMTs in an AD are numbered sequentially as well as by which ring and column they are in. Rings count from 1 to 8 starting at the bottom and going upwards. Columns count from 1 to 24 starting at the column just above the X-axis[2] and continuing counter clockwise if looking down at the AD. The sequential ID number can be calculated by:

column# + 24*(ring# - 1)

[1] We may create a separate PMT logical volume for the AD and one or two for the Pool to handle differences in PMT models actually in use.

[2] Here the X-axis points to the exit of the hall.

Besides the 192 primary PMTs there are 6 calibration PMTs. Their ID numbers are assigned 193 - 198 as 192 +:

1. top, target-viewing
2. bottom, target-viewing
3. top, gamma-catcher-viewing
4. bottom, gamma-catcher-viewing
5. top, mineral-oil-viewing
6. bottom, mineral-oil-viewing

FIXME: Add figures showing PMT row and column counts, orientation of ADs in Pool. AD numbers. coordinate system w.r.t pool.

9.2.2 Pool PMTs

Pool PMT counting, coordinate system w.r.t hall.

9.2.3 RPC

RPC sensor id convention. Coordinate system w.r.t. hall.

9.3 Coordinate System

As described above, every mother volume provides a coordinate system with which to place daughters. For human consumption there are three canonical coordinate system conventions. They are:

Global Th global coordinate system has its origin at the mid site with X pointing East, Y pointing North and Z pointing up. It is this system in which Geant4 works.

Site Each site has a local coordinate system with X pointing towards the exit and Z pointing up. Looking down, the X-Y origin is at the center of the tank, mid way between the center of the ADs. The Z origin is at the floor level which is also the nominal water surface. This makes the Pools and ADs at negative Z, the RPCs at positive Z.

AD Each AD has an even more local coordinate system. The Z origin is mid way between the inside top and bottom of the Stainless Steal vessel. This $Z_{AD} = 0$ origin is nominally at $Z_{Site} = -(5m - 7.5mm)$. The Z axis is collinear with the AD cylinder axis and the X and Y are parallel to X and Y of the Site coordinate system, respectively.

The Site and AD coordinate systems are related to each other by translation alone. Site coordinate systems are translated and rotated with respect to the Global system.

Given a global point, the local Site or AD coordinate system can be found using the `CoordSysSvc` service like:

```
1 // Assumed in a GaudiAlgorithm:
2 IService* isvc = 0;
3 StatusCode sc = service("CoordSysSvc", isvc, true);
4 if (sc.isFailure()) handle_error();
5 ICoordSvc* icss = 0;
6 sc = isvc->queryInterface(IID_ICoordSysSvc,(void**)&icss);
7 if (sc.isFailure()) handle_error();
```

```
8
9 Gaudi::XYZPoint globalPoint = ...;
10 IDetectorElement* de = icss->coordSysDE(globalPoint);
11 if (!de) handle_error();
12 Gaudi::XYZPoint localPoint = de->geometry()->toLocal(globalPoint);
```

9.4 XML Files

Schema, conventions.

9.5 Transient Detector Store

In a subclass of `GaudiAlgorithm` you can simply access the Transient Detector Store (TDS) using `getDet()` templated method or the `SmartDataPtr` smart pointer.

```
1
2 // if in a GaudiAlgorithm can use getDet():
3 DetectorElement* de = getDet<DetectorElement>("/dd/Structure/DayaBay");
4 LVolume* lv = getDet<LVolume>("/dd/Geometry/AD/lvOIL");
5
6 // or if not in a GaudiAlgorithm do it more directly:
7 IDataProviderSvc* detSvc = 0;
8 StatusCode sc = service("DetectorDataSvc",detSvc,true);
9 if (sc.isFailure()) handle_error();
10
11 SmartDataPtr<IDetectorElement> topDE(detSvc,"/dd/Structure/DayaBay");
12 if (!topDE) return handle_error();
13
14 // use topDE...
15
16 detSvc->release();
```

9.6 Configuring the Detector Description

The detector description is automatically configured for the user in `nuwa.py`.

9.7 PMT Lookups

Information about PMTs can be looked up using the `PmtGeomInfoSvc`. You can do the lookup using one of these types of keys:

Structure path which is the `/dd/Structure/...` path of the PMT

PMT id the PMT id that encodes what PMT in what detector at what site the PMT is

`DetectorElement` the pointer to the `DetectorElement` that embodies the PMT

The resulting `PmtGeomInfo` object gives access to global and local PMT positions and directions.

9.8 Visualization

Visualization can be done using our version of LHCb's PANORAMIX display. This display is started by running:

```
1 shell> nuwa.py -V
```

Take this tour:

- First, note that in the tree viewer on the left hand side, if you click on a folder icon it opens but if you click on a folder name nothing happens. The opposite is true for the leaf nodes. Clicking on a leaf's name adds the volume to the viewer.
- Try openning /dd/Geometry/PMT/lvHemiPmt. You may see a tiny dot in the middle of the viewer or nothing because it is too small.
- Next click on the yellow/blue eyeball icon on the right. This should zoom you to the PMT.
- You can then rotate with a mouse drag or the on-screen rollers. If you have a mouse with a wheel it will zoom in/out. Cntl-drag or Shift-drag pans.
- Click on the red arrow and you can "pick" volumes. A Ctrl-pick will delete a volume. A Shift-click will restore it (note some display artifacts can occur during these delete/restores).
- Go back to the Michael Jackson glove to do 3D moves.
- You can clear the scene with Scene-¿Scene-¿Clear. You will likely want to do this before displaying any new volumes as each new volume is centered at the same point.
- Scene-¿"Frame m" is useful thing to add.
- Materials can't be viewed but /dd/Structure can be.
- Another thing to try: Click on /dd/Structure/DayaBay, select the yellow/blue eye, then the red arror and Ctrl-click away the big cube. This shows the 3 sites. You can drill down them further until you get to the AD pmt arrays.
- Finally, note that there is still a lot of non-DayaBay "cruft" that should be cleaned out so many menu items are not particularly useful.

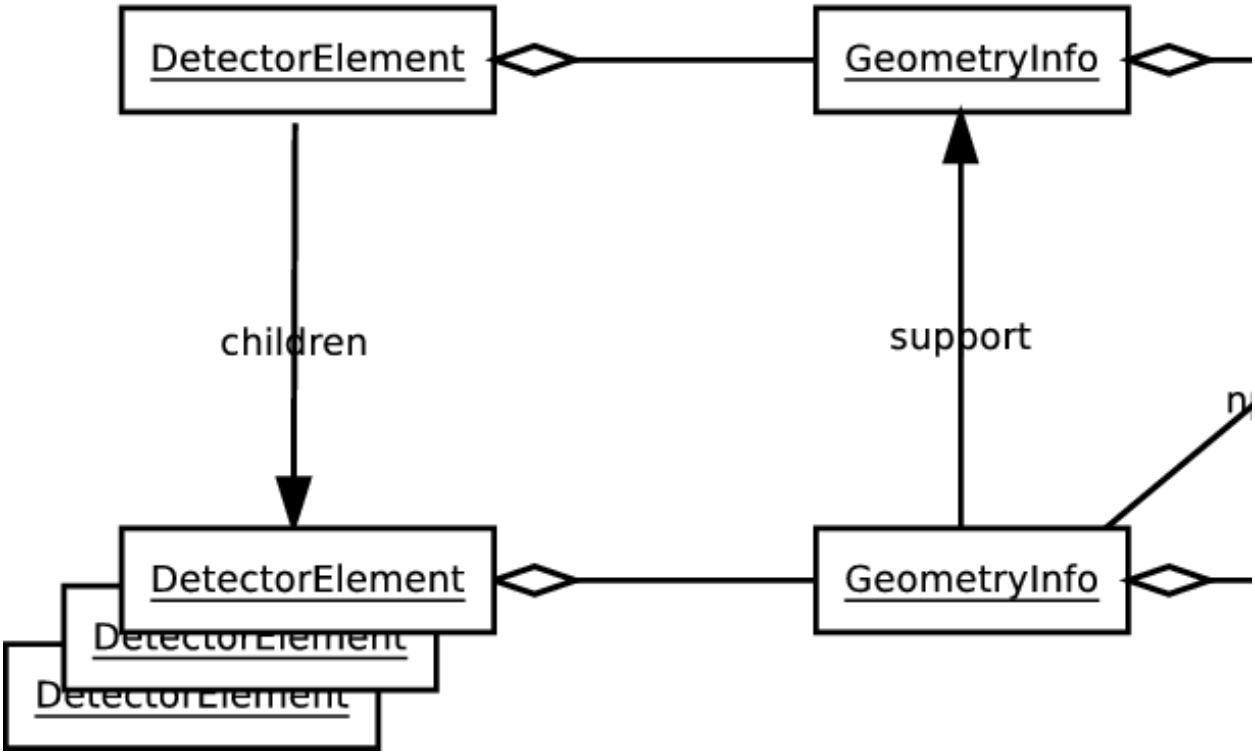

Figure 9.1: Logical, Physical and Touchable volumes.

Chapter 10

Kinematic Generators

10.1 Introduction

Generators provide the initial kinematics of events to be further simulated. They must provide a 4-position, 4-momentum and a particle type for every particle to be tracked through the detector simulation. They may supply additional "information" particles that are otherwise ignored. The incoming neutrino or radioactive decay parent are two examples of such particles.

10.2 Generator output

Each generated event is placed in the event store at the default location `/Event/Gen/GenHeader` but when multiple generators are active in a single job they will place their data in other locations under `/Event/Gen`.

The data model for this object is in `DataModel/GenEvent`. The `GenHeader` object is simply a thin wrapper that holds a pointer to a `HepMC::GenEvent` object. See HepMC documentation for necessary details on using this and related objects.

10.3 Generator Tools

A `GenEvent` is built from one or more special Gaudi Tools called `GenTools`. Each `GenTool` is responsible for constructing part of the kinematic information and multiple tools work in concert to produce a fully described event. This lets the user easily swap in different tools to get different results.

10.4 Generator Packages

There are a number of packages providing `GenTools`. The primary package is called `GenTools` and provides basic tools as well as the `GtGenerator` algorithm that ties the tools together. Every execution cycle the algorithm will run through its tools, in order, and place the resulting event in the event data store. A separate package, `GenDecay`, provides GenTools that will produce kinematics for various radioactive nuclear decays.

The `GtGenerator` is suitable only for "linear" jobs that only simulate a single type of event. In order to mix multiple events together the, so called, Fifteen suite of packages (see Ch **??**) are used. To configure for this type of job the `Gnrt` package's `Configure` is used.

10.5 Types of GenTools

The available GenTools and a sample of their properties are given. You can query their full properties with `properties.py ToolName`.

10.5.1 GenTools package

GtPositionerTool provides a local vertex 3-position. It does it by placing the vertex at its given point or distributing it about its given volume in various ways.

GtTransformTool provides global vertex 3-position and 3-direction given local ones. This will take existing an position and direction, interpret them as being defined in the given volume and transform them into global coordinates (needed for further simulation). It can optionally transform only position or direction.

GtTimeratorTool provides a vertex time. Based on a given lifetime (rate) it can distribute times exponentially or uniformly. It can also set the time in an "Absolut" (spelling intentional) or Relative manner. The former will set the time unconditionally and the latter will add the generated time to any existing value.

GtGunGenTool provides a local 4-momentum. It simulates a virtual particle "gun" that will shoot a given particle type in various ways. It can be set to point in a given direction or spray particles in a few patterns. It can select a fixed or distributed momentum.

GtBeamerTool provides a global 3-vertex and a global 4-momentum. It produces a parallel beam of circular cross section pointed at some detector element and starting from a given direction and distance away.

GtDiffuserBallTool provides a relative 3-vertex and local 4-momentum. It simulates the diffuser balls used in calibration. Subsequent positioner and transform tools are needed to place it at some non origin position relative to an actual volume.

GtHepEvtGenTool provides a local 4-momentum. It is used to read in kinematics in HepEVT format either from a file or through a pipe from a running executable. Depending on the HepEVT source it may need to be followed by positioner, timerator or transform tools.

10.5.2 GenDecay Package

The GenDecay package simulation radioactive decay of nuclei. It relies on Evaluated Nuclear Structure Data File (ENSDF) data sets maintained by National Nuclear Data Center (NNDC) located at BNL. It is fully data driven in that all information on branching fractions, half lifes and radiation type are taken from the ENSDF data sets. GenDecay will set up a hierarchy of mothers and daughters connected by a decay radiation. When it is asked to perform a decay, it does so by walking this hierarchy and randomly selecting branches to follow. It will apply a correlation time to the lifetime of every daughter state to determine if it should force that state to decay along with its mother. The abundances of all uncorrelated nuclear states must be specified by the user.

The GenDecay package provides a single tool called `GtDecayerator` which provides a local 4-vertex and 4-momentum for all products. It should be followed up by positioner and transformer tools.

10.6 Configuration

General configuration is described in Ch 6. The `GenTools` and related packages follow these conventions. This section goes from low level to high level configuration.

10.6.1 Configurables

As described above, a `GtGenerator` algorithm is used to collect. It is configured with the following properties

TimeStamp sets an absolute starting time in integral number of seconds. Note, the unit is implicit, do not multiple by seconds from the system of units.

GenTools sets the ordered list of tools to apply.

GenName sets a label for this generator.

Location sets where in the event store to place the results.

Each tool is configured with its own, specific properties. For the most up to date documentation on them, use the `properties.py` tool. Common or important properties are described:

Volume names a volume, specifically a Detector Element, in the geometry. The name is of the form "`/dd/Structure/Detector/SomElement`".

Position sets a local position, relative to a volume's coordinate system.

Spread alone or as a modifier is used to specify some distribution width.

Strategy or Mode alone or as a modifier is used to modify some behavior of the tool.

GenDecay Configurables

The GenDecay package provides a GtDecayerator tool which has the following properties.

ParentNuclide names the nuclide that begins the decay chain of interest. It can use any libmore supported form such as "U-238" or "238U" and is case insensitive.

ParentAbundance the abundance of this nuclide, that is, the number of nuclides of this type.

AbundanceMap a map of abundances for all nuclides that are found in the chain starting at, and including, the parent. If the parent is listed and ParentAbundance is set the latter takes precedence.

SecularEquilibrium If true (default), set abundances of uncorrelated daughter nuclides (see CorrelationTime property) to be in secular equilibrium with the parent. If any values are given by the AbundanceMap property, they will take precedence.

CorrelationTime Any nuclide in the chain that has a decay branch with a half life (total nuclide halflife * branching fraction) shorter than this correlation time will be considered correlated with the parent(s) that produced it and the resulting kinematics will include both parent and child decays together and with a time chosen based on the parent abundance. Otherwise, the decay of the nuclide is considered dependent from its parent and will decay based on its own abundance.

10.6.2 `GenTools.Configure`

The `GenTools` package's `Configure` object will take care of setting up a `GtGenerator` and adding it to the list of "top algorithms". The `Configure` object requires a "helper" object to provide the tools.

There are several helpers provided by `GenTools` and one provided by `GenDecay` that cover most requirements. If a job must be configured in a way that no helper provides, then a new helper can be written using the existing ones as examples. The only requirement is that a helper object provides a `tools()` method that returns a list of the tools to add to a GtGenerator algorithm.

Each helper described below takes a number of arguments in its constructor. They are given default values so a default helper can be constructed to properly set up the job to do something, but it may not be what you want. After construction the objects are available as object members taking the same name as the argument.

Helpers are self documented and the best way to read this is using the `pydoc` program which takes the full Python name. For example:

```
shell> pydoc GenTools.Helpers.Gun
Help on class Gun in GenTools.Helpers:

GenTools.Helpers.Gun = class Gun
 |  Configure a particle gun based kinematics
 |
 |  Methods defined here:
 |
 |  __init__(....)
....
```

Remember that `__init__()` is the constructor in Python.

The rest of this section gives the full Python name and a general description of the available helpers. Again, use `pydoc` to see the reference information.

`GenTools.Helpers.Gun` takes a volume and a gun, positioner, timerator and a transformer to set up a `GtGunGenTool` based generator.

`GenTools.Helpers.DiffuserBall` as above but sets up a diffuser ball. It also takes an AutoPositionerTool to modify the location of the diffuser ball in the geometry.

`GenTools.Helpers.HepEVT` takes a source of HepEVT formatted data and positioner, timerator and transformer tools.

`GenDecay.Helpers.Decay` takes a volume and decayerator, positioner and timerator tools.

10.6.3 `Gnrtr.Configure` and its Stages

Currently, the, so called, "pull mode" or "Fifteen style" of mixing of different types of events configuration mechanisms need work.

10.6.4 GenTools Dumper Algorithm

The `GenTools` package provides an algorithm to dump the contents of the generator output to the log. It can be included in the job by creating an instance of the `GenTools.Dumper` class. The algorithm can be accessed through the resulting object via its `.dumper` member. From that you can set the properties:

Location in the event store to find the kinematics to dump.

StandardDumper set to `True` to use the dumper that HepMC provides. By default it will use one implemented in the algorithm.

10.6.5 GenTools Job Option Modules

The GenTools package provides a `GenTools.Test` Job Option Module which gives command line access to some of the helpers. It is used in its unit test "`test_gentools.py`". It takes various command line options of its own which can be displayed via:

```
shell> nuwa.py -m 'GenTools.Test --help'
Importing modules GenTools.Test [ --help ]
Trying to call configure() on GenTools.Test
Usage:
This module can be used from nuwa.py to run GenTools in a few canned way as a test.
```

```
It is run as a unit test in GenTools/tests/test_gentools.py

Options:
  -h, --help            show this help message and exit
  -a HELPER, --helper=HELPER
                        Define a "helper" to help set up GenTools is gun,
                        diffuser or hepevt.
  -v VOLUME, --volume=VOLUME
                        Define a volume to focus on.
  -s DATA_SOURCE, --data-source=DATA_SOURCE
                        Define the data source to use for HepEVT helper
```

10.7 MuonProphet

10.7.1 Motivation

MuonProphet [DocDB 4153, DocDB 4441] is designed to address the simulation of muon which will be a major background source of Daya Bay neutrino experiment. Spallation neutrons and cosmogenic background, namely 9Li, 8He etc., are supposed to give the biggest systematic uncertainty.

The vast majority of muons are very easy to identify due to its distinguishable characteristic in reality. Usually its long trajectory in water pool or AD will leave a huge amount of light and different time pattern rather than a point source.

The simulation of muon in Geant4 is quite time-consuming. The hugh amount of optical photons' propargation in detector, usually over a few million, can bring any computer to its knee. One CPU has to spend 20-30 minutes for a muon track sometimes. The real muon rate requires to simulate is a few hundred to a thousand per second.

In the end people realized that they only need to know whether a muon has passed the detector and tagged, while not really care too much about how light are generated and distributed in water pool and AD.

Beside that it is technically impossible to finish all these muon simulation, the physics model of radioative isotope's generation in Geant4 is not very reliable. Photon nuclear process triggered by virturl or real photon, pion-nucleus interaction, nucleon-nucleus interaction, etc. are all possible be responsible to spallation background generation. They are poorly described in Genat4. Tuning the generation rate of some background is very difficult, since they are usually very low, then it is very inefficient to do MC study.

Based on these consideration MuonProphet is designed so that the tiresome optical photon simulation can be skipped and the generation of spallation background can be fully controled and fully simulated by Geant4.

10.7.2 Generation Mechanism

Firstly it starts from a muon track with initial vertex and momentum. The intersections of the muon track with each sub-detectors' surface and track lengths in each segment are calculated. Low energy muon could stop in detector according to a calculation based on an average dE/dx. According to its track length in water and whether it crossed RPC and user configuration it will determine whether this track is going to be triggered. Spallation neutron and cosmogenic background generation rate is usually a function of muon's energy, track length and material density. According to a few empirical formulas from early test beam and neutrino experiments, spallation neutron and/or radioactive isotopes are generated around the muon track. Because water is not sensitive to radioactive isotopes and their initial momentum is very low, they are only generated in AD. Muon is always tagged as "don't need simulation" by a trick in Geant4. However neutron and radioactive isotope are left for full Geant4 simulation.

10.7.3 Code Organisation

Besides the big structure determined by the motivation most parts of the codes are loosely bound together. Under MuonProphet/src/functions, all generation probabity functions, vertex and energy distribution functions are included. They can easily be modified and replaced. Under MuonProphet/src/components, MpGeometry.cc is dedicated to geometry related calculation; MpTrigger.cc is for trigger prediction; MpNeutron.cc and MpSpallation.cc handle the production of neutron and other isotopes respectively. All of them are controlled by MuonProphet::mutate like a usual gentool. It will make use of other radioactive background generators, so no need for extra code development.

10.7.4 Configuration

Here one example is given for 9Li or 8He background configuration. It will create a gentool - prophet. This tool should be attached after muon GtPositionerTool, GtTimeratorTool and GtTransformTool like demonstrated in MuonProphet/python/MuonProphet/FastMuon.py . According the formulas in [DocDB 4153, DocDB 4441] a set of four parameters including a gentool for an isotope background, yield, the energy where the yield is measured and lifetime must supplied. Following is a snippet of python code from FastMuon.py showing how it is configured.

```
# - muonprophet
prophet=MuonProphet()
prophet.Site= ‘‘DayaBay’’
# - spallation background
## - The tool to generate 9Li or 8He background
## - According to the formula refered in [DocDB 4153, DocDB 4441]
## - every isotope need a set of four parameters.
prophet.GenTools= [ ‘‘Li9He8Decayerator/Li9He8’’ ]
## - There is a measurement of yield 2.2e-7 cm2/g for 260 GeV muon,
## - then we can extrapolate the yield to other energy point.
prophet.GenYields= [ 2.2e-7 *units.cm2/units.g ]
prophet.GenYieldMeasuredAt= [ 260*units.GeV]
## - The lifetime of them is set to 0.002 second
prophet.GenLifetimes= [ 0.002*units.s]
# - trigger related configuration
## - Any muon track with a track length in water above 20 cm will be tagged as triggered.
prophet.TrkLengthInWaterThres= 20*units.cm
## - We can also assign a trigger efficiency even it passed above track length cut.
prophet.WaterPoolTriggerEff = 0.9999
```

10.7.5 Output

Geant4 will skip the muon simulation and do full simulation for neutron and other isotopes. The rest of the simulation chain in Fifteen is set up to be able to respond that correctly. Electronic simulation will only simulate the hits from spallation background and only pass a empty ElecHeader for the muon to the next simulation stage. If muon is tagged triggered, then trigger simulation will pop out a trigger header for the muon, otherwise, it will be dropped there like the real system.

In the final output of readout stream, user should expect the following situations: a) Only muon is triggered. There will be an empty ReadoutHeader for muon. User can trace back to the original GenHeader to confirm the situaion. b) Only spallation background is triggered. c) Both muon and background induced by this muon are triggered. There will be a empty ReadoutHeader for muon and another one with hits for the background. d) No trigger.

In reality if there is something very close to the muon in time, their hits will overlap and their hits are not distinguishable. For example, some fast background following muon won't be triggered separately. User should do the background trigger efficiency calculation based on the understanding of the real Daya Bay electronics.

10.7.6 Trigger Bits

Although the output got from MuonProphet simulation is empty, i.e. no hit, but the trigger information is set according to the fast simulation result. According to the geometry input it could have RPC and waterpool trigger.

10.7.7 Quick Start

There is one example already installed with nuwa. After you get into nuwa environment, you can start with

```
> nuwa.py -n50 -o fifteen.root -m "MuonProphet.FullChain" > log
```

It will invoke the FastMuon.py.

Chapter 11

Detector Simulation

11.1 Introduction

The detector simulation performs a Monte Carlo integration by tracking particles through the materials of our detectors and their surroundings until any are registered (hit) sensitive elements (PMTs, RPCs). The main package that provides this is called DetSim.

DetSim provides the following:

- Glue Geant4 into Gaudi through GiGa
- Takes initial kinematics from a generator, converts them to a format Geant4 understands.
- Takes the resulting collection of hits and, optionally, any unobservable statistics or particle histories, and saves them to the event data store.
- Modified (improved) Geant4 classes such as those enacting Cherenkov and scintillation processes.

The collection of "unobservable statistics" and "particle histories" is a fairly unique ability and is described more below.

11.2 Configuring DetSim

The `DetSim` package can be extensively configured. A default is set up done like:

```
1 import DetSim
2 detsim = DetSim.Configure()
```

You can provide various options to `DetSim`'s `Configure()`:

`site` indicating which site's geometry should be loaded. This can be "far" (the default) or one of the two near sites "dayabay" or "lingao" or you can combine them if you wish to load more than one.

`physics_list` gives the list of modules of Physics processes to load. There are two lists provided by the `configure` class: `physics_list_basic` and `physics_list_nuclear`. By default, both are loaded.

You can also configure the particle `Historian` and the `UnObserver` (unobservable statistics collector). Here is a more full example:

```
1 import DetSim.configure
2 # only load basic physics
3 detsim = DetSim.configure(physics_list=DetSim.configure.physics_list_basic)
4 detsim.historian(trackSelection="...", vertexSelection="...")
5 detsim.unobserver(stats=[...])
```

Details of how to form `trackSelection`, `vertexSelection` and `stats` are given below.

11.3 Truth Information

Besides hits, information on the "true" simulated quantities is available in the form of a *particle history* and a collection of *unobservable statistics*.

11.3.1 Particle Histories

Geant 4 is good at simulating particles efficiently. To this end, it uses a continually-evolving stack of particles that require processing. As particles are simulated, they are permanently removed from the stack. This allows many particles to be simulated in a large event without requiring the entire event to be stored at one time.

However, users frequently wish to know about more than simply the input (primary particles) and output (hits) of a simulation, and instead want to know about the intermediate particles. But simply storing all intermediate particles is problematic for the reason above: too many particles will bring a computer's virtual memory to it's knees.

Particle Histories attempts to give the user tools to investigate event evolution without generating too much extraneous data. The philosophy here is to generate only what the user requests, up to the granularity of the simulation, and to deliver the output in a Geant-agnostic way, so that data may be persisted and used outside the Geant framework.

Particle History Data Objects

Let us briefly review how Geant operates. A particle is taken off the stack, and a `G4Track` object is initialized to hold it's data. The particle is then moved forward a step, with an associated `G4Step` object to hold the relevant information. In particular, a `G4Step` holds two `G4StepPoint` representing the start and end states of the that particle.

The *Particle Histories* package crudely corresponds to these structures. There are two main data objects: `SimTrack` which corresponds to `G4Track`, and `SimVertex` which corresponds to a `G4StepPoint`. [1]

So, each particle that is simulated in by Geant can create a `SimTrack`. If the particle takes n steps in the Geant simulation, then it can create at most $n+1$ `SimVertex` objects (one at the start, and one for each step thereafter). If all vertices are saved, then this represents the finest granularity possible for saving the history of the simulation.

The data saved in a Track or Vertex is shown in Figures 11.3.1 and 11.3.1. Generally speaking, a SimTrack simply holds the PDG code for the particle, while a SimVertex holds a the state: position, time, volume, momentum, energy, and the process appropriate for that point in the simulation. Other information may be derived from these variables. For instance, the properties of a particle may be derived by looking up the PDG code via the ParticlePropertiesSvc, and the material of a step may be looked up by accessing the IPVolume pointer. (If there are two vertices with different materials, the material in between is represented by the first vertex. This is not true if vertices have been pruned.)

Each track contains a list of vertices that correspond to the state of the particle at different locations in it's history. Each track contains at least one vertex, the start vertex. Each Vertex has a pointer to it's parent Track. The relationship between SimVertices and SimTracks is shown in Figure 11.3.1.

The user may decide *which* vertices or tracks get saved, as described in Sec 11.3.1. If a SimVertex is pruned from the output, then any references that should have gone to that SimVertex instead point to the SimVertex preceeding it on the Track. If a SimTrack is pruned from the output, then any references that would have pointed to that track in fact point back to that track's parent. The output is guaranteed to have at least one SimTrack created for each primary particle that the generator makes, and each SimTrack is

[1] Another way to describe this is that a SimTrack corresponds to a single G4Trajectory, and SimVertex corresponds to a single G4TrajectoryPoint. The G4Trajectory objects, however, are relatively lightweight objects that are used by nothing other than the Geant visualization. It was decided not to use the G4Trajectory objects as our basis so as to remain Geant-independent in our output files. The similarity between the *Particle Histories* output and the G4Trajectories is largely the product of convergent evolution.

Figure 11.1: **SimTrack Accessors.** A list of accessible data from the SimTrack object.

```
class SimTrack {
        ...
        /// Geant4 track ID
        int trackId() const;

        /// PDG code of this track
        int particle() const;

        /// PDG code of the immediate parent to this track
        int parentParticle() const;

        /// Reference to the parent or ancestor of this track.
        const DayaBay::SimTrackReference& ancestorTrack() const;

        /// Reference to the parent or ancestor of this track.
        const DayaBay::SimVertexReference& ancestorVertex() const;

        /// Pointer to the ancestor primary kinematics particle
        const HepMC::GenParticle* primaryParticle() const;

        /// Pointers to the vertices along this track. Not owned.
        const vertex_list& vertices() const;

        /// Get number of unrecordeds for given pdg type
        unsigned int unrecordedDescendants(int pdg) const;
        ...
}
```

Figure 11.2: **SimVertex Accessors.** A list of accessible data from the SimVertex object.

```
class SimVertex {
  ...
  const SimTrackReference&    track()          const;
  const SimProcess&           process()        const;
  double                      time()           const;
  Gaudi::XYZPoint             position()       const;
  double                      totalEnergy()    const;
  Gaudi::XYZVector            momentum()       const;

  double                      mass()           const; // Approximate from 4-momentum.
  double                      kineticEnergy()  const; // Approximate from 4-momentum.

  const std::vector<SimTrackReference>& secondaries() const;
  ...
}
```

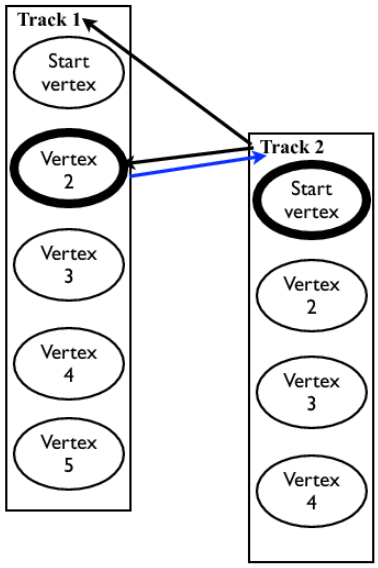

Figure 11.3: **Relationship between SimTrack and SimVertex** Track 1 represents a primary SimTrack, and Track 2 a secondary particle created at the end of Track 1s first step. Thus, the position, time, volume, and process may be the same for the two highlighted vertices. Track 2 contains a link both to its parent track (Track 1) and to its parent vertex (Vertex 2 of Track 1). There is also a forward link from Vertex 2 of Track 1 to Track 2. Not shown is that every SimVertex has pointer to its parent SimTrack, and each SimTrack has a list of its daughter SimVertices.

guaranteed to have at least one vertex, the start vertex for that particle, so all of these references eventually hand somewhere. An example of this pruning is shown in Figure **??**.

To keep track of this indirect parentage, links to a `SimTrack` or `SimVertex` actually use lightweight objects called `SimTrackReference` and `SimVertexReference`. These objects record not only a pointer to the object in question, but also a count of how indirect the reference is.. i.e. how many intervening tracks were removed during the pruning process.

Because pruning necessarily throws away information, some detail is kept in the parent track about those daughters that were pruned. This is kept as map by pdg code of "Unrecorded Descendents". This allows the user to see, for instance, how many optical photons came from a given track when those photons are not recorded with their own SimTracks. The only information recorded is the number of tracks pruned - for more elaborate information, users are advised to try Unobservable Statistics.

To get ahold of Particle Histories, you need to get the `SimHeader`. Each running of the Geant simulation creates a single `SimHeader` object, which contains a pointer to a single `SimParticleHistory` object. A `SimParticleHistory` object contains a list of primary tracks, which act as entrance points to the history for those who wish to navigate from first causes to final state. Alternatively, you may instead start with `SimHit` objects, which each contain a SimTrackReference. The references point back to the particles that created

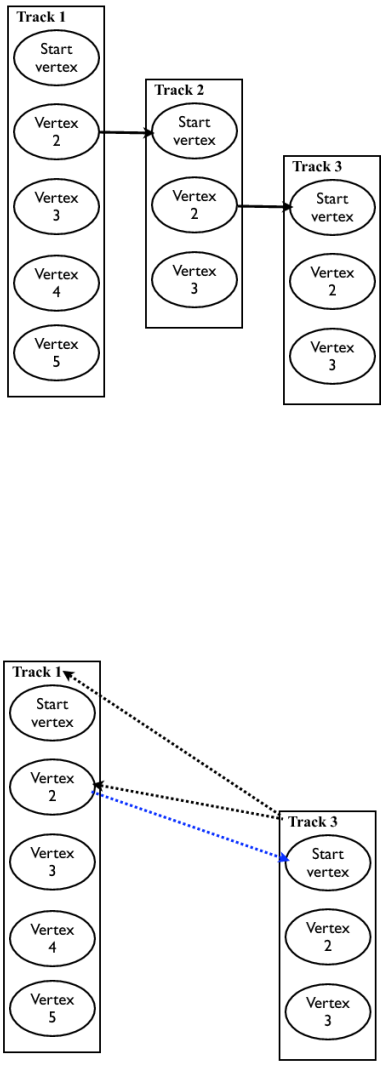

Figure 11.4: **History Pruning** The first figure shows a hypothetical case before pruning. The second case shows the links after pruning Track 2. The dotted lines indicate that the data objects record that the links are indirect.

the hit (e.g. optical photons in the case of a PMT), or the ancestor of that particle if its been pruned from the output.

Creation Rules

The Historian module makes use of the BOOST "Spirit" parser to build rules to select whether particles get saved as tracks and vertices. The user provides two selection strings: one for vertices and one for tracks. At initialization, these strings are parsed to create a set of fast Rule objects that are used to quickly and efficiently select whether candidate G4Tracks and G4StepPoints get turned into SimTracks or SimVertices respectively.

The selection strings describe the criteria neccessary for *acceptance*, not for rejection. Thus, the default strings are both "none", indicating that no tracks or vertices meet the criteria. In fact, the Historian knows to always record primary SimTracks and the first SimVertex on every track as the minimal set.

Selection strings may be:

"None" Only the default items are selected

"All" All items are created

An expression which is interpreted left-to-right.

Expressions consist of comparisons which are separated by boolean operators, grouped by parentheses. For example, a valid selection string could be:
`"(pdg != 20022 and totalEnergy<10 eV) or (materialName =='MineralOil')"` Each comparison must be of the form `<PARAMETER OPERATOR CONSTANT [UNIT]>`. A list of valid `PARAMETER`s is given in table 11.4. Valid `OPERATOR`s consist of `>,>=,<,<=,==,!=` for numerical parameters, and `==,!=` for string parameters. A few parameters accept custom operators - such as `in` for the detector element relational parameter. For numerical operators, `CONSTANT` is a floating-point number. For string paramters, `CONSTANT` should be of the form `'CaseSensitiveString'`, using a single quote to delimit the string. For numerical parameters, the user may (should) use the optional `UNIT`. Units include all the standard CLHEP-defined constants. All parameters and unit names are case-insensitive.

Boolean operators must come only in pairs. Use parentheses to limit them. This is a limitation of the parser. For instance, `"a<2 and b>2 and c==1"` will fail, but `"(a<2 and b>2) and c==1"` will be acceptable. This ensures the user has grouped his 'and' and 'or' operators correctly.

Because these selections are applied to every single G4Track and every single G4Step, having efficient selection improves simulation time. After compilation, selection is evaluated in the same order as provided by the user, left-to-right. Efficient selection is obtained if the user puts the easiest-to-compute parameters early in the selection. The slowest parameters to evaluate are those that derive from DetectorElement, including NicheID, Niche, DetectorId, SiteId, Site, AD, AdNumber, local_(xyz), DetectorElementName, etc. The fastest parameters are those that are already in the G4 data structures, such as particle code IDs, energy, global position, etc. String comparisons are of medium speed.

11.3.2 Examples, Tips, Tricks

Choosing specific particle types is easy. For instance, the following selects all particles except for optical photons. (This is an excellent use case for low-energy events like IBD.)

```
1 historian.TrackSelection = "(pdg != 20022)"
```

Here is a brief list of the more important PDG codes. A complete list can be found at the PDG website.

(Antiparticles are denoted by negative numbers.)

e^-	11
μ^-	13
γ	22
optical photon	20022
neutron	2112
proton	2212
π^0	111
$\pi-$	211

This example will save all tracks that are not optical photons, plus save one out of every 100 optical photons. This might be nice for an event viewer:

```
historian.TrackSelection = "(pdg != 20022) or (prescale by 100)"
```

This example will select any track created by a neutron capture (likely gamma rays):

```
historian.TrackSelection = "CreatorProcess == 'G4NeutronCapture'"
```

This should be contrasted with this example, which will save vertices with a neutron capture. This means: the vertex saved will be a neutron capture vertex, and is only valid for neutron tracks:

```
historian.VertexSelection = "Process == 'G4NeutronCapture'"
```

This example is slightly tricksy, but useful for muon-induced showers. It will select muons and particles that came off the muon, but not sub-particles of those. This lets you see delta rays or muon-induced neutrons, for example, but not record the entire shower.

```
historian.Track = "((AncestorTrackPdg = 13 or AncestorTrackPdg = -13)
                      and AncestorIndirection < 2)
                  or (pdg == 13 or pdg == -13)"
```

This example selects only vertices which are inside the oil volume or sub-volume of the oil at the LingAo detector 1. i.e. in oil, AVs, or scintillator volumes:

```
historian.VertexSelection = "DetElem in '/dd/Structure/AD/la-oil1'"
```

This example selects vertices which are in the oil, not any subvolumes:

```
historian.VertexSelection = "DetectorElementName == '/dd/Structure/AD/la-oil1'"
```

This example saves only start and end vertices, as well as vertices that change materials:

```
historian.VertexSelection = "IsStopping ==1 and MaterialChanged > 0"
```

This example saves a vertex about every 20 cm, or if the track direction changes by more than 15 degrees:

```
historian.VertexSelection = "distanceFromLastVertex > 20 cm or AngleFromLastVertex > 15 deg"
```

Users should fill out more useful examples here.

11.3.3 Unobservable Statistics

Description

Although users may be able to answer nearly any question about the history of an event with the *Particle Histories*, it may be awkward or time-consuming to compile certain variables. To this end, users may request "Unobservable" statistics to be compiled during the running of the code.

Figure 11.5: **SimStatistic** A Statistic object used for Unobservable Statistics.

```
class SimStatistic {
    SimStatistic() : m_count(0),
                     m_sum(0),
                     m_squaredsum(0) {}

    double count() const;      /// Counts of increment() call
    double sum() const;        /// Total of x over all counts.
    double squaredsum() const;/// Total of x^2 over all counts.
    double mean() const;       /// sum()/count()
    double rms() const;        /// Root mean square

    void increment(double x);   /// count+=1, sum+=x, sum2+=x*x

  private:
    double m_count;       ///< No. of increments
    double m_sum;         ///< Total of x over all counts.
    double m_squaredsum; ///< Total of x^2 over all counts.
}
```

For instance, let us say we want to know how many meters of water were traversed by all the muons in the event. We could do this above by turning on SimTracks for all muons and turning on all the SimVertecies at which the muon changed material.

```
historian.TrackSelection = "(pdg == 13 or pdg == -13)"
historian.VertexSelection = "(pdg == 13 or pdg == -13)
                            and (MaterialChanged >0 )"
```

Then, after the event had been completed, we would need to go through all the saved SimTracks and look for the tracks that were muons. For each muon SimTrack, we would need to go through each pair of adjacent SimVertices, and find the distance between each pair, where the first SimVertex was in water. Then we would need to add up all these distances. This would get us exactly what we wanted, but considerable code would need to be written, and we've cluttered up memory with a lot of SimVertices that we're only using for one little task.

To do the same job with the Unobserverable Statistics method, we need only run the "Unobserver" SteppingTask, and give it the following configuration:

```
UnObserver.Stats =[ ["mu_track_length_in_water" , "dx" ,
                     "(pdg == 13 or pdg == -13) and MaterialName==’Water’" ] ]
```

This creates a new statistic with the name `mu_track_length_in_water`, and fills it with exactly what we want to know!

This method is very powerful and allows the description of some sophisticated analysis questions at run-time. However, compiling many of these Statistics can be time-consuming during the execution of the simulaton. For serious, repeated analyses, using the *Particle Histories* may yield better results in the long run.

"Unobservable" Statistic Objects

Unobservable Statistics are stored in a SimStatistic object shown in Figure 11.3.3.

These statistic objects are stored in a map, referenced by name, in the `SimUnobservableStatisticsHeader`. This object in turn is stored in the `SimHeader`, once per simulated event.

Creation Rules

The `Unobserver` module operates using the same principles as the *Particle History* selector, above. At initialization, a selection string and variable string is parsed into a set of Rule objects that can be rapidly evaluated on the current G4Step. The user supplies a list of Statistics to the module. Each Statistic is defined as follows:
`["STATNAME" , "VARIABLE" , "EXPRESSION"]` or

```
["STATNAME_1" , "VARIABLE_1" ,
 "STATNAME_2" , "VARIABLE_2" ,
 "STATNAME_3" , "VARIABLE_3" ,
 ... , "EXPRESSION"]
```

Here, `STATNAME` is a string of the user's choosing that describes the statistic, and is used to name the statistic in the `SimUnobservableStatisticsHeader` for later retrieval. `VARIABLE` is a parameter listed in Table 11.4 that is the actual value to be filled. Only numeric parameters may be used as variables. `EXPRESSION` is a selection string, as described in Sec. 11.3.1. In the second form of listing, several different variables may be defined using the same selection string, to improve runtime performance (and make the configuration clearer).

Any number of statistics may be defined, at the cost of run-time during the simulation.

The statistics are filled as follows. At each step of the simulation, the current G4Step is tested against each `EXPRESSION` rule to see if the current step is valid for that statistic. If it is, then the `VARIABLE` is computed, and the Statistic object is incremented with the value of the variable.

11.3.4 Examples, Tips, Trucks

Statistics are *per-step*. For example:

```
UnObserver.Stats =[ ["x_vertex" , "global_x" ,
                     "(pdg == 13 or pdg == -13)'" ] ]
```

will yield a statistic n entries, where n is the number of steps taken by the muon, with each entry being that step's global X coordinate. However, you can do something like the following:

```
UnObserver.Stats =[ ["x_vertex" , "global_x" ,
            "(pdg == 13 or pdg == -13)' and IsStarting==1" ] ]
```

which will select only the start points for muon tracks. If you know that there will be at most one muon per event, this will yield a statistic with one entry at the muon start vertex. However, this solution is not generally useful, because a second muon in the event will confuse the issue - all you will be able to retrieve is the mean X start position, which is not usually informative. For specific queries of this kind, users are advised to use Particle Histories.

Users should fill out more useful examples here.

11.3.5 Parameter Reference

The Particle History parser and the Unobservable Statistics parser recognize the parameter names listed in table 11.4

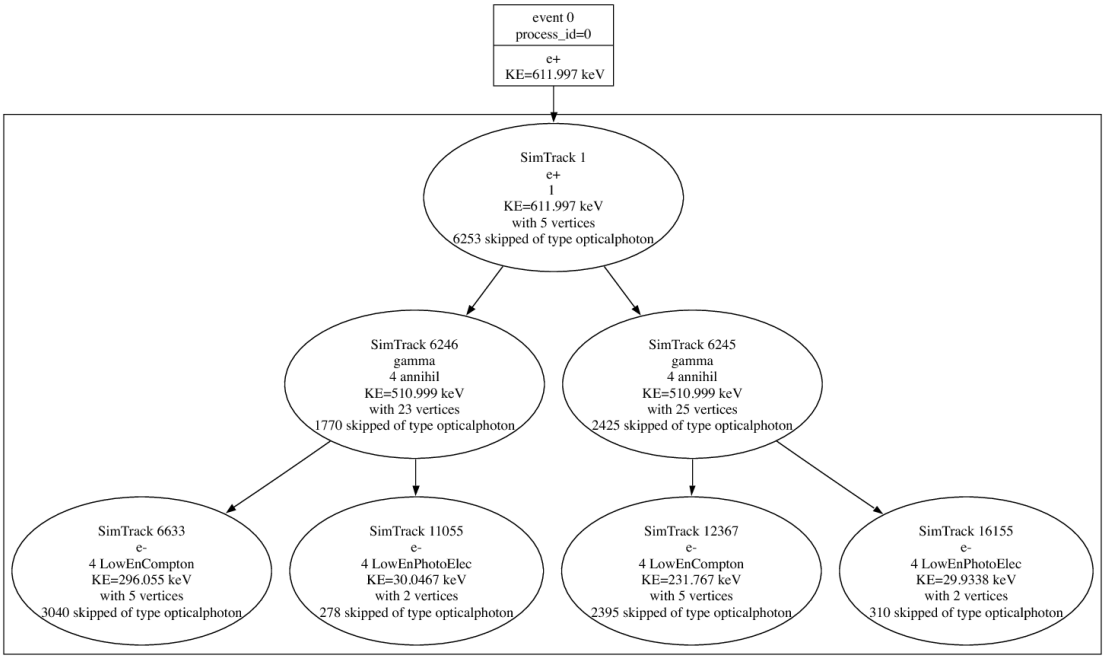

Figure 11.6: Output of tracks file for a single 1 MeV positron. Circles denote SimTracks - values listed are starting values. In this example, do_hits was set to zero.

11.3.6 The DrawHistoryAlg Algorithm

These lines in your python script will allow you to run the DrawHistoryAlg and the DumpUnobservableStatisticsAlg, which provide a straightforward way of viewing the output of the Particle Histories and Unobservables, respectively:

```
simseq.Members = [ "GiGaInputStream/GGInStream",
                   "DsPushKine/PushKine",
                   "DsPullEvent/PullEvent",
                   "DrawHistoryAlg/DrawHistory",
                   "DumpUnobservableStatisticsAlg/DumpUnobserved"
                 ]
```

The DrawHistoryAlg produces two “dot” files which can be processed by the GraphViz application. (A very nice, user-friendly version of this exists for the Mac.) The dot files describe the inter-relation of the output objects so that they can be drawn in tree-like structures. Sample output is shown in Figures 11.3.6 and 11.3.6.

The DrawHistoryAlg can be configured like so:

```
app.algorithm("DrawHistory").do_hits = 0
app.algorithm("DrawHistory").track_filename = 'tracks_%d.dot'
app.algorithm("DrawHistory").trackandvertex_filename = 'vertices_and_tracks_%d.dot'
```

The filename configuration is for two output files. Using ’%d’ indicates that the event number should be used, to output one file per event. The `do_hits` option indicates whether SimHits should be shown on the plot. (For scintillator events, this often generates much too much detail.)

The DumpUnobservableStatisticsAlg algorithm simply prints out the counts, sum, mean, and rms for each statistic that was declared, for each event. This is useful for simple debugging.

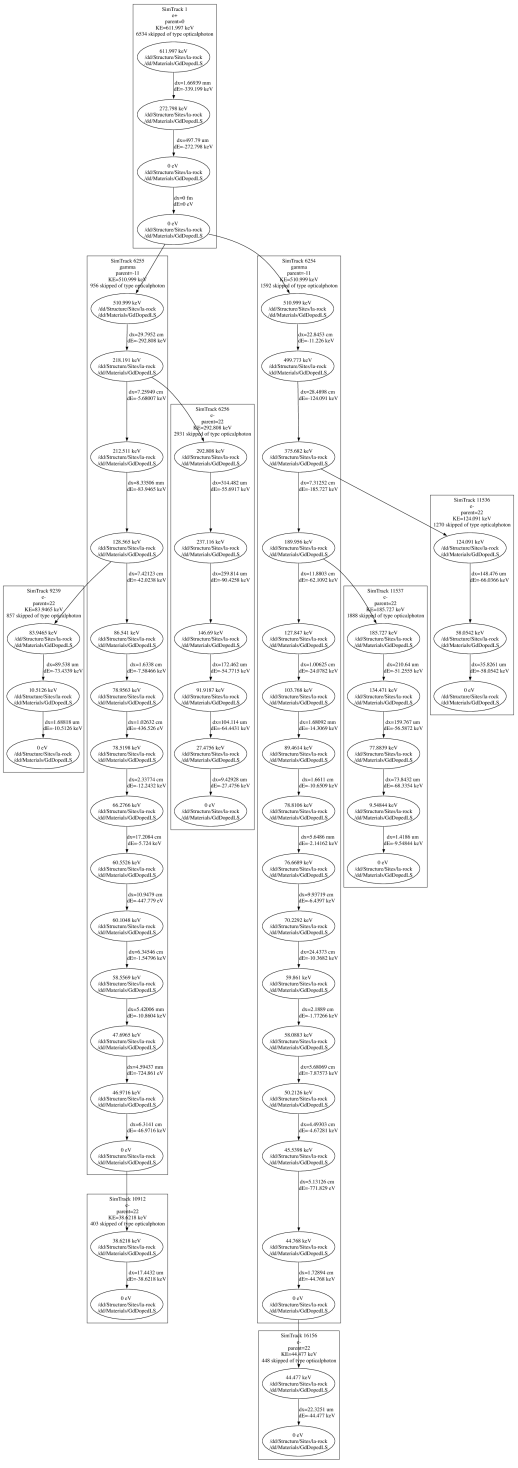

Figure 11.7: Output of tracks-and-vertices file for a single 1 MeV position. Boxes represent SimTracks, and circles represent SimVertecies.

11.4 Truth Parameters

Name & Synonyms	Type	Track	Vertex	Stats	Description
time t	double	X	X	X	Time of the vertex/track start/step
x global_x	double	X	X	X	Global X position of the vertex/-track start/step
y global_y	double	X	X	X	Global Y position of the vertex/-track start/step
z global_z	double	X	X	X	Global Z position of the vertex/-track start/step
r radius pos_r	double	X	X	X	Global sqrt(X*X+Y*Y) position of the vertex/step/start
lx local_x det_x	double	X	X	X	X Position relative to the local physical volume
ly local_y det_y	double	X	X	X	Y Position relative to the local physical volume
lz local_z det_z	double	X	X	X	Z Position relative to the local physical volume
lr local_r det_r	double	X	X	X	sqrt(X*X+Y*Y) position relative to the local physical volume
Volume VolumeName LogicalVolume	string	X	X	X	Name of the logical volume of vertex/track start/step
Material MaterialName	string	X	X	X	Name of material at vertex/track start/step
DetectorElementName	double	X	X	X	Name of best-match Detector Element at vertex/track start/step
Match DetectorElementMatch	double	X	X	X	Level of match for Detector Element. 0=perfect postive = inside
NicheId Niche	double	X	X	X	ID number (4-byte) best associated with DetElem
DetectorId	double	X	X	X	Detector ID number (4-byte)
SiteId	double	X	X	X	Site ID number (4-byte)
Site	double	X	X	X	Site number (1-16)
AD AdNumber	double	X	X	X	AD number (1-4)
momentum p	double	X	X	X	Momentum at vertex/track start/step
E totEnergy TotalEnergy	double	X	X	X	Energy at track start or vertex

Name & Synonyms	Type	Track	Vertex	Stats	Description
KE kineticEnergy	double	X	X	X	Kinetic energy at vertex/track start/step
vx dir_x u	double	X	X	X	X-direction cosine
vy dir_y v	double	X	X	X	Y-direction cosine
vz dir_z w	double	X	X	X	Z-direction cosine
ProcessType	double	X	X	X	Type of process (see below)
Process ProcessName	string	X	X	X	Name of current process (via `G4VProcess->GetProcessName()`)
pdg pdgcode particle	double	X	X	X	PDG code of particle. Note that opticalphoton=20022
charge ParticleCharge q	double	X	X	X	Charge of particle
id trackid	double	X	X	X	Geant TrackID of particle. Useful for debugging
creatorPdg creator	double	X	X	X	PDG code for the immediate parent particle
mass m	double	X	X	X	Mass of the particle
ParticleName	string	X	X	X	Name of the particle (Geant4 name)
CreatorProcessName CreatorProcess	string	X	X	X	Name of process that created this particle. (Tracks: same as "ProcessName")
DetElem in DetectorElement in	custom	X	X	X	Special: matches if the detector element specified supports the Track/Vertex volume
Step_dE dE	double		X	X	Energy deposited in current step
Step_dE_Ion de_ion ionization	double		X	X	Energy deposited by ionization in current step
Step_qDE quenched_dE qdE	double		X	X	Quenched energy. Valid only for scintillator
Step_dx StepLength dx	double		X	X	Step length
Step_dt StepDuration dt	double		X	X	Step duration

Name & Synonyms	Type	Track	Vertex	Stats	Description
Step_dAngle dAngle	double		X	X	Change in particle angle before/after this Step (degrees)
Ex E_weighted_x	double		X	X	Energy-weighted global position - x
Ey E_weighted_y	double		X	X	Energy-weighted global position - y
Ez E_weighted_z	double		X	X	Energy-weighted global position - z
Et E_weighted_t	double		X	X	Energy-weighted global time
qEx qE_weighted_x quenched_weighted_x	double		X	X	Quenched energy-weighted global position - x
qEy qE_weighted_y quenched_weighted_y	double		X	X	Quenched energy-weighted global position - y
qEz qE_weighted_z quenched_weighted_z	double		X	X	Quenched energy-weighted global position - z
qEt qE_weighted_t quenched_weighted_t	double		X	X	Quenched energy-weighted global time
IsStopping stop End	double		X	X	1 if particle is stopping 0 otherwise
IsStarting start begin	double		X	X	1 if particle is starting (this is the first step) 0 otherwise
StepNumber	double		X	X	Number of steps completed for this particle
VolumeChanged NewVolume	double		X	X	1 if the particle is entering a new volume 0 otherwise
MaterialChanged NewMaterial	double		X	X	1 if the particle is entering a new material 0 otherwise
ParentPdg AncestorPdg Ancestor	double	X	X		PDG code of the last ancestor where a SimTrack was created
ParentIndirection AncestorIndirection	double	X	X		Generations passed since the last ancestor was created
GrandParentPdg GrandParent	double	X	X		PDG code of the immediate ancestor's ancestor
GrandParentIndirection	double	X	X		Indirection to the immediate ancestor's ancestor
distanceFromLastVertex	double		X		Distance from the last created SimVertex.

Name & Synonyms	Type	Track	Vertex	Stats	Description
TimeSinceLastVertex	double		X		Time since the last created SimVertex.
EnergyLostSinceLastVertex	double		X		Energy difference sine the last created SimVertex
AngleFromLastVertex	double		X		Change in direction since the last created SimVertex (degrees)

Chapter 12

Electronics Simulation

12.1 Introduction

The Electronics Simulation is in the ElecSim package. It takes an SimHeader as input and produces an ElecHeader, which will be read in by the Trigger Simulation package. The position where ElecSim fit in the full simulation chain is given in figure 12.1. The data model used in ElecSim is summarized in the UML form in figure 12.2.

12.2 Algorithms

There are two algorithms. They are listed in table 12.1

12.3 Tools

Tools are declared as properties in the algorithms in the previous section. Two kinds of tools are present in the EleSim package. They are:

- Hit tools: these types of tools take SimHitHeader as input and generate ElecPulseHeader.
- FEE/FEC tools: these tools takes the output from hit tools in ElecPulseHeader and create ElecCrate. The foundation of these tools are the hardware of FEE for AD and FEC(Front-end Card) for RPC electronics.

12.3.1 Hit Tools

12.3.2 FEE Tool: EsIdealFeeTool

The properties is summaried in table 12.2.

Pulses(**ElecPulse**) generated in HitTools are first mapped to channels in each FEE board via CableSvc service. For each channel, pulses are then converted and time-sequenced to create two analog signals to simulate real signals in FEE. The two major analog signals are RawSignal and shapedSignal. The following shows the generation steps.

- pmt Analog Signal (**m_pmtPulse(nSample) vector<double>**): each pulse (**ElePulse**) is converted to a pmt analog signal(**m_pmtPulse(nSample)**) according to an ideal pmt waveform parametrization given in equation 12.1.
- Shaped PMT Signal(**m_shapedPmtPulse(nSample)**): the pmt analog signal (**m_pmtPulse(nSample)** is convoluted with shaper transfer function to get the shaper output analog singal (**shapedPmtPulse(nSample)**).

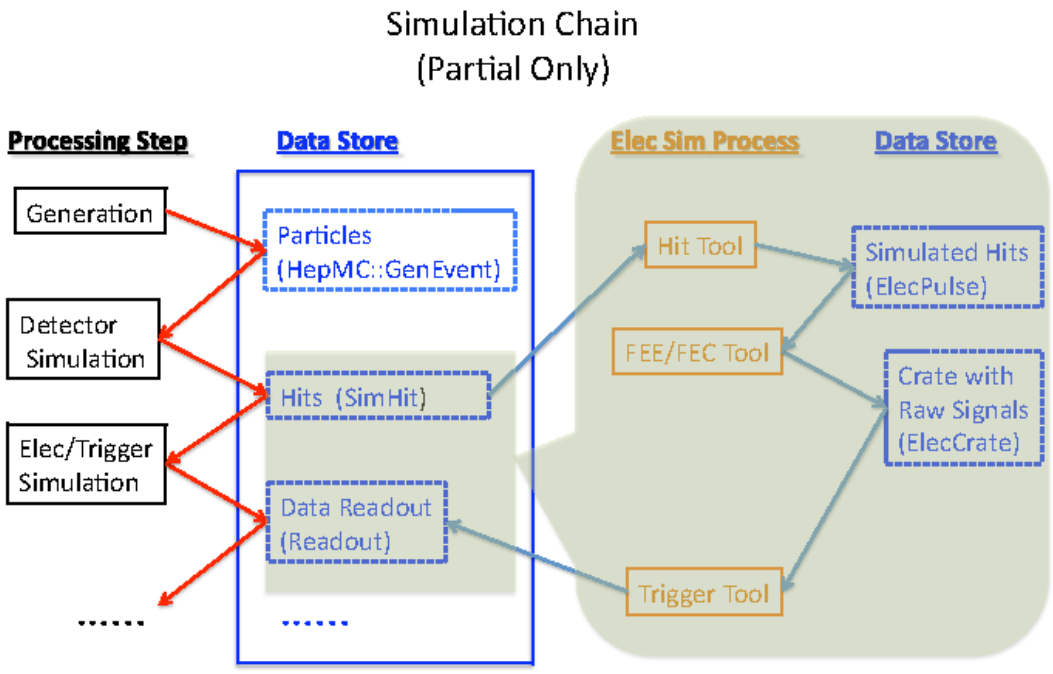

Figure 12.1: Electronics Simulation Chain

- RawSignal (**RawSignal(simSamples) vector<double>**): represents the time sequenced pmt signal with gaussian distributed noises included. This RawSignal is sent to discriminator to form multiplicit and TDC values. Analogsum is also based on this RawSignal.
- shapedSignal(**shapedSignal(SimSample) vector<double>**) is composed of time-sequenced shapedPMTsignals(**shapedPmtPulse**).

$$V(t) = VoltageScale \cdot \frac{(e^{-t/t_0} - e^{-t/t_1})}{(t_1 - t_0)} \qquad (12.1)$$
$$t_0 = 3.6ns$$
$$t_1 = 5.4ns$$

Multiplicity Generation and TDC

Multiplicity at hit Clock i for one FEE board is the sum of the hitHold signal(**hitHold vector<int>**) at the hit Clock hitHold(i) for all the hitted channels in the FEE channel. Figure 12.3 shows the flow on how the hitHold signals are generated. One example of two 1 p.e. pulses are shown in figure 12.4.

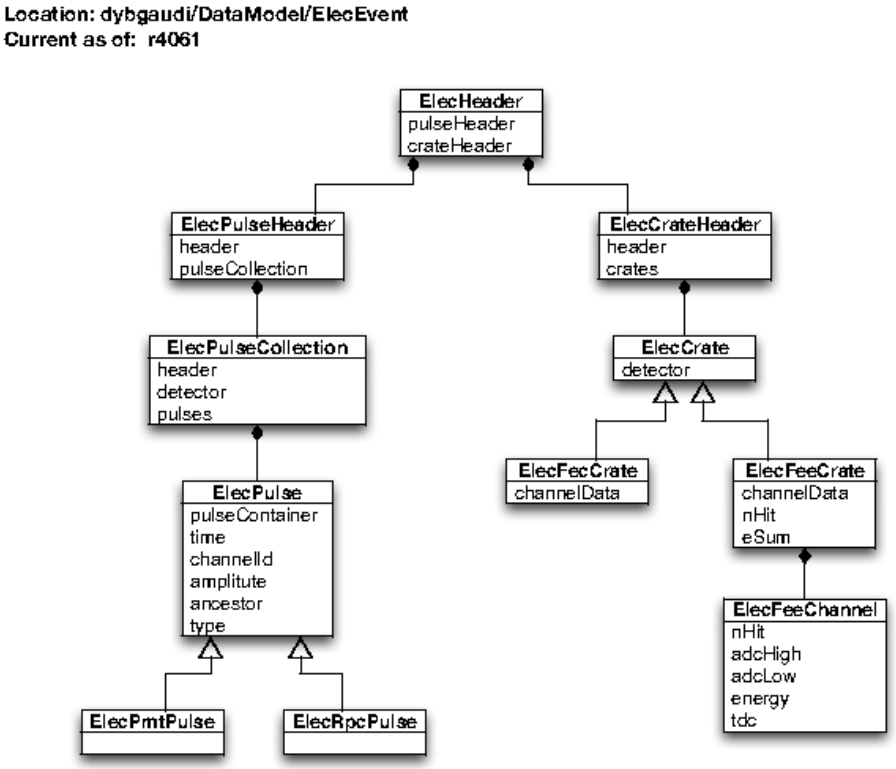

Figure 12.2: UML for data model in ElecSim.

ADC Generation

12.4 Simulation Constant

Simulation constants based on electronics hardware is defined in dybgaudi/DataModel/Conventions/Conventions/Electronics.h Table 12.3 summaries the major vaiables defined and their hardwired values.

Algorithm Name	Property	Defualt
EsFrontEndAlg	SimLocation	SimHeaderLocationDefault
	Detectors	DayaBayAD1(2,3,4)
	PmtTool	EsPmtEffectPulseTool
	RpcTool	EsIdealPulseTool
	FeeTool	EsIdealFeeTool
	FecTool	EsIdealFecTool
	MaxSimulationTime	50 us

Table 12.1: Algorithms and their properties.

Property	Default
CableSvcName	StaticCableSvc
SimDataSvcName	StaticSimDataSvc
TriggerWindowCycles	Dayabay::TriggerWindowCylces
NoiseBool	true
NoiseAmp	0.5mV

Table 12.2: Properties declared in EsIdealFeeTool.

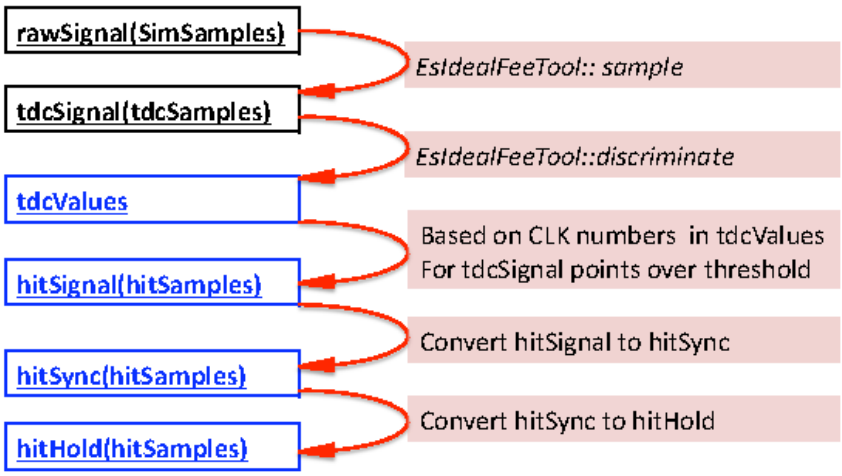

Figure 12.3: : hitHold signal generation sequence. Analog Signals are shown in the black box. And ditigal signals are shown in blue boxes. On the right hand side, related functions or comments are listed to specify the convertion between different signals.

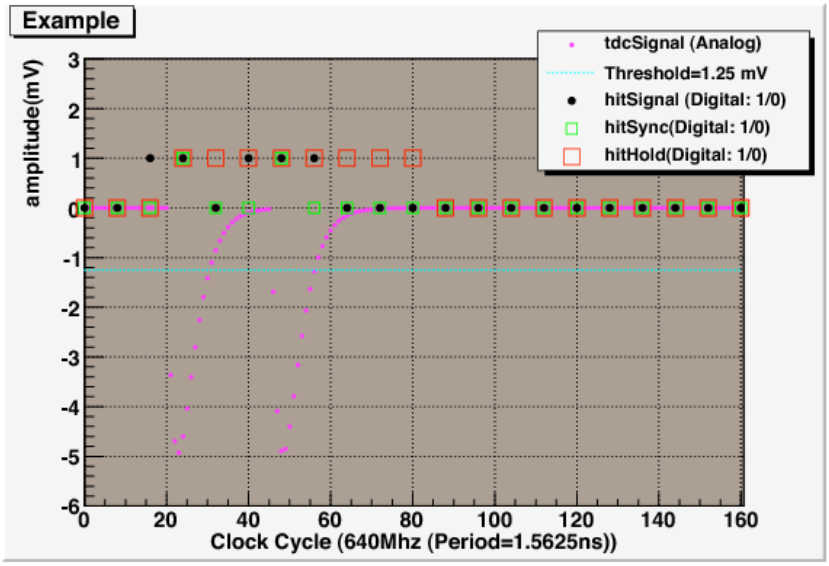

Figure 12.4: : An example of convertions from tdcSignal to hitHold Signal. The the label in Y axis is only for the analog signal tdcSignal and the Threshold line.

Variable Defined	Value
BaseFrequency	$40 \cdot 1E6$ (hz)
TdcCycle	16
AdcCycle	1
EsumCycle	5
NhitCycle	2
preTimeTolerance	300ns
postTimeTolerance	10us
TriggerWindowCycle	8

Table 12.3:

Chapter 13

Trigger Simulation

13.1 Introduction

The Trigger Simulation is implemented in the TrigSim package. `TrigSim` takes an `ElecHeader` as input and produces a `SimTrigHeader`. See Figure 13.1.

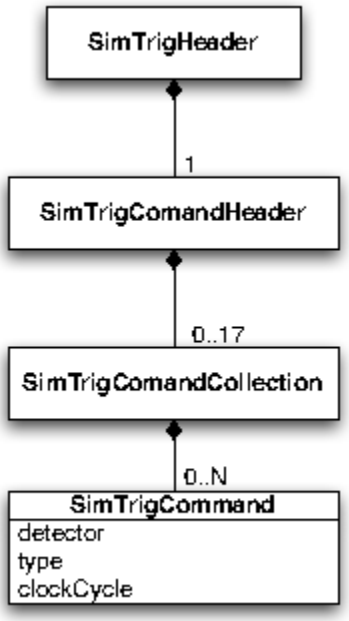

Figure 13.1: `SimTrigHeader` contains a single `SimTrigCommandHeader` which in turn potentially contains a `SimTrigCommandCollection` for each detector. Each `SimTrigCommandCollection` contains `SimTrigCommands` which correspond to an actual trigger.

13.2 Configuration

The main algorithm in TrigSim, `TsTriggerAlg` has 3 properties which can be specified by the user.

TrigTools `Default:` "TsMultTriggerTool" List of Tools to run.

TrigName `Default:` "TriggerAlg" Name of the main trigger algorithm for bookkeeping.

ElecLocation `Default:` "/Event/Electroincs/ElecHeader" Path of ElecSimHeader in the TES, currently the default is picked up from ElecSimHeader.h

The user can change the properties through the TrigSimConf module as follows:

```
import TrigSim
trigsim = TrigSim.Configure()
import TrigSim.TrigSimConf as TsConf
TsConf.TsTriggerAlg().TrigTools = [ "TsExternalTriggerTool" ]
```

The `TrigTools` property takes a list as an argument allowing multiple triggers to be specified. The user can apply multiple triggers as follows:

```
import TrigSim
trigsim = TrigSim.Configure()
import TrigSim.TrigSimConf as TsConf
TsConf.TsTriggerAlg().TrigTools = [ "TsMultTriggerTool" ,
                                    "TsEsumTriggerTool" ,
                                    "TsCrossTriggerTool" ]
```

The mutate method within each tool will be called once per event in the order in which they are listed.

13.3 Current Triggers

This section will describe specific trigger implementations. Most implementations will have properties which can be set like this:

INSERT EXAMPLE

13.3.1 TsMultTriggerTool

A Multiplicity Trigger implementation. This will issue a local trigger when a specified number of channels are go over threshold within a given time window. This tool has two properties:

DetectorsToProcess is a list of detectors for this trigger to work on. The default value for this property is a list containing all pmt based detectors. This tool loops over all detectors within the ElecHeader and checks it against those in the list. If the detector is in the list the tool issues all applicable triggers for that detector. If the detector is not found in the DetectorsToProcess list the detector is ignored.

RecoveryTime sets the number of nhit clock cycles to wait after a trigger is issued before potentially issuing another trigger. The default value is 24 which corresponds to 300ns for the 80MHz clock.

13.3.2 TsExternalTriggerTool

An External Trigger implementation. This will issue a local triggers at a specified frequency. Currently used with the dark rate module for the MDC08. The properties are:

DetectorsToProcess Same as TsMultTriggerTool in section 13.3.1.

TriggerOffset

Frequency

AutoSet

13.4 Adding a new Trigger

To add a new trigger type, create a new class which inherets from GaudiTool and ITsTriggerTool as shown here:

```
class TsMyTriggerTool : public GaudiTool,
                              virtual public ITsTriggerTool
{
 public:

  TsMyTriggerTool(const std::string& type,
                  const std::string& name,
                  const IInterface* parent);
  virtual ~TsMyTriggerTool();

  virtual StatusCode mutate(DayaBay::SimTrigHeader* trigHeader,
                            const DayaBay::ElecHeader& elecHeader);
  virtual StatusCode initialize();
  virtual StatusCode finalize();

 private:
  std::vector<std::string> m_detectorsToProcess;
};
```

Chapter 14

Readout

14.1 Introduction

`ReadoutSim` is located in `Simulation/ReadoutSim` within the dybgaudi project. It uses `SimTrigCommand`'s and `ElecCrate`'s to produce `readouts`. The produced `readouts` are held within a `SimReadoutHeader` object. An addition `ReadoutHeader` object exists to satify the requirement that only one (1) `readout` be produced each execution cycle. The details of the header objects are shown in figures 14.2 and 14.1

14.2 ReadoutHeader

The `ReadoutHeader` contains a single readout which consists of the following:

detector `Detector` uniquely identifying the subsystem that was readout to produce this object.

triggerNumber `unsigned int` enumerating triggers.

triggerTime `TimeStamp` of trigger issuance.

triggerType `TriggerType_t` enum which constructs a bitmap to define the trigger type.

readoutHeader A pointer back to the `ReadoutHeader` which contains this object.

Two flavors of Readouts exist, `ReadoutPmtCrate` and `ReadoutRpcCrate`. The `ReadoutPmtCrate` contains a map of `FeeChannelId`'s to `ReadoutPmtChannel`'s and the `ReadoutRpcCrate` contains a similar map of `FeeChannelId`'s to `ReadoutRpcChannel`'s. The `ReadoutPmtChannel` Contains

channelId `FeeChannelId` uniquely identifying the channel that was read out.

tdc a vector of tdc values

adc a map of adc values, keyed with their clock cycle.

adcGain `FeeGain_t` denoting either that the high or low gain was read out.

readout pointer back to the `ReadoutPmtCrate` which contains this channel readout.

The ReadoutRpcChannel contains

channelId `FeeChannelId` uniquely identifying the channel that was read out.

hit a boolean value indicating a hit.

readout a pointer back to the `ReadoutRpcCrate` which contains this channel readout.

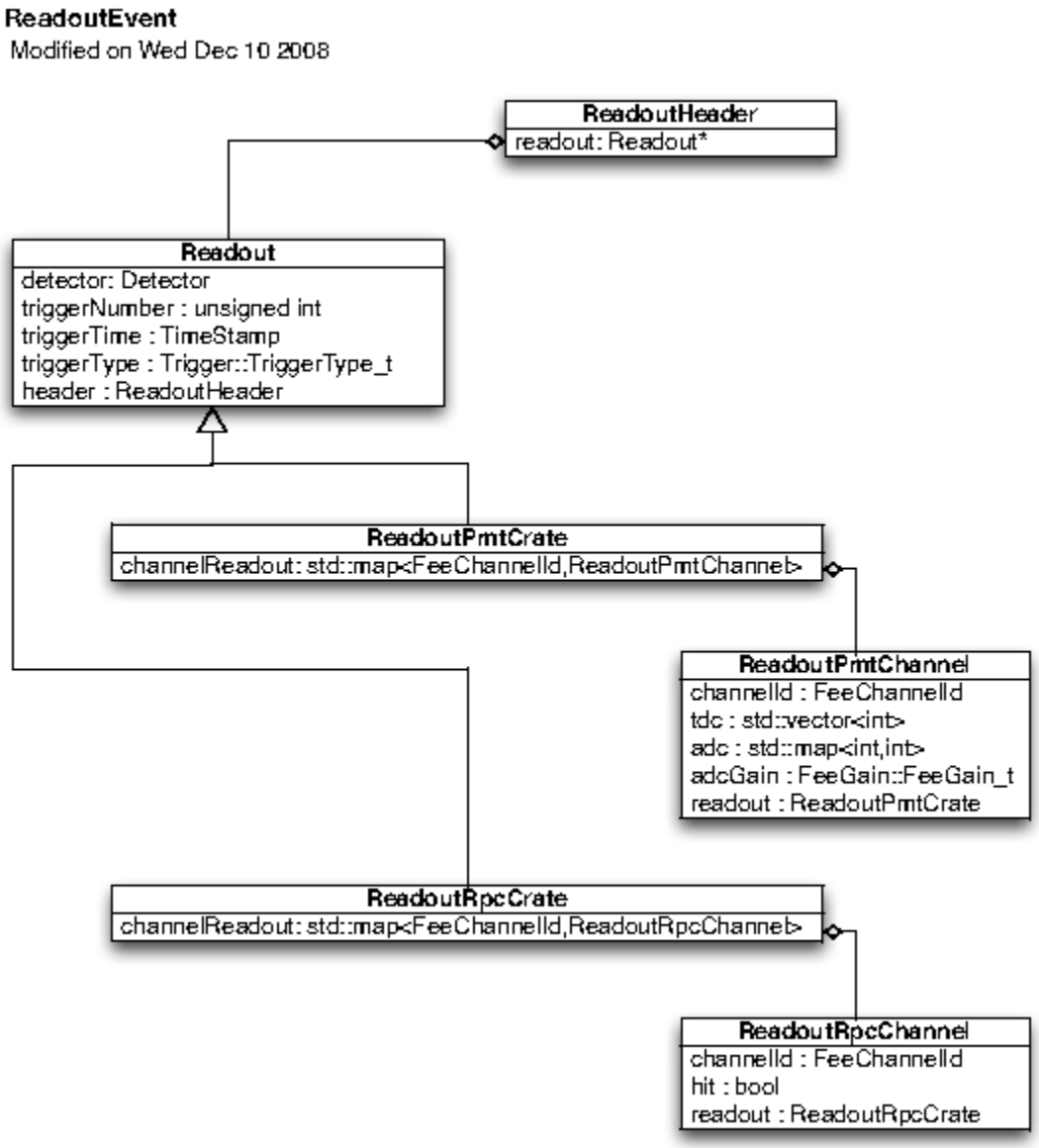

Figure 14.1: The `ReadoutHeader` contains a single Readout. The two flavors of readouts are discussed in 14.2

14.3 SimReadoutHeader

The `SimReadoutHeader` contains all the readout headers produced during a single execution cycle. This can include $0..N$ readouts for each detector.

14.4 Readout Algorithms

ReadoutSim currently has two Algorithms described below:

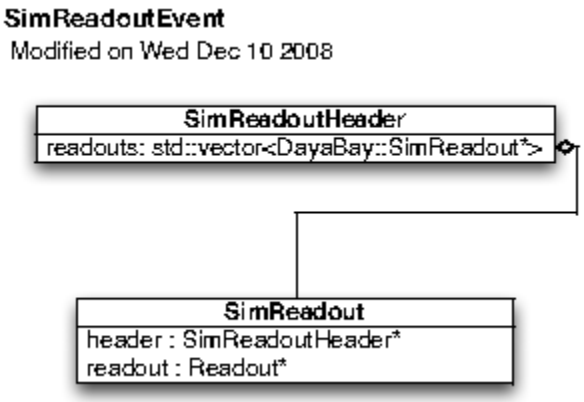

Figure 14.2: The `SimReadoutHeader` holds multiple `SimReadout`'s which in turn contain a pointer to a single `Readout` object. The `Readout` object pointer points to the same object a `SimReadoutHeader` points to.

14.4.1 ROsSequencerAlg

`ROsSequencerAlg` tries to fix the many-to-one, readouts to execution cycle mismatch. The sequencer fill the ReadoutHeader object with only the first ReadoutEvent produced during each execution cycle.

14.4.2 ROsReadoutAlg

`ROsReadoutAlg` is the driving algorithm for `ReadoutSim`. This algorithm applies each tool specified in the `RoTools` property for each trigger event. It is up to the tool to decide if it should act or not. The default setup is as follows:

```
import ReadoutSim
rosim = ReadoutSim.Configure()
import ReadoutSim.ReadoutSimConf as ROsConf
ROsConf.ROsReadoutAlg().RoTools=["ROsFecReadoutTool","ROsFeeReadoutTool"]
ROsConf.ROsReadoutAlg().RoName="ReadoutAlg"
ROsConf.ROsReadoutAlg().TrigLocation="/Event/SimTrig/SimTrigHeader"
ROsConf.ROsReadoutAlg().ElecLocation="Event/Elec/ElecHeader"
```

14.5 Readout Tools

ReadoutSim currently has 5 tools described below which can be used to customize readout.

14.5.1 ROsFeeReadoutTool

`ROsFeeReadoutTool` handles reading out pmt based detectors. By default this tool acts on all trigger commands associated with a pmt based detector. To specify different parameters for specific pmt based detectors create multiple instances of this tool and specify `DetectorsToProcess` appropriately in each. The default configuration is shown below.

```
import ReadoutSim.ReadoutSimConf as ROsConf
ROsConf.ROsFeeReadoutTool().DetectorsToProcess=["DayaBayAD1","DayaBayAD2",\
                        "DayaBayIWS","DayaBayOWS","LingAoAD1","LingAoAD2",\
                        "LingAoIWS","LingAoOWS","FarAD1", "FarAD2",\
```

```
                          "FarAD3","FarAD4", "FarIWS","FarOWS"]
    ROsConf.ROsFeeReadoutTool().AdcTool="ROsFeeAdcPeakOnlyTool"
    ROsConf.ROsFeeReadoutTool().TdcTool="ROsFeeTdcTool"
    ROsConf.ROsFeeReadoutTool().ReadoutLength=12
    ROsConf.ROsFeeReadoutTool().TriggerOffset=2
```

14.5.2 ROsFecReadoutTool

`ROsFecReadoutTool` handles reading out the rpc based detectors. By default this acts on all rpc based detectors. This is the only property currently available as seen below in the default setup.

```
    import ReadoutSim.ReadoutSimConf as ROsConf
    ROsConf.ROsFeeReadoutTool()=detectorsToProcess=[ "DayaBayRPC" ,\
                                                     "LingAoRPC" , "FarRPC" ]
```

14.5.3 ROsFeeAdcMultiTool

`ROsFeeAdcMultiTool` reads out samples the adc values in the readout window based on the readout window start. The user specifies the `ReadoutCycles` with 0 corresponding the adc value at the beginning of the readout window.

```
    ROsConf.ROsFeeReadoutTool().AdcTool="ROsFeeAdcMultiTool"
    ROsConf.ROsFeeAdcMultiTool().ReadoutCycles=[ 0 , 2 , 3 , 4 , 8 ]
```

14.5.4 ROsFeeAdcPeakOnlyTool

`ROsFeeAdcPeakOnlyTool` reads out the peak adc value in the readout window.

```
    ROsConf.ROsFeeReadoutTool().AdcTool="ROsFeeAdcPeakOnlyTool"
```

14.5.5 ROsFeeTdcTool

`ROsFeeTdcTool` readout the tdc values during the readout window. The user has the option to readout multiple tdc values but changing the `UseMultiHitTdc` property.

```
    ROsConf.ROsFeeReadoutTool().TdcTool="ROsFeeTdcTool"
    ROsConf.ROsFeeTdcTool().UseMultiHitTdc=False
    ROsConf.ROsFeeTdcTool().TdcResetCycles=True
```

Chapter 15

Simulation Processing Models

15.1 Introduction

To properly simulate Daya Bay experiment, events from different event classifications must be properly mixed with any overlapping in space and time properly handled. To do this is complex and so a simpler simulation that only considers a single event type at a time is also desired. The former model goes by the name of "pull mode" or "Fifteen minutes style" simulation. The latter is known as "push mode" or "linear style" simulation.

15.2 Fifteen

Fifteen package successfully extends gaudi frame work to another level. It makes use of many advance features of dybgaudi, like AES, inputHeaders and using Stage tool to handle data transfer. Fifteen package is designed to handle the max complexity in simulation. It has sophisticated consideration on all kinds of possible physics scenario, event time information handling and data management. After two years' usage and the feedback from users, it's already absorbed a lot of ideas, like mixing pre-simulated events and reusing, and has gone into a mature stage.

15.2.1 Quick Start

After you get into nuwa environment, you are ready to start to generate your own simulation sample. In /NuWa-trunk-dbg/NuWa-trunk/dybgaudi/Tutorial/Sim15/aileron, after type in nuwa.py -n50 -o fifteen.root -m "FullChainSimple -T SingleLoader" > log it will generate 50 readouts from IBD and K40 events.

15.2.2 Simulation Stage

Simulation is separated into a few stages: Kinematic, Detector, Electronic, TrigRead and SingleLoader. Kinematic stage generates kinematic information, including time, position, particle and its momentum, etc. Detector stage is to geant4 to do detector response simulation, like scattering, cerenkov and scintillation light. At the end it will generate hit number (P.E.) in each PMT and hit information on RPC. Electronic simulation convert these physics hit into electronic signal. For example, hits on PMT are converted to pulses. TrigRead will do trigger judgement based on user setting, like NHit¿10, which means number of fired PMTs must be above 10. When an event is triggered, it also produces readout. That means it will output ADC and TDC instead of a raw PMT pulse. The real data acquisition system works like a pipe line, it outputs its result one by one in time order. SingleLoader is designed for this purpose. The above description can be summarized in Fig. 15.1

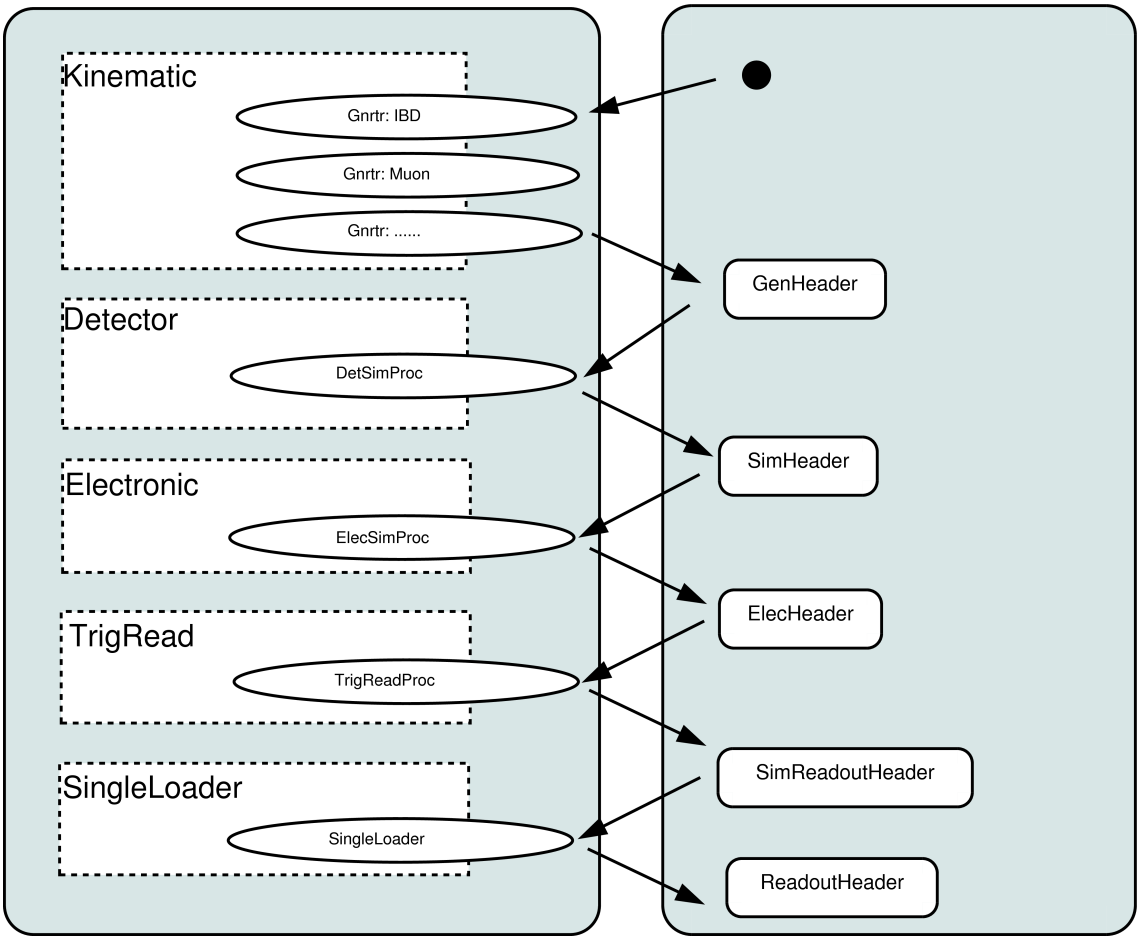

Figure 15.1: Simulation stages.

15.2.3 Stage Tool

Stage as explained in previous sections is an abstract concept in dividing all simulation components. For Fifteen package, stage tool physically separates each simulation tools, but also is a media in data transfer.

While synchronizing many generation sources, they generate many data in the same time – same execution cycle. For dybgaudi they are held by AES and inputHeaders. The time sequence in which they are generated is disordered. Stage tool is put in charge of managing all the processors in one simulation stage. it manages the execution of them, i.e. only run them when data is needed, and it caches the data from all processors, sorts them and output them in time order. A bad metaphor might be stage tool works like a central train station. It controls the incoming stream of all the trains to avoid possible crushing. It has some ability to let train stop for some period, then let them leave on time.

15.2.4 Gnrtr

Gnrtr stands for Generator. For one type of events one generator needs to be specified. The type here is not limited to its physics generation mechanism. The same type of event in different volume or geometry structure may have different event rates, so they should be specified as two different Gnrtr. For example a type of radioactive background have different abundance in two types of material, then it will have different

event rate.

While running Gnrtr will invoke each GenTools it owns, i.e. a real generator, timrator positioner, etc. User needs to specify all these tools for it.

15.2.5 DetSimProc

One of DetSimProc's main functions is to call the real simulation tool Geant4 through its Gaudi interface GiGa. The other important feature is to output each simheader in time order.

Imagine two GenHeaders' times are very close: the first one in time is far away to any PMTs, while the second one is close to one PMT, it is possible that because of the time of light propagation, light from the second event will generate a PMT hit first. The chance of this to happen is small, but it is serious enough to cause whole simulation process to crush and all the following electronic and trigger logic to fail.

DetSimProc asks data input from simulation stage "Kinematic". As promised by stage tool, all the kinematic information out of stage "Kinematic" are in time order, earliest to latest, no violation. Then DetSimProc take this advantage to ensure its output is also in time order. After DetSimProc got a GenHeader to simulate, it finished the detector simulation for that GenHeader first. That is it can know the earliest hit time of this SimHeader. DetSimProc keeps asking GenHeader from its lower stage and doing their detector simulation, until a time comparison test is success. DetSimProc caches all the information of processed GenHeaders and SimHeaders. It compares the earliest time of all SimHeaders and the time of the last GenHeader. When the time of a SimHeader is less than the last GenHeader, it claims safe for output for that SimHeader. Because the causality of event development, since the last GenHeader time is already bigger than the time of a previous SimHeader, any new simulated result SimHeader won't go before this GenHeader, i.e. the previous SimHeader.

15.2.6 ElecSimProc

ElecSimProc maintains a pipeline of SimHits which are sorted by time. Normal geant4 simulated PMT and RPC hits from all kinds of sources are kept in this pipeline.

The first thing to do every time execute ElecSimProc is to find a time gap between two successive hits in this hit pipeline. The size of the gap is determined by DayaBay::preTimeTolerance + DayaBay::postTimeTolerance which should be actually corresponding to the time period where a prepulse or a afterpulse exist. Then in the real electronics simulation, prepulses and afterpulse can be inserted into these places. Certainly as explained in previous sections, when a time gap is found, the time of the gap stop must be less the current time of detector simulation stage. This is the only way to know there won't be any hits from later simulation will fool into this gap.

The chunk of hits before the gap start are packed together and made a new hit collection, then sent to electronic simulation. So hits of all kinds of sources have a chance to mix and overlap. Electronics simulation tools will take over the job and each sub detector will process its part separately.

For each fast simualted MuonProphet muon, a fake hit is created and put into this pipeline. Instead of going into a full eletronics simulation, they are pushed into a fast electronics simulation. They are always 100 percent accepted even they didn't passed trigger. Since they are also in the pipeline, their time is synchronized to the other geant4 simulated hits. User won't obeserve a big delay between fast simulated muon and other events.

15.2.7 TrigReadProc

Trigger simulation and Readout simulation are combined together into one simulation stage, because they all needs input from electronic simulation, i.e. pulses information. In electronic simulation there is no such requirement that only some detector can join the simulation, so in the same way, trigger will work for all required detectors.

In principle the different delay from different electronic channel can flip the time order between different events, however the time gap requirement is at the scale of $10\mu s$. It is believed that the possible time flip

caused by electronic simulation will never go beyond that and there is no physics concern in simulating such a effect, so there is no complex time comparison in TrigReadProc.

15.2.8 SingleLoader

Triggers and readouts found in ElecHeader are packed into one SimReadoutHeader. Certainly it is also possible that no trigger is found, since there are many low energy background events. SingleLoader caches all the triggers and readouts and output them one by one. When its own buffer is empty it will automatically ask data from lower stage.

15.2.9 LoadingProc

The only chance that events of different type can overlap and produce some impact is in electronic simulation. Hits from different events which are close in time may not be distinguished in electronics. A correct mixing approaching with pre-simulated sample should happen before it goes into electronic simulation.

Another idea is to re-use some geant4 pre-simulated sample. Like for muon events, it has a high frequency and is extremely time-consuming. We care a lot more about its influence on its adjacent events than its own topology.

LoadingProc is created on this background. It accepts a pre-simulated file, which must contain SimHeaders, as an input stream and output them to Stage Detector tool.

At the same time it can be configured to reset the event rate, i.e. time of the generated events. It also simplify the process if any trigger or electronic simulation parameter needs to be adjusted, since don't have to waste time to redo the longest geant4 simulation.

15.2.10 Algorithm Sim15

Algorithm Sim15 is a simple Gaudi algorithm which is inserted into Gaudi top algorithm list. It runs once every execution cycle. It sends out the initial request for generating MC events.

15.2.11 Customize Your Simulation Job

A General Example

This part will explain how exactly to write your own simulation script with Fifteen package. The example is from dybgaudi/Tutorial/Sim15/aileron/FullChainSimple.py which implements all the basic elements.

```
#!/usr/bin/env python
'''
Configure the full chain of simulation from kinematics to readouts and
with multiple kinematics types mixed together.

usage:
    nuwa.py -n50 -o fifteen.root -m "FullChainSimple -T SingleLoader" > log

    -T: Optional stages are: Kinematic, Detector, Electronic, TrigRead or SingleLoader.

    More options are available like -w: wall clock starting time
                                    -F: time format
                                    -s: seed for IBD generator

    //////
    Aside:
    This is a copy of MDC09b.runIBD15.FullChain, however with less options,
    less generators configured and less truth info saved.
    //////
'''
```

This is the first part of this script. In the first line it declares the running environment. What follows, quoted by "', are a brief introduction of this script and usage of this script. It tells that this script will configure a full chain of simulation. It also includes a command line which can be used right away to start. Before looking into the script it also explains what arguments can be set and what are their options. These arguments will explained later.

Next I will follow the order of how this script is going to be executed in nuwa. Then it will bring us to the end of the script.

```
def configure(argv=[]):
    cfc = ConfigureFullChain(argv)
    cfc.configure()
    return

if __name__ == "__main__":
    configure()
    pass
```

A python script is executable in a shell environment when it has

```
if __name__ == "__main__":
```

Like this FullChainSimple.py, you can directly type FullChainSimple.py in a tcsh or bash see what happens. It is often used to test the configuration needed before running nuwa.

When nuwa is loading a python module it will check whether it has a configure() method. User's gaudi algorithms, services and tools' should go into there. Here an object about Fifteen is created and some parameters "argv" are passed to it. Next we will see some details in Fifteen package configuration.

```
class ConfigureFullChain:
    def __init__(self,argv):
        ...
    def parse_args(self,argv):
        ...
    def configureKinematic(self):
        ...
    def configureDetector(self):
        ...
    def configureElectronic(self):
        ...
    def configureTrigRead(self):
        ...
    def configureSingleLoader(self):
        ...
    def configureSim15(self):
        ...
    def configure(self):
        ...
```

Now all the details are stripped out, and only the skeleton are left. "..." indicates the real working code are omitted for a second. A class ConfigureFullChain is defined.

```
__init__(self,argv)
```

is always called when a data object is created. The useful interface invoked by nuwa will be $configure(self)$. Note don't confuse with the $configure(argv = [])$ mentioned previously.

Apparently it has configure functions for Kinematic, Detector, Electronic, TrigRead, SingleLoader simulation stages. It also can handle some parameters to be more user friendly in $parse_args$. The configureSim15 will create an algorithm called Sim15 which is the diver of the simulation job. Algorithm Sim15 sits on the top of all the simulation stages asking output.

Stage tools are firstly set up in the following.

```
    def configure(self):

        from Stage import Configure as StageConfigure
        self.stage_cfg = StageConfigure()

        stagedic={'Kinematic':1,'Detector':2,'Electronic':3,'TrigRead':4,'SingleLoader':5}
    ...

        if stagedic[self.opts.top_stage]>=1:
            self.configureKinematic()
        if stagedic[self.opts.top_stage]>=2:
            self.configureDetector()
        if stagedic[self.opts.top_stage]>=3:
            self.configureElectronic()
        if stagedic[self.opts.top_stage]>=4:
            self.configureTrigRead()
        if stagedic[self.opts.top_stage]>=5:
            self.configureSingleLoader()

        self.configureSim15()
```

According to the top simulation stage all required lower stage tools are created. For example if top stage is set to be Detector, then only stage tool Kinematic and Detector will be added. In the end the algorithm Sim15 is configured. Correspondingly Sim15 will ask data from stage tool Detector.

Next we will see the configuration of Gnrtr. In this example two generators, IBD and K40 are added to work at the same time.

```
    def configureKinematic(self):
        #IBD
        from Gnrtr.IBD import EvtGenerator
        # from IBD import EvtGenerator
        ibd_gds = EvtGenerator(name     = 'IBD_gds',
                               seed     = self.opts.seed,
                               volume   = '/dd/Structure/AD/db-oil1',
                               strategy = 'Material',
                               material = 'GdDopedLS',
                               mode     = 'Uniform',
                               lifetime = 78.4*units.second, #daya bay site
                               wallTime = self.start_time_seconds)

        ibd_gds.ThisStageName = "Kinematic"
        self.stage_cfg.KinematicSequence.Members.append( ibd_gds )

        from Gnrtr.Radioact import Radioact
        #K40
        k40_gds = Radioact(name       = 'K40_gds',
                           volume     = '/dd/Structure/AD/db-oil1',
                           nuclide    = 'K40',
                               abundance  = 3.01e17,
                           strategy   = 'Material',
                           material   = 'GdDopedLS',
                               start_time = self.start_time_seconds)

            k40_gds.ThisStageName = "Kinematic"
        self.stage_cfg.KinematicSequence.Members.append( k40_gds )
```

Basically only one line command is needed to specify one type of event. In the end their stage names are all assigned to be "Kinematic" and it generator algorithms are also added to stage tool Kinematic. i.e. the connection between stage tool and processors are built up. For details about generators' configuration user can refer to previous sections, and they also need to have the knowledge of detector geometry and material.

```
    def configureDetector(self):
        '''Configure the Detector stage'''

```

```
4         import DetSim
5         ds = DetSim.Configure(physlist=DetSim.physics_list_basic+DetSim.physics_list_nuclear,
6                               site="dayabay",
7                               use_push_algs = False)
8
9         # QuantumEfficiency*CollectionEfficiency*QEScale = 0.24*1/0.9
10        from DetSim.DetSimConf import DsPhysConsOptical
11        optical = DsPhysConsOptical()
12        #optical.UseScintillation = False
13        optical.CerenPhotonScaleWeight = 3.5
14        #optical.UseCerenkov = False
15        optical.ScintPhotonScaleWeight = 3.5
16
17        from DetSimProc.DetSimProcConf import DetSimProc
18        dsp = DetSimProc()
19        dsp.ThisStageName = "Detector"
20        dsp.LowerStageName = "Kinematic"
21        #dsp.OutputLevel = 2
22        self.stage_cfg.DetectorSequence.Members.append(dsp)
23
24        ds.historian(trackSelection="(pdg == 2112)",vertexSelection="(pdg == 2112)")
25        return
```

The above example shows how detector simulation part is configured. Usually DetSim works in a normal gaudi manner, here the option $use_push_algs = False$ will stop adding its to top algorithm list. The lines assigning stage names, lower stage and this stage, tells where the input data is from, and what the current stage is. Then this DetSimProc algorithm was added to the stage tool Detector.

In the rest the physics list is customized and both cerenkov and scintillation light are pre-scaled. From this example and the above one for generator it is already very obvious that Fifteen package just uses the simulation tools as others. It doesn't create another set of tools. All setting of them can be directly moved to here.

Next we will see how electronic simulation is set up.

```
1     def configureElectronic(self):
2         '''Configure the Electronics stage'''
3
4         import ElecSim
5         es = ElecSim.Configure(use_push_algs = False)
6
7         from ElecSimProc.ElecSimProcConf import ElecSimProc
8         esp = ElecSimProc()
9         esp.ThisStageName = "Electronic"
10        esp.LowerStageName = "Detector"
11        #esp.OutputLevel = 2
12        self.stage_cfg.ElectronicSequence.Members.append(esp)
13
14        from ElecSim.ElecSimConf import EsIdealFeeTool
15        feetool = EsIdealFeeTool()
16            feetool.EnableNonlinearity=False
17
18        return
```

There is nothing new here regarding about Fifteen package configuration, except that name of this stage is "Electronic" and lower stage is "Detector". The simulation chain is setup in this way.

Here a non-linearity option is turn off to demonstrate how to configure the real working tool.

For completeness the configuration of TrigReadProc and SingleLoader are included.

```
1     def configureTrigRead(self):
2         '''Configure the Trigger and Readout stage'''
3         from TrigReadProc.TrigReadProcConf import TrigReadProc
4         tsp = TrigReadProc()
5         tsp.ThisStageName = "TrigRead"
6         tsp.LowerStageName = "Electronic"
```

```
7         #tsp.TrigTools = [...]
8         #tsp.RoTools = [...]
9         #tsp.OutputLevel = 2
10        self.stage_cfg.TrigReadSequence.Members.append(tsp)
11        return
12
13    def configureSingleLoader(self):
14        '''Configure the SingleLoader stage'''
15        from SingleLoader.SingleLoaderConf import SingleLoader
16        sll = SingleLoader()
17        sll.ThisStageName = "SingleLoader"
18        sll.LowerStageName = "TrigRead"
19        #sll.OutputLevel = 2
20        self.stage_cfg.SingleLoaderSequence.Members.append(sll)
```

In the end the top pulling algorithm Sim15 is added to gaudi top algorithm list. Its only job is to bring up the initial request from top stage tool.

```
1     def configureSim15(self):
2         from Stage.StageConf import Sim15
3         sim15=Sim15()
4         sim15.TopStage=self.opts.top_stage
5
6         from Gaudi.Configuration import ApplicationMgr
7         theApp = ApplicationMgr()
8         theApp.TopAlg.append(sim15)
```

Example for LoadingProc

LoadingProc is another input stream for SimHeader. So the configuration of LoadingProc should be a replacement for configureDetector in the above example. A working example can be found in Fifteen/LoadingProc/aileron/testAll.py

Here the configuration after stage Detector will not be repeated. Only the part for LoadingProc is shown. In that example two input files are specified. Each one is set to a new start time and a new event rate. Details are shown below. As usual the chain of simulation line are set up and input file are specified as expected.

```
1     def configureLoadingProc(self):
2         from LoadingProc.LoadingProcConf import LoadingProc
3         load = LoadingProc("LoadingProc.Oxygen18")
4         load.StartSec = 0
5         load.StartNano = 0
6         #load.Distribution = "Exponential"
7         load.Distribution = "Periodic"
8         load.Rate = 1.0
9         assembler_name = "Ox18Assem"
10        load.HsAssembler = assembler_name
11        load.OutputLevel = 2
12            assem = Assembler(toolname = assembler_name,
13                              filename = "input.root")
14
15        # This and lower stage
16        load.ThisStageName = "Detector"
17        load.LowerStageName = ""
18        # Add this processor to Gaudi sequencer
19        self.stage_cfg.DetectorSequence.Members.append(load)
20        return
```

15.2.12 Reminders and Some Common Errors

AES must be used to use Fifteen to generate simulation sample. The number of events specified on the command line is the number of execution cycles. If asking readout as the final output, then the initial number of GenHeader varies depending on trigger efficiency.

Chapter 16

Reconstruction

Chapter 17

Database

17.1 Database Interface

This chapter is organized into the following sections.

Concepts is an introduction to the basic concepts behind the DatabaseInterface. You can skip this section if you are in a hurry, but reading it will help you understand the package.

Installing and Running provides a few tips on building running programs that use the DatabaseInterface.

Accessing Existing Tables tells you how you write code to retrieve data from existing tables.

Creating New Tables describes how new tables are added to the database and the corresponding classes, that serve the data, are designed.

Filling Tables explains how new data is added to existing tables in the database.

MySQL Crib gives the bare minimum necessary to use MySQL to manage a database. The DatabaseInterface runs directly on top ROOT under which MySql and flat ASCII files are used to implement a hierarchical database.

17.2 Concepts

17.2.1 Types of Data

Besides the data from the detector itself, off-line software requires additional types of data. Some possible examples:

Detector Description i.e. data that describes the construction of the detector and how it responds to the passage of particles through it. The geometry, the cabling map and calibration constants are all examples of this type of data.

Reactor Data i.e. reactor power, fuel makeup, or extrapolated neutrino spectra

Physics Data i.e. cross-section tables, optical constants, etc.

It is the purpose of the DatabaseInterface to provide simple and efficient access to such data and to provide a framework in which new types of data can be added with minimal effort.

17.2.2 Simple, Compound and Aggregated

Within the database, data is organised into tables. When the user requests data from a table, the DatabaseInterface collect rows of data from the appropriate table. From the perspective of the interface, there are 3 types of organisation:-

Simple A single row is retrieved. Algorithm Configuration data is always simple; even if multiple configurations are possible, only one can be selected at a time. Detector Description, on the other hand, is almost never Simple.

Compound Multiple rows are retrieved. Each row represents a single sub-system and the request retrieves data for a complete set of sub-systems. For example a request for PMT positions will produce a set of rows, one for each PMT.

Aggregated A special form of Compound depending on the way new data is added to the database:-

- If data for the entire detector is written as a single logical block, then it is Compound. A table that describes the way PMTs to electronics channels might be compound: a complete description is written as a single unit
- If it is written in smaller chunks (called aggregates) then it is Aggregated. For example, it might be possible to calibrate individual electronics cards independently of the rest of the detectors at on sit. When calibrated, you will want to update only a subset of the calibrations in the database. One of the jobs of the interface is to reassemble these aggregates so that the user only ever sees a complete set.

There are two types of aggregation:-

Complete In this type the number of aggregates present at any time is constant, with the possible exception of detector construction periods during which the number increases with time. This is the normal form and is used to describe a set of sub-systems that are permanently present e.g. the set of steel planes.

Sparse In this type the number of aggregates present at any time is variable, there could even be none. This form is used to describe abnormal conditions such as alarms.

17.2.3 Tables of Data

The DatabaseInterface provides a simple, uniform concept regardless of the data being accessed. Each request for data produces a pointer giving read access to a results table, which is effectively a slice of the underlying database table. Each row of the results table is an object, the type of which is table-specific. These table row objects give access to the data from one row but can hide the way the database table is organised. So changes to the physical layout of a database table should only effect its table row object, not the end users of the data. Note that a single request only ever accesses a single table; the interface does not support retrieval of data from multiple database tables simultaneously.

If the request for data fails for some reason, then the resulting table will be empty, otherwise it will have a single row for Simple organisation and more than one row for Compound and Aggregated. The user can ask how many rows the table has and can directly access any of them. The physical ordering of the rows in the table reflects the way the data was originally written, so for Aggregated data, the ordering is not optimised for retrieval. To deal with this, each table row object can declare a natural index, independent of its physical position, and this natural index can be used to retrieve data.

17.2.4 A Cascade of Databases

The DatabaseInterface can access data for more than one database. During initialisation it is given a list of database URLs. The list order reflects priority; the interface first looks for data in the first database in the list, but if that fails, tries the others in turn until all have been tried or data is found. This scheme allows a user to override parts of the official database by creating a mini-database with their own data and then placing it in the list ahead of the official database. The concept of a set of overlaying databases is called a cascade.

17.2.5 Context Sensitive

In principle, any of the data retrieved by the interface could depend on the the current event being processed. Clearly Detector Descriptions, such as calibration constants, will change with time and the interface has to retrieve the right ones for the current event. For this reason, all requests for data through the interface must supply information about the:-

- The type of data: real or Monte Carlo.
- The site of the detector: Daya Bay, Ling Ao, Mid, Far, or Aberdeen
- The date and times of the event.

Collectively this information is called the *Context* and is represented by the `Context` class of the Context package. Note that in common with event data and times

All Database date and times are in UTC.

In the database all data is tagged by a *Context Range* which identifies the types of data and detector and the ranges of date times for which it is valid. This is represented by the ContextRange class of the Context package. Some data is universal; the same database data can be used for any event. Others may be very specific to a single type of data and detector and a limited date time range.

Note that the Context Range of the data defines the context at for which the data will be accessed, NOT where data is generated. For example, reactor data will be associated with all detector sites, not assigned to a reactor site.

Physically, the way to associate the Context Range metadata with the actual data is to have a pair of tables:-

Context Range Table This table consists of rows of ContextRange objects, each with a unique sequence number which is used as a key into the Main Data Table.

Main Data Table Each row has a sequence number corresponding to an entry in the Context Range Table.

The interface first finds a match in the Context Range Table for the current context and then retrieves all rows in the Main Data Table that match its sequence number. The reasons for this two step approach are:-

- To simplify the task of Context Management.
- To avoid repeated data. For Compound and Aggregated data, many rows can share a single Context Range. So this range only appears once and only a simple sequence number has to be repeated in the main table.

17.2.6 Extended Context

The primary function of DatabaseInterface is to provide the best information for a specific context, but it can also retrieve information for much more general queries. The query is still broken into two parts: the "context" which is matched to the Context Range Table and then the data from the main table is taken for the selected sequence number(s). However the user can supply a context such as "All ranges that start between this time and that time" hence the term "Extended Context". Further, during the retrieval of data from the main table addition restrictions can be imposed. The result of an Extended Context query is a collection of rows that will not normally represent the state of the detector at a single moment in time and it is up to the user to interpret the results meaningfully. However, it does allow the user the power of raw SQL queries.

17.2.7 SimFlag Association

As explained in the preceding section, the interface finds the database data that best matches the context of the data. There are occasions when this matching needs to be changed, for example there can be times when Monte Carlo data needs to be treated exactly as if it were event data and this includes the way it retrieves from the database. To support this the user can specify, for any type of data, an associated list of data types. If this is done then, instead of using the current type, each of the alternative types are tried until a match is found. This matching takes precedence over the cascade i.e. all associated types are tried on the first database in the cascade before moving on to the second and subsequent cascade members. This ensures that higher members, which might even refer back to the ORACLE database at FNAL, are only tried as a last resort.

17.2.8 Authorising Databases and Global Sequence Numbers

As explained in the previous section, sequence numbers in the Context Range Table are unique. However this can present a problem if the same type of data is being entered into several different databases. For example calibration constants will be created in the Near, Far and Calibration detectors. Eventually the tables will be merged but it is essential that there is no conflict in the sequence numbers. To solve this problem, certain databases are special: they are able to produce globally unique sequences numbers. They do this as each is allocated a unique block of 10,000,000 sequence numbers (which is enough to allow a new entry to be made every minute for 20 years!). These blocks are recorded in a special table: `GLOBALSEQNO` that holds the last used sequence number for each table. The block 1..9,999,999 is used for local sequence numbers i.e. ones that are only guaranteed unique within the current database table.

By default permanent data written to an authorising database will be written with global sequence numbers. For temporary data, or if writing to a non-authorising database, local sequence numbers are used and in this case a `LOCALSEQNO` table is generated automatically if required.

Important:-

Merging database tables that have local sequence numbers will require a special procedure to avoid conflicts.

`GLOBALSEQNO` and `LOCALSEQNO` tables must never be propagated between databases.

17.2.9 Validity Management

For constants that change with time (if that is not a contradiction in terms!) it makes sense to have overlapping Context Ranges. For example, suppose we know that a certain sort of calibration constants drifts with time and that, once determined, is only satisfactory for the next week's worth of data. A sensible procedure would be to limit its validity to a week when writing to the database but to determine new constants every few days to ensure that the constants are always "fresh" and that there is no danger that there will be a gap. However, this means that the interface has to perform two types of Validity Management:-

Ambiguity Resolution When faced with two or more sets of data the interface has to pick the best. It does this simply by picking the one with the latest creation date time.

Context Range Trimming Having found the best set, the interface wants to know how long it will remain the best. Any set whose creation date is later will be better according to the above rule and so the retrieved data has its range trimmed so as not to overlap it. This reduced Context Range is called the *Effective Context Range*. This only happens in memory; the database itself is not modified, but it does mean that the interface does not need to check the database again for this set of data until the Effective Context Ranges has expired. This trimming also applies between databases in a cascade, with sets in higher priority databases trimming those in lower ones.

Overlay Version Dates As explained above, creation dates play a crucial role in resolving which set of data to use; later creation dates take priority over earlier ones. This scheme assumes that constants from earlier runs are created before constants from later runs, but this isn't always true. When improving e.g. calibration constants, it's quite normal to recalibrate recent runs before going back and fixing earlier ones and then, simply to use the date when the constants were created would mean that the constants from earlier runs would take priority over any later runs they overlapped. To allow constants to be created in any order the interface provides a system for deducing the best creation dates for any constants as follows:-

- A query is made using as the context, the start of the validity for the new constants.
- If the query finds no data, the creation date of the new constants is set to its validity start date.
- If the query finds data, the creation date of the new data is set to be 1 minute greater than the creation date of the found data i.e. just late enough to replace it.

The scheme means that creation dates always follow that dates of the runs that they correspond to rather than the dates when their constants were created. When using the scheme its probably better to consider the "dates" to be version numbers.

17.2.10 Rollback

The database changes almost constantly to reflect the state of the detector, particularly with regard to the calibration constants. However this can mean that running the same job twice can produce different results if database updates that have occurred between the two runs. For certain tasks, e.g. validation, its necessary to decouple jobs from recent updates and this requires database rollback i.e. restoring the database to a previous state. Rollback works by exploiting the fact that data is not, in general, ever deleted from the database. Instead new data is added and, by the rules of Ambiguity Resolution (see the previous section) supersede the old data. All data is tagged by the date it was inserted into the local database, so rollback is implemented by imposing an upper limit on the insertion date, effectively masking out all updates made after this limit.

17.2.11 Lightweight Pointers to Heavyweight Data

One of the interface's responsibilities is to minimise I/O. Some requests, particularly for Detector Configuration, can pull in large amounts of data but users must not load it once at the start of the job and then use it repeatedly; it may not be valid for all the data they process. Also multiple users may want access to the same data and it would be wasteful for each to have their own copy.

To deal with both of the above, the interface reuses the concept of a handle, or proxy, that appears in other packages such as Candidate. The system works as follows:-

1. When the user wants to access a particular table they construct a table-specific pointer object. This object is very small and is suitable to be stack based and passed by value, thus reducing the risk of a memory leak.

2. During construction of the pointer, a request for data is passed down through the interface and the results table, which could be large, is created on the heap. The interface places the table in its cache and the user's pointer is attached to the table, but the table is owned by the interface, not the user.

3. Each request for data is first sent to the cache and if already present then the table is reused.

4. Each table knows how many user pointers are connected to it. As each pointer is discarded by its owner, it disconnects itself from the table it points to.

5. Once a table has no pointers left it is a candidate for being dropped by its cache. However this is not done at once as, between events, there are likely to be no user pointers, so just because a table is not currently being pointed to, it doesn't mean that it won't be needed again.

17.2.12 Natural Table Index

For Detector Description data, tables can be large and the user will require direct access to every row. However, the way the table is arranged in memory reflects the way the data was originally written to the database. For Simple and Compound data the table designer can control this organisation as complete sets are written as a single unit. For Aggregated data, the layout reflects the way aggregates are written. This allows the interface to replace individual aggregates as their validity expires. However this means that the physical layout may not be convenient for access. To deal with this table row objects, which all inherit from `DbiTableRow` are obliged to return a Natural Table Index, if the physical ordering is not a natural one for access. Tables can then be accessed by this index.

17.2.13 Task

Task will provide a way to further select the type of data retrieved. For example:-

- There might be nominal set of geometry offsets, or a jittered geometry to test for systematic effects.
- Detector Configuration data could have two tasks, one for raw calibration and another for refined calibration.

The aim is that Task will allow a particular database table to be sub-divided according to the mode of use. Currently Task is a data type defined in Dbi i.e. `Dbi::Task` and is implemented as an integer. The default value is zero.

17.2.14 Sub-Site

Sub-Site can be used like the Task to disambiguate things at a single site. For example, this can be used to distinguish between antineutrino detector modules, between electronics crates, etc.

Currently SubSite is a data type defined in Dbi i.e. `Dbi::SubSite` and is implemented as an integer. The default value is zero.

17.2.15 Level 2 (disk) Cache

Loading a large table from the database is a lot of work:-

1. The query has to be applied and the raw data loaded.

2. The row objects have to be individually allocated on the heap.

3. Each data word of each row object has to be individually converted through several layers of the support database software from the raw data.

Now as the detector configuration changes slowly with time identically the same process outlined above is repeated many times, in many jobs that process the data, so the obvious solution is to cache the results to disk in some way that can be reloaded rapidly when required. The technique essentially involves making an image copy of the table to disk. It can only be applied to some tables, but these include the Calibration tables which represent the largest database I/O load, and for these tables loading times can be reduced by an order of magnitude.

17.3 Running

17.3.1 Setting up the Environment

The interface needs a list of Database URLs, a user name and a password. This was previously done using envvars `ENV_TSQL_URL`, `ENV_TSQL_USER`, `ENV_TSQL_PSWD` that directly contained this configuration information. As this approach resulted in the configuration information being duplicated many times a new DBCONF approach has now been adopted.

The DBCONF approach is based on the standard mysql configuration file `HOME/.my.cnf` which has the form :

```
[testdb]

host = dybdb1.ihep.ac.cn
user = dayabay
password = youknowit
database = testdb

[dyb_cascade]
host = dybdb1.ihep.ac.cn
user = dayabay
password = youknowit
database =

db1 = offline_db
db2 = dyb_temp
```

Typical configurations can be communicated via the setting of a single environment variable `DBCONF` that points to a named section in the configuration file. Other envvars can also be used to change the default behaviour allowing more complex configurations such as cascades of multiple databases to be configured.

envvar	default	notes
DBCONF		name of section in config file
DBCONF_URL	mysql://%(host)s/%(database)s	
DBCONF_USER	%(user)s	
DBCONF_PSWD	%(password)s	
DBCONF_HOST	%(host)s	
DBCONF_DB	%(database)s	
DBCONF_PATH	/etc/my.cnf:$SITEROOT/../.my.cnf: /.my.cnf	list of config file paths

The defaults are python patterns that are filled in using the context variables obtained from the section of the config

The meanings are as follows.

DBCONF_PATH Colon delimited list of paths (which can include envvars such as `$SITEROOT` and the home directory tilde symbol). Non-existing paths are silently ignored and sections from the later config files override sections from prior files. Using the default paths shown in the table allows the system administrator to manage config in `/etc/my.cnf` which is overridden by the dybinst administrator managed `$SITEROOT/../.my.cnf`.

Users only need to create their own config file in `HOME/.my.cnf` if they need to override the standard configuration.

DBCONF_URL This is a semi-colon separated list of URLs. Each URL takes the form:-

```
protocol://host[:port]/[database][?options]
where:
   protocol - DBMS type , e.g. mysql etc.
   host - host name or IP address of database server
   port - port number
   database - name of database
   options - string key=value's separated by ';' or '&'
 Example:
 "mysql://myhost:3306/test?Trace=Yes;TraceFile=qq.log"
```

DBCONF_USER Pattern that yields database user name. Only needs to be set if you require different names for different databases in the cascade then this can be a semi-colon separated list in the same order as DBCONF_URL. If the list is shorter than that list, then the first entry is used for the missing entries.

DBCONF_PSWD Pattern that yields database password. As with `DBCONF_USER` it can be a semi-colon separated list with the first entry providing the default if the list is shorter than `DBCONF_URL`. It only needs to be set if you require different passwords for the different databases in a cascade. Security risks are avoided by never using actual passwords in this envvar but rather using a pattern such as `%(pass1)s;%(pass2)s` that will be filled in using the parameters from the config file section identified by `DBCONF`. Setting it to null will mean that it will be prompted for when the interface initializes.

These variable should be set for the standard read-only configuration. These variables can be trivially overridden for specific jobs by resetting the environment variables in the python script:
Note that using setdefault allows the config to be overridded without editing the file :

```
import os
os.environ.setdefault('DBCONF','dyb_offline')
print 'Using Database Config %s ' % os.environ['DBCONF']
```

For framework jobs when write-access to the database is required, or other special configuration is desired a less flexible approach is preferred. With a comment pointing out that some special configuration in /.my.cnf is required. Be careful not to disclose real passwords; passwords do not belong in repositories.

```
"""
    NB requires section of ~/.my.cnf

    [dyb_offline]
    host = dybdb1.ihep.ac.cn
    user = dayabay
    password = youknowit
    db1 = dyb_offline
    db2 = dyb_other

"""
import os
os.environ['DBCONF'] = 'dyb_offline'
os.environ['DBCONF_URL'] = 'mysql://%(host)s/%(db1)s;mysql://%(host)s/%(db2)s'
print 'Using Database Config %s ' % os.environ['DBCONF']
```

17.3.2 Configuring

The database can be configured through a Gaudi Service before starting your job.

Once the job is running you can configure the DatabaseInterface via the DbiSvc:

```
from gaudimodule import *
theApp = AppMgr()
theApp.Dlls += ['Conventions']
theApp.Dlls += ['Context']
theApp.Dlls += ['DatabaseInterface']
theApp.createSvc('DbiSvc')

dbisvc = theApp.service('DbiSvc')
dbisvc.<property>=<newvalue>
dbisvc.<property>=<newvalue>
...
```

Rollback

To impose a global rollback date to say September 27th 2002:-

```
theApp.service('DbiSvc').RollbacDates ='* = 2002-09-27 00:00:00'
```

This will ensure that the interface ignores data inserted after this date for all future queries. The hours, minutes and seconds can be omitted and default to 00:00:00.

Rollback can be more selective, specifying either a single table or a group of tables with a common prefix. For example:-

```
theApp.service('DbiSvc').RollbackDates ='*          =  2002-09-01';
theApp.service('DbiSvc').RollbackDates ='Cal*       =  2002-08-01'
theApp.service('DbiSvc').RollbackDates ='CalPmtGain =  2002-07-01'
```

Now the table CalPmtGain is frozen at July 2002, other Cal tables at August and all other tables at September. The ordering of the commands is not important; the interface always picks the most specific one to apply to each table.

> *Rollback only applies to future queries, it does not invalidate any existing query result in the cache which are still available to satisfy future requests. So impose rollback conditions at the start of the program to ensure they apply consistently.*

MakeConnectionsPermanent

By default the DatabaseInterface closes connection to the database between queries, to minimise use of resources - see section 17.9.1. If the job is doing a lot of database I/O, for example creating calibration constants then this may degrade performance in which case all connections can be made permanent by:-

```
theApp.service('DbiSvc').MakeConnectionsPermanent='true'
```

Ordering Context Query Results

By default when the DatabaseInterface retrieves the data for a Context Query, it does not impose an order on the data beyond requiring that it be in sequence number order. When an ordering is not imposed, the database server is under no obligation to return data in a particular order. This means that the same job running twice connected to the same database could end up with result sets that contain the same data but with different ordering. Normally this doesn't matter, the ordering of rows is not significant. However, results from two such jobs may not be identical as floating point calculations can change at machine level precision if their ordering is changed. There are situations where it is required that the results be identical. For example:-

- When bug hunting.
- When checking compatibility between two databases that should be identical.

and for such occasions it is possible to completely specify the ordering of rows within a sequence number by forcing sub-ordering by ROW_COUNTER, a column that should be present in all Main Data tables:-

```
theApp.service('DbiSvc').OrderContextQuery='true'
```

Level 2 Cache

Enabling the Level 2 Cache allows certain large tables query results to be written to disk from which they can be reloaded by subsequent jobs saving as much as an order of magnitude in load time. Data in the cache will not prevent changes in the database from taking affect for the DatabaseInterface does an initial (lightweight) query of the database to confirm that the data in the cache is not stale. To enable the cache, the user specifies a directory to which they have read/write access. For example, to make the current working directory the cache:-

```
theApp.service('DbiSvc').Level2Cache='./'
```

Cache files all have the extension .dbi_cache. Not all tables are suitable for Level 2 caching; the DatabaseInterface will only cache the ones that are.

Cache files can be shared between users at a site to maximise the benefit. In this case the local Database Manager must set up a directory to which the group has read/write access. Management is trivial, should the cache become too large, it can simply be erased and then the next few jobs that run will re-populate it with the currently hot queries.

Note that Cache performance is achieved by doing raw binary I/O so the cache files are platform specific, so if running in a heterogeneous cluster the Database Manager should designate a platform specific directory. To simplify this, the name of the directory used by the cache can include environmental variables e.g.:-

```
theApp.service('DbiSvc').Level2Cache='$DBI_L2CACHE'
```

Output Level

The verbosity of the error log from the DatabaseInterface can be controlled by:

```
theApp.service('DbiSvc').OutputLevel = 3
```

The output levels are standard Gaudi levels.

17.4 Accessing Existing Tables

17.4.1 Introduction

To access database data, the user specifies the database table to be accessed and supplies a "context" for the query. The context describes the type and date time of the current event. This is stored in a Context package `Context` object.

FIXME: Need a description here of how to get a Context from a Data Model object.
It should be something like:

```
Context   GetContext() const
```

methods to get their context. The DatabaseInterface uses the context to extract all the rows from the database table that are valid for this event. It forms the result into a table in memory and returns a object that acts like a pointer to it.

You are NOT responsible for deleting the table; the Database Interface will do that when the table is no longer needed

You have random access to any row of the results table. Each row is an object which is specific to that table. The key to understanding how to get data from a database table is study the class that represent a row of it results table.

17.4.2 Accessing Detector Descriptions

Making the Query

As explained above, the key to getting data is to locate the class that represents one row in a database table. To understand how this all works look at one of the sample tables included in the DbiTest package and imaginatively called `DbiDemoData1, DbiDemoData2` and `DbiDemodata3`. For purposes of illustration we will pick the first of these. Its header can be found in:-

```
DbiTest/DbiDemoData1.h
```

To make a query you create a `DbiResultPtr` object. Its header can be found in:-

```
DatabaseInterface/DatabaseInterface/DbiResultPtr.h
```

This is a class that is templated on the table row class, so in this case the instantiated class is:-

```
DbiResultPtr<DbiDemoData1>
```

and to instantiate an object of this class you just need a `Context` object. Suppose vc is such an object, then this creates the pointer:-

```
DbiResultPtr<DbiDemoData1> myResPtr(vc);
```

This statement creates a DbiResultPtr for DbiDemoData1 class. First it searches through the database for all DbiDemoData1 objects that are valid for vc, then it assembles them into a table and finally passes back a pointer to it. Not bad for one statement! The constructor can take a second argument:-

```
 DbiResultPtr(Context vc,Dbi::SubSite subsite=0,Dbi::Task task=0);
```

Dbi::SubSite is an optional parameter that sub-divides a table to select a specific component at a given detector Site, e.g. an antineutrino detector.

Dbi::Task offers a way to sub-divided a table according to the mode of operation. For example a Detector Configuration data could have two modes, one for raw calibration and another for refined calibration.

If the concept of a subsite or task is not relevant for a particular database table, then the parameter should be left at its default value of 0. Otherwise data should be allocated a unique positive number and then selection will only pick rows with the required value of task.

The constructor can take further arguments which can normally be left at their default values - a `Dbi::AbortTest` see section 17.4.4 and a `Bool_t findFullTimeWindow` see section 17.9.2.

Accessing the Results Table

Having got a pointer to the table the first thing you will want to know is how many rows it has. Do this using the method:-

```
UInt_t GetNumRows() const;
```

If the query failed then the number of rows returned will be zero. This could either be the result of some catastrophic failure, for example the database could not be opened, or simply that no appropriate data exists for the current event. If you want to know which of these it is you can use the:-

```
const DbiValidityRec* GetValidityRec() const;
```

If this returns a null pointer, then the failure was a major one, see 17.4.4. If not then the `DbiValidityRec` tells you about the validity of the gap. Its method:-

```
 const ContextRange& GetContextRange() const;
```

returns a Context package `ContextRange` object that can yield the start and end times of the gap. Due to the way the DatabaseInterface forms the query, this may be an underestimate, but never an overestimate.

If the table has rows then the GetContextRange() will give you an object that tells you the range of the data. Again, the range may be an underestimate. To get to the data itself, use the method:-

```
const T* GetRow(UInt_t i) const;
```

where T = DbiDemoData1 in this case. This gives you a const pointer to the i^{th} row where i is in the range $0 <= i <$ GetNumRows().

FIXME Need complete example here including DataModel object.

Putting this all together, suppose you have a `CandDigitListHandle` object cdlh, and you want to loop over all `DbiDemoData1` objects that are valid for it, the code is:-

```
DbiTest/DbiDemoData1.h
DatabaseInterface/DbiResultPtr.h

...

DbiResultPtr<DbiDemoData1> myResPtr(cdlh.GetContext());

for ( UInt_t irow = 0; irow < myResPtr.GetNumRows(); ++ires) {
  const DbiDemoData1* ddd1 = myResPtr.GetRow(irow);

// Process row.

}
```

GetRow is guaranteed to return a non-zero pointer if the row number is within range, otherwise it returns zero. The ordering of rows reflects the way the data was written to the database. For some types of data this layout is not well suited for access. For example, for pulser data, all the strip ends illuminated by an LED will appear together in the table. To deal with this table row object are obliged to return a Natural Table Index, if the physical ordering is not a natural one for access. You get rows from a table according to their index using the method:-

```
const T* GetRowByIndex(UInt_t index) const;
```

You should always check the return to ensure that its non-zero when using this method unless you are absolutely certain that the entry must be present.

Getting Data from a Row

Having got to the table row you want, the last job is to get its data. Its up to the table row objects themselves to determine how they will present the database table row they represent. In our example, the `DbiDemoData1` is particularly dumb. Its internal state is:-

```
    Int_t    fSubSystem;
    Float_t  fPedestal;
    Float_t  fGain1;
    Float_t  fGain2;
```

which it is content to expose fully:-

```
    Int_t GetSubSystem() const { return fSubSystem; }
  Float_t GetPedestal() const { return fPedestal; }
  Float_t GetGain1() const { return  fGain1; }
  Float_t GetGain2() const { return fGain2; }
```

Its worth pointing out though that it is the job of the table row object to hide the physical layout of the database table and so shield its clients from changes to the underlying database. Its just another example of data encapsulation.

Making Further Queries

Even though a `DbiResultPtr` is lightweight it is also reusable; you can make a fresh query using the `NewQuery` method:-

```
UInt_t NewQuery(Context vc, Dbi::Task task=0);
```

which returns the number of rows found in the new query. For example:-

```
DbiResultPtr<DbiDemoData1> myResPtr(vc);
...
Context newVc;
...
myResPtr.NewQuery(newVc);
...
```

Having made a query you can also step forwards or backwards to the adjacent validity range using the method:-

```
UInt_t NextQuery(Bool_t forwards = kTRUE);
```

supply a false value to step backwards. This method can be used to "scan" through a database table, for example to study calibration constants changes as a function of time. To use this efficiently you need to request accurate validity ranges for your initial query, although this is the default see section 17.9.2. For aggregated data stepping to a neighbouring range will almost certainly contain some rows in common unless all component aggregates have context ranges that end on the boundary you are crossing. See the next section for a way to detect changes to data using the `DbiResult::GetID()` method.

Simple Optimisation

The first, and most important, level of optimisation is done within the DatabaseInterface itself. Each time it retrieves data from the database it places the data in an internal cache. This is then checked during subsequent queries and reused as appropriate. So the first request for a large table of calibration constants may require a lot of I/O. However the constants may remain valid for an entire job and in which case there is no further I/O for this table.

Although satisfying repeat requests for the same data is quick it still requires the location of the appropriate cache and then a search through it looking for a result that it is suitable for the current event. There are situations when even this overhead can be a burden: when processing many rows in a single event. Take for example the procedure of applying calibration. Here every digitization needs to be calibrated using its corresponding row in the database. The naive way to do this would be to loop over the digits, instantiating a `DbiResultPtr` for each, extracting the appropriate row and applying the calibration. However it would be far more efficient to create a little calibration object something like this:-

```
class MyCalibrator {
public:
  MyCalibrator(const Context vc): fResPtr(vc) {}
  Float_t Calibrate(DataObject& thing) {
     /*  Use fResPtr to calibrate thing  */
}

private
  DbiResultPtr<DbiDemoData1> fResPtr;
};
```

`MyCalibrator` is a lightweight object holding only a pointer to a results table. It is created with a `Context` object which it uses to prime its pointer. After that it can be passed `DataObject` objects for which it returns calibrated results using its Calibrate method. Now the loop over all digitizations can use this object without any calls to the DatabaseInterface at all. Being lightweight `MyCalibrator` is fine as a stack object, staying in scope just long enough to do its job.

Another optimisation strategy involves caching results derived from a query. In this case it is important to identify changes in the query results so that the cached data can be refreshed. To aid this, each `DbiResult` is given an key which uniquely identifies it. This key can be obtained and stored as follows:-

```
DbiResultKey MyResultKey(myResPtr.GetKey());
```

This should be stored by value (the DbiResultKey pointed to by GetKey will be deleted when the results expire) as part of the cache and checked each time a change is possible:-

```
if ( ! MyResultKey.IsEqualTo(myResPtr.GetKey()) ) {

  // recreate the cache data ...

  MyResultKey = *myResPtr.GetKey();
}
```

Caution: This tests to see that the current `DbiResult` has exactly the same data as that used when the cached was filled, but not that it is physically the same object. If there have been intervening queries the original object may have been deleted but this should not matter *unless the cache holds pointers* back to the `DbiResult`. In this case the result ID should be used. Initialise with:-

```
Int_t MyResultID(myResPtr.GetResultID());
```

and then check as follows:-

```
if ( MyResultID != (myResPtr.GetResultID()) ) {

  // recreate the cache data ...

   MyResultID = myResPtr.GetResultID();
}
```

17.4.3 Extended Context Queries

Making the Query

The constructor of a `DbiResultPtr` for an Extended Context Query is:-

```
   DbiResultPtr(const string& tableName,
                const DbiSqlContext& context,
                const Dbi::SubSite& subsite = Dbi::kAnySubSite,
                const Dbi::Task& task = Dbi::kAnyTask,
                const string& data = "",
                const string& fillOpts = "",
```

Dealing with each of these arguments in turn:-

const string& tableName The name of the table that is to be accessed. This allows any type of `DbiTableRow` to be loaded from any type of table, but see section 17.6 on filling if you are going to play tricks!

const DbiSqlContext& context This argument provides the extended context through the utility class `DbiSqlContext`. Consider the following code:-

```
// Construct the extended context: FarDet data that starts on Sept 1 2003.
// (note: then end time stamp is exclusive)
TimeStamp tsStart(2003,9,1,0,0,0);
TimeStamp   tsEnd(2003,9,2,0,0,0);
DbiSqlContext context(DbiSqlContext::kStarts,tsStart,
                      tsEnd,Site::kFar,SimFlag::kData);
```

You supply the type of context (in this case `DbiSqlContext::kStarts`), the date range and the detector type and sim flag. Other types of context are kEnds and kThroughout. See

```
DatabaseInterface/DbiSqlContext.h
```

for the complete list.

You are not limited to the contexts that `DbiSqlContext` provides. If you know the SQL string you want to apply then you can create a DbiSqlContext with the WHERE clause you require e.g.:-

```
DbiSqlContext myContext("SITEMASK & 4")
```

which would access every row that is suitable for the CalDet detector.

const Dbi::Task& task The task is as for other queries but with the default value of:-

```
Dbi::kAnyTask
```

which results in the task being omitted from the context query and also allows for more general queries: anything that is is valid after the **where** is permitted. For example:-

```
DbiSqlContext myContext("versiondate > '2004-01-01 00:00:00' "
                        " order by versiondate limit 1");
```

The SQL must have a where condition, but if you don't need one, create a dummy that is always true e.g.:-

```
DbiSqlContext myContext("1 = 1 order by timeend desc limit 1 ")
```

const string& data This is an SQL fragment, that if not empty (the default value) is used to extend the WHERE clause that is applied when querying the main table. For example consider:-

```
DbiSqlContext context(DbiSqlContext::kStarts,tsStart,tsEnd,
                      Site::kFar,SimFlag::kData);
DbiResultPtr<DbuSubRunSummary>
                      runs("DBUSUBRUNSUMMARY",context,
                           Dbi::kAnyTask,"RUNTYPENAME = 'NormalData'");
```

This query reads the DBUSUBRUNSUMMARY table, and besides imposing the context query also demands that the data rows satisfies a constraint on RUNTYPENAME.

const string& fillOpts This is a string that can be retrieved from `DbiResultSet` when filling each row so could be used to program the way an object fills itself e.g. by only filling certain columns. The DatabaseInterface plays no part here; it merely provides this way to communicate between the query maker and the the author of the class that is being filled.

Accessing the Results Table

Accessing the results of an Extended Context query are essentially the same as for a standard query but with following caveats:-

- If the method:-

  ```
  const DbiValidityRec* GetValidityRec(const DbiTableRow* row=0) const;
  ```

  is used with the default argument then the "global validity" of the set i.e. the overlap of all the rows is returned. Given the nature of Extended Queries there may be no overlap at all. In general it is far better to call this method and pass a pointer to a specific row for in this case you will get that validity of that particular row.

- The method:-

  ```
  const T* GetRowByIndex(UInt_t index) const;
  ```

  will not be able to access all the data in the table if two or more rows have the same Natural Index. This is prohibited in a standard query but extended ones break all the rules and have to pay a price!

17.4.4 Error Handling

Response to Errors

All `DbiResultPtr` constructors, except the default constructor, have a optional argument:-

```
Dbi::AbortTest abortTest = Dbi::kTableMissing
```

Left at its default value any query that attempts to access a non-existent table will abort the job. The other values that can be supplied are:-

kDisabled Never abort. This value is used for the default constructor.

kDataMissing Abort if the query returns no data. Use this option with care and only if further processing is impossible.

Currently aborting means just that; there is no graceful shut down and saving of existing results. You have been warned!

Error Logging

Errors from the database are recorded in a `DbiExceptionLog`. There is a global version of that records all errors. The contents can be printed as follows:-

```
#include "DatabaseInterface/DbiExceptionLog.h"
...
LOGINFO(mylog) << "Contents of the Global Exception Log: \n"
          <<  DbiExceptionLog::GetGELog();
```

Query results are held in a `DbiResult` and each of these also holds a `DbiExceptionLog` of the errors (if any) recorded when the query was made. If `myResPtr` is a `DbiResultPtr`, then to check and print associated errors:-

```
const DbiExceptionLog& el(myResPtr.GetResult()->GetExceptionLog());
if ( el.Size() == 0 ) LOGINFO(mylog) << "No errors found" << endl;
else                  LOGINFO(mylog) << "Following errors found" << el << endl;
```

17.5 Creating New Tables

17.5.1 Choosing Table Names

The general rule is that a table name should match the `DbiTableRow` subclass object that it is used to fill. For example the table CalPmtGain corresponds to the class `CalPmtGain`. The rules are

- Use only upper and lower case characters
- Avoid common names such as VIEW and MODE are used by ORACLE. A good list of names to avoid can be found at:-

  http://home.fnal.gov/%7Edbox/SQL_API_Portability.html

These restrictions also apply to column names. Moreover, column names should be all capital letters.

17.5.2 Creating Detector Descriptions

A Simple Example

Creating new Detector Descriptions involves the creation of a database table and the corresponding table row Class. The main features can be illustrated using the example we have already studied: DbiDemoData1. Recall that its state data is:-

```
    Int_t  fSubSystem;
  Float_t  fPedestal;
  Float_t  fGain1;
  Float_t  fGain2;
```

Its database table, which bears the same name, is defined, in MySQL, as:-

```
CREATE TABLE DBIDEMODATA1(
        SEQNO INTEGER not null,
        ROW_COUNTER INTEGER not null,
        SUBSYSTEM  INT,
        PEDESTAL   FLOAT,
        GAIN1      FLOAT,
        GAIN2      FLOAT,
        primary key(SEQNO,ROW_COUNTER));
```

as you can see there is a simple 1:1 correspondence between them except that the database table has two additional leading entries:-

```
        SEQNO INTEGER not null,
        ROW_COUNTER INTEGER not null,
```

and a trailing entry:-

```
        primary key(SEQNO,ROW_COUNTER));
```

ROW_COUNTER is a column whose value is generated by the interface, it isn't part of table row class. Its sole purpose is to ensure that every row in the table is unique; an import design constraint for any database. This is achieved by ensuring that, for a given SEQNO, each row has a different value of ROW_COUNTER. This allows the combination of these two values to form a primary (unique) key, which is declared in the trailing entry.

All database tables supported by the DatabaseInterface have an auxiliary Context Range Tables that defines validity ranges for them. Each validity range is given a unique sequence number that acts as a key and corresponds to `SeqNo`. In our case, indeed every case apart from the table name, the definition is:-

```
create table  DbiDemoData1Vld(
         SEQNO integer not null primary key,
         TIMESTART datetime not null,
         TIMEEND datetime not null,
         SITEMASK tinyint(4),
         SIMMASK tinyint(4),
         TASK integer,
         AGGREGATENO integer,
         VERSIONDATE datetime not null,
         INSERTDATE datetime not null,
         key TIMESTART (TIMESTART),
         key TIMEEND (TIMEEND));
```

When the DatabaseInterface looks for data that is acceptable for a give validity it:-

1. Matches the validity to an entry in the appropriate Context Range Table and gets its SeqNo.
2. Uses SeqNo as a key into the main table to get all the rows that match that key.

So, as a designer, you need to be aware of the sequence number, and the row counter *must* be the first two columns in the database table, but are not reflected in the table row class.

Filling a table row object from the database is done using the class's Fill method. For our example:-

```
void DbiDemoData1::Fill(DbiResultSet& rs,
                        const DbiValidityRec* vrec) {

  rs >> fSubSystem >> fPedestal >> fGain1 >> fGain2;

}
```

the table row object is passed a `DbiResultSet` which acts rather like an input stream. The sequence number has already been stripped off; the class just has to fill its own data member. The DatabaseInterface does type checking (see the next section) but does not fail if there is a conflict; it just produces a warning message and puts default data into the variable to be filled.

The second argument is a `DbiValidityRec` which can, if required, be interrogated to find out the validity of the row. For example:-

```
  const ContextRange& range = vrec->GetContextRange();
```

vrec may be zero, but only when filling `DbiValidityRec` objects themselves. On all other occasions vrec should be set.

Creating a Database Table

The previous section gave a simple MySQL example of how a database table is defined. There is a bit more about MySql in section 17.8. The table name normally must match the name of the table row class that it corresponds to. There is a strict mapping between database column types and table row data members, although in a few cases one column type can be used to load more than one type of table row member. The table 17.1 gives the recommended mapping between table row, and MySQL column type.

Notes

Table Row Type	MySQL Type	Comments
Bool_t	CHAR	
Char_t	CHAR	
Char_t*	CHAR(n) n<4	n <4
Char_t*	TEXT	n >3
string	TEXT	
Short_t	TINYINT	8 bit capacity
Short_t	SMALLINT	16 bit capacity
Int_t	TINYINT	8 bit capacity
Int_t	SMALLINT	16 bit capacity
Int_t	INT or INTEGER	32 bit capacity
Float_t	FLOAT	
Double_t	DOUBLE	
TimeStamp	DATETIME	

Table 17.1: Recommended table row and database column type mappings

1. To save table space, select CHAR(n) for characters strings with 3 or less characters and select the smallest capacity for integers.
2. The long (64 bit) integer forms are not supported as on (some?) Intel processors they are only 4 bytes long.
3. Although MySQL supports unsigned values we banned them when attempting to get a previous interface to work with ORACLE, so unsigned in database column type should be avoided. It is allowed to have unsigned in the table row when a signed value is not appropriate and the interface will correctly handle I/O to the signed value in the database even if the most significant bit is set i.e. the signed value in the database is negative. It is unfortunate that the signed value in the database will look odd in such cases.

Designing a Table Row Class

Here is a list of the requirements for a table row class.

Must inherit from DbiTableRow All table row objects must publicly inherit from the abstract class`DbiTableRow`. `DbiTableRow` does provide some default methods even though it is abstract.

Must provide a public default constructor e.g.:-

```
DbiDemoData1::DbiDemoData1() { }
```

The DatabaseInterface needs to keep a object of every type of table row class.

Must implement CreateTableRow method e.g.:-

```
virtual DbiTableRow* CreateTableRow() const {
                            return new DbiDemoData1; }
```

The DatabaseInterface uses this method to populate results tables.

May overload the GetIndex method As explained in section 17.4.2 the ordering of rows in a table is determined by the way data is written to the database. Where that does not form a natural way to access it, table row objects can declare their own index using:-

```
UInt_t GetIndex(UInt_t defIndex) const
```

`DbiDemoData2` provides a rather artificial example:-

```
   UInt_t GetIndex(UInt_t defIndex) const { return fSubSystem/10; }
```

and is just meant to demonstrate how a unique index could be extracted from some packed identification word.

The following is required of an index:-

- The number must be unique within the set.
- It must fit within 4 bytes.

GetIndex returns an unsigned integer as the sign bit has no special significance, but its O.K. to derive the index from a signed value, for example:-

```
Int_t PlexStripEndId::GetEncoded() const
```

would be a suitable index for tables indexed by strip end.

Must implement Fill method This is the way table row objects get filled from a `DbiResultSet` that acts like an input stream. We have seen a simple example in `DbiDemoData1`:-

```
void DbiDemoData1::Fill(DbiResultSet& rs,
                        const DbiValidityRec* vrec) {

  rs >> fSubSystem >> fPedestal >> fGain1 >> fGain2;

}
```

However, filling can be more sophisticated. `DbiResultSet` provides the following services:-

```
     string DbiResultSet::CurColName() const;
     UInt_t DbiResultSet::CurColNum() const;
     UInt_t DbiResultSet::NumCols() const;
DbiFieldType DbiResultSet::CurColFieldType() const;
```

The first 3 give you the name of the current column, its number (numbering starts at one), and the total number of columns in the row. `DbiFieldType` can give you information about the type, concept and size of the data in this column. In particular you can see if two are compatible i.e. of the same type:-

```
Bool_t DbiFieldType::IsCompatible(DbiFieldType& other) const;
```

and if they are of the same capacity i.e. size:-

```
Bool_t DbiFieldType::IsSmaller(DbiFieldType& other) const;
```

You can create `DbiFieldType` objects e.g:-

```
DbiFieldType myFldType(Dbi::kInt)
```

see enum Dbi::DataTypes for a full list, to compare with the one obtained from the current row.

In this way filling can be controlled by the names, numbers and types of the columns. The Fill method of `DbiDemoData1` contains both a "dumb" (take the data as it comes) and a "smart" (look at the column name) code. Here is the latter:-

```
Int_t numCol = rs.NumCols();

//  The first column (SeqNo) has already been processed.
for (Int_t curCol = 2; curCol <= numCol; ++curCol) {
  string colName = rs.CurColName();
  if (      colName == "SubSystem" ) rs >> fSubSystem;
  else if ( colName == "Pedestal" )  rs >> fPedestal;
  else if ( colName == "Gain1" )     rs >> fGain1;
  else if ( colName == "Gain2" )     rs >> fGain2;
  else {
    LOGDEBUG1(dbi) << "Ignoring column " << curCol
                   << "(" << colName << ")"
                   << "; not part of DbiDemoData1" << endl;
    rs.IncrementCurCol();
  }
}
```

Being "smart" comes at a price; if your table has many rows valid at at time, defensive programming like this can cost performance!

In such cases, and if the table only exists is a few variants, its better to determine the variant and then branch to code that hardwires that form

Other services that `DbiResultSet` offers are:-

```
UInt_t DbiResultSet::CurRowNum() const;
Bool_t DbiResultSet::IsExhausted() const;
string DbiResultSet::TableName();
```

These tell you the current row number, whether there is no data left and the name of the table.

Also note that it is not a rule that database columns and class data members have to be in a 1:1 correspondence. So long as the table row can satisfy its clients (see below) it can store information derived from the database table rather than the data itself.

Must impliment the Store method Similar to the Fill method, a row must know how to store itself in the database. Again, this is usually simple; you simply stream out the row elements to the stream provided:

```
void DbiDemoData1::Store(  (DbiOutRowStream& ors,
                           const DbiValidityRec* /* vrec */) const {

  ors << fSubSystem << fPedestal << fGain1 << fGain2;

}
```

must impliment the GetDatabaseLayout method This method is used by a user wanting to do first-time creation of the databases from within the code. Doing this simplifies the table creation process slightly: simply list the columns that this class requires.

```
std::string  DbiDemoData1::GetDatabaseLayout()
{
  std::string table_format =
    "SUBSYSTEM  int,      "
    "PEDESTAL    float,   "
    "GAIN1       float,   "
    "GAIN2       float    ";
  return table_format;
}
```

May overload the CanL2Cache method As explained in section 17.2 the Level 2 cache allows table loading to be speeded up by caching the query results as disk files. Only certain tables support this option which by default is disabled. To enable it the table row object overrides this method as follows:-

```
Bool_t CanL2Cache() const { return kTRUE; }
```

Only table row classes who data members are built-in data types (ints, floats and chars) should do this. Table rows having objects or dynamic data e.g. string or pointers must not claim to support L2 caching. Note the table row doesn't need code to save/restore to the cache, this is handled by the `DbiTableProxy`

Must Provide Services to its Clients There would not be much point in its existence otherwise would there? However its not necessarily the case that all its does is to provide direct access to all the data that came from the table. This subject is explored in the next section.

The Dictionary files

FIXME: Need to include instructions for properly doing dict.h and dict.xml files describing table rows, DbiResultPtr and DbiWriter, if I ever figure out how.

Data Encapsulation

A table row object is the gateway between a database table and the end users who want to use the data it contains. Like any good OO design, the aim should be to hide implementation and only expose the abstraction. There is nothing wrong in effectively giving a 1:1 mapping between the columns of the database table and the getters in the table row object if that is appropriate. For example, a table that gives the position of each PMT in a detector is going to have an X, Y and Z both in the database and in the getter. However at the other extreme there is calibration. Its going to be well into detector operation before the best form of calibration has been found, but it would be bad design to constantly change the table row getters. Its far better to keep the data in the database table very generic, for example:-

```
SeqNo      int,
SubSystem  int,
CalibForm  int,
parm0      float,
parm1      float,
parm2      float,
...
```

The significance of parm0,... depends on CalibForm. The table row object could then provide a calibration service:-

```
Float_t Calibrate(Float_t rawValue) const;
```

rather than expose parm0,.. Calibrate() would have code that tests the value of CalibForm and then uses the appropriate formula involving parm0... Of course some validation code will want to look at the quality of the calibration by looking at the calibration constants themselves, but this too could be abstracted into a set of values that hide the details of the form of the calibration.

However, it is strongly advised to make the raw table values available to the user.

17.6 Filling Tables

17.6.1 Overview

DatabaseInterface can be used to write back into any table from which it can read. To do this you need the services of a `DbiWriter` which is a templated class like `DbiResultPtr`. For example, to write `DbiDemoData1` rows you need an object of the class:-

```
DbiWriter<DbiDemoData1>
```

`DbiWriter` only fills tables, it does not create them

Always create new tables with mysql before attempting to fill them

If you want to create the tables within the same job as the one that fills it then you can do so as follows:-

```
// Create a single instance of the database row, and use
// it to prime the database.  This needs only be done once.
// It will do nothing if the tables already exist.
MyRowClass dummy; // Inherits from DbiTableRow.
int db = 0;       // DB number. If 0, this data is put into the first
                  // database in the cascade;
                  // i.e. the first database in the ENV_TSQL_URL
dummy.CreateDatabaseTables(db);
```

In outline the filling procedure is as follows:-

1. Decide the validity range of the data to be written and store it in a `ContextRange` object.
2. Instantiate a `DbiWriter` object using this `ContextRange` object together with an aggregate number and task. Aggregate numbers are discussed below.
3. Pass filled `DbiTableRow` sub-class objects (e.g. `DbiDemoData1`) to the `DbiWriter`. It in turn will send these objects their Store message that performs the inverse of the Fill message. DbiWriter caches the data but performs no database I/O at this stage.
4. Finally send the `DbiWriter` its Close message which triggers the output of the data to the database.

The fact that I/O does not occur until all data has been collected has a couple of consequences:-

- It minimises the chances of writing bad data. If you discover a problem with the data while `DbiWriter` is assembling it you use `DbiWriter`'s Abort method to cancel the I/O. Likewise if `DbiWriter` detects an error it will not perform output when Close is invoked. Destroying a `DbiWriter` before using Close also aborts the output.
- Although `DbiWriter` starts life as very lightweight, it grows as the table rows are cached.

  Be very sure that you delete the `DbiWriter` once you have finished with it or you will have a serious memory leak!

  To cut down the risk of a memory leak, you cannot copy construct or assign to `DbiWriter` objects.

17.6.2 Aggregate Numbers

As explained in Concepts (see section 17.2) some types of data are written for the entire detector as a single logical block. For example the way PMT pixels map to electronics channels might be written this way. On the other hand if it is written in smaller, sub-detector, chunks then it is Aggregated. For example light injection constants come from pulser data and it is quite possible that a calibration run will only pulse some LEDs and so only part of a full detector set of constants gets written to the database for the run. Each chunk is called an aggregate and given an aggregate number which defines the sub-section of the detector it represents. For pulser data, the aggregate number will probably be the logical (positional) LED number A single `DbiWriter` can only write a single aggregate at a time, for every aggregate can in principle have a different validity range. For unaggregated data, the aggregate number is -1, for aggregated data numbers start at 0,1,2...

The way that the DatabaseInterface assembles all valid data for a given context is as follows:-

- First if finds all aggregate records that are currently valid.
- For each aggregate number it finds the best (most recently created) record and loads all data associated with it.

This has two consequences:-

- For a given table, the regime whereby the data is organised into aggregates should remain constant throughout all records in the table. If absolutely necessary the regime can be changed, but no records must have validities that span the boundary between one regime and another. Were that to be the case the same entry could appear in two valid records with different aggregates numbers and end up appearing in the table multiple times. The system checks to see that this does not happen by asking each row to confirm it's aggregate number on input.
- For any given context it is not necessary for all detector elements to be present; just the ones that are really in the detector at that time. For example, the Far detector will grow steadily over more than a year and this will be reflected in some database tables with the number of valid aggregates similarly growing with time. What aggregates are present can appear in any order in the database tables, the interface will assemble them into the proper order as it loads them.

Its perfectly possible that a calibration procedure might produce database data for multiple aggregates at a single pass. If you are faced with this situation and want to write all aggregates in parallel, then simply have a vector of `DbiWriter`'s indexed by aggregate number and pass rows to the appropriate one. See `DbiValidate::Test_6()` for an example of this type of parallel processing.

17.6.3 Simple Example

We will use the class `DbiDemoData1` to illustrate each of the above steps.

1. Set up `ContextRange` object.
   Typically the `ContextRange` will be based on the `Context` for the event data that was used to generate the database data that is to be stored. For our example we will assume that `DbiDemoData1` represents calibration data derived from event data. It will be valid for 1 week from the date of the current event and be suitable for the same type of data.

```
Context now;  // Event context e.g. CandHandle::GetContext()
TimeStamp start = now.GetTimeStamp();
// Add 7 days (in secs) to get end date.
time_t vcSec = start.GetSec() + 7*24*60*60;
```

```
TimeStamp    end(vcSec,0);
// Construct the ContextRange.
ContextRange      range(now.GetDetector(),
                    now.GetSimFlag(),
                    start,
                    end,
                    "Demo");
```

2. Instantiate a `DbiWriter`.
   Create a `DbiDemoData1` writer for unaggregated data task 0.

```
Int_t aggNo = -1;
Dbi::SubSite subsite = 0;
Dbi::Task task = 0;
// Decide a creation date (default value is now)
TimeStamp create;
DbiWriter<DbiDemoData1> writer(range,aggNo,subsite,task,create);
```

3. Pass filled `DbiDemoData1` objects.

```
// Create some silly data.
DbiDemoData1 row0(0,10.,20.,30.);
DbiDemoData1 row1(0,11.,21.,31.);
DbiDemoData1 row2(0,12.,22.,32.);

// Store the silly data.
writer << row0;
writer << row1;
writer << row2;
```

   The `DbiWriter` will call `DbiDemoData1`'s Store method.

   Again notice that the SeqNo, which is part of the table row, but not part of the class data, is silently handled by the system.

4. Send the `DbiWriter` its Close message.

```
writer.Close();
```

17.6.4 Using DbiWriter

- The `DbiWriter`'s constructor is:-

```
DbiWriter(const ContextRange& vr,
          Int_t aggNo,
          Dbi::SubSite subsite= 0,
          Dbi::Task task = 0,
          TimeStamp versiondate = TimeStamp(0,0),
          UInt_t dbNo = 0,
          const std::string& LogComment = "",
          const std::string& tableName = ""
          );
```

  – The first argument determines the validity range of the data to be written, i.e. what set of `Contexts` it is suitable for. You can control the date range as well as the type(s) of data and detector.

- The second argument is the aggregate number. For unaggregated data it is -1, for aggregated data its a number in the range 0..n-1 where n is the number of aggregates.
- The third argument is the SubSite of the data. It has a default of 0.
- The third argument is the Task of the data. It has a default of 0.
- The fourth argument supplies the data's version date. The default is a special date and time which signifies that `DbiWriter` is to use Overlay Version Dates (see Concepts section 17.2.9.) Alternatively, at any time before writing data, use the method:-

  ```
  void SetOverlayVersionDate();
  ```

  to ensure that `DbiWriter` uses Overlay Version Dates.
- The fifth argument defines which entry in the database cascade the data is destined for. By default it is entry 0 i.e. the highest priority one.

  Caution: Supplying the entry number assumes that at execution time the cascade is defined in a way that is consistent with the code that is using the `DbiWriter`. As an alternative, you can supply the database name (e.g. offline) if you know it and are certain it will appear in the cascade.
- The sixth argument supplies a comment for the update. Alternatively, at any time before writing data, use the method:-

  ```
  void  SetLogComment(const std::string& LogComment)
  ```

  Update comments are ignored unless writing to a Master database (i.e. one used as a source database e,g. the database at FNAL), and in this case a non-blank comment is mandatory unless the table is exempt. Currently only DBI, DCS and PULSER tables are exempt.

  If the first character on the string is the '@' character then the rest of the string will be treated as the name of a file that contains the comment. If using DbiWriter to write multiple records to the same table as part of a single update then only create a single DbiWriter and use the Open method to initialise for the second and subsequent records. That way a single database log entry will be written to cover all updates.
- The last argument supplies the name of the table to be written to. Leaving it blank will mean that the default table will be used i.e. the one whose name matches, apart from case, the name of object being stored. Only use this feature if the same object can be used to fill more than one table.

- Having instantiated a `DbiWriter`, filled table row objects must be passed using the operator:-

  ```
  DbiWriter<T>& operator<<(const T& row);
  ```

  for example:-

  ```
  writer << row0;
  writer << row1;
  writer << row2;
  ```

  `DbiWriter` calls the table row's Store method, see the next section. It also performs some basic sanity checks:-

  - The row's aggregate number matches its own.
  - The type of the data written is compatible with database table.

  If either check fails then an error message is output and the data marked as bad and the subsequent Close method will not produce any output.

- Once all rows for the current aggregate have been passed to `DbiWriter` the data can be output using:-

```
Bool_t Close();
```

  which returns true if the data is successfully output.

  Alternatively, you can write out the data as a DBMauto update file by passing the name of the file to the Close command:-

```
Close("my_dbmauto_update_file.dbm");
```

- On output a new sequence number is chosen automatically. By default, if writing permanent data to an authorising database or if writing to a file, a global sequence number will be allocated. In all other cases a local sequence number will be be used. For database I/O, as opposed to file I/O, you can change this behaviour with

```
void SetRequireGlobalSeqno(Int_t requireGlobal)

Where requireGlobal
  > 0  Must be global
  = 0  Must be global if writing permanent data to an authorising database
  < 0  Must be local
```

- At any time before issuing the Close command you can cancel the I/O by either:-
  - Destroying the `DbiWriter`.
  - Using the method:-

```
void Abort();
```

- If you want to, you can reuse a `DbiWriter` by using:-

```
Bool_t Open(const ContextRange& vr,
            Int_t aggNo,
            Dbi::Task task = 0,
            TimeStamp versionDate = TimeStamp(),
            UInt_t dbNo = 0);
```

  The arguments have the same meaning as for the constructor. An alternative form of the Open statement allows the database name to be supplied instead of its number. If the `DbiWriter` is already assembling data then the Close method is called internally to complete the I/O. The method returns true if successful. As explained above, the Open method must be used if writing multiple records to the same table as part of a single update for then a single database log entry will be written to cover all updates.

17.6.5 Table Row Responsibilities

All `DbiTableRow` sub-class objects must support the input interface accessed through `DbiResultPtr`. The responsibilities that this implies are itemised in section 17.5.2. The output interface is optional; the responsibilities listed here apply only if you want to write data to the database using this interface.

Must override GetAggregateNo method if aggregated `DbiTableRow` supplies a default that returns -1. The GetAggregateNo method is used to check that table row objects passed to a particular `DbiWriter` have the right aggregate number.

Must override Store Method The Store method is the inverse to Fill although it is passed a `DbiOutRowStream` reference:-

```
void Store(DbiOutRowStream& ors) const;
```

rather than a `DbiResultSet` reference. Both these classes inherit from `DbiRowStream` so the same set of methods:-

```
      string DbiResultSet::CurColName() const;
      UInt_t DbiResultSet::CurColNum() const;
      UInt_t DbiResultSet::NumCols() const;
DbiFieldType DbiResultSet::CurColFieldType() const;
      UInt_t DbiResultSet::CurRowNum() const;
      string DbiResultSet::TableName();
```

are available. So, as with the Fill method, there is scope for Store to be "smart". The quotes are there because it often does not pay to be too clever! Also like the Fill method its passed a `DbiValidityRec` pointer (which is only zero when filling `DbiValidityRec` objects) so that the validity of the row can be accessed if required.

17.6.6 Creating and Writing Temporary Tables

It is possible to create and write temporary tables during execution. Temporary tables have the following properties:-

- For the remainder of the job they look like any other database table, but they are deleted when the job ends.
- They completely obscure all data from any permanent table with the same name in the same database. Contrast this with the cascade, which only obscures data with the same validity.
- They are local to the process that creates them. Even the same user running another job using the same executable will not see these tables.

Temporary tables are a good way to try out new types of table, or different types of data for an existing table, without modifying the database. Writing data is as normal, by means of a `DbiWriter`, however before you write data you must locate a database in the cascade that will accept temporary tables and pass it a description of the table. This is done using the `DbiCascader` method `CreateTemporaryTable`. You can access the cascader by first locating the singleton `DbiTableProxyRegister` which is in overall charge of the DatabaseInterface. The following code fragment shows how you can define a new table for DbiDemoData1:-

```
#include "DatabaseInterface/DbiCascader.h"
#include "DatabaseInterface/DbiTableProxyRegistry.h"

...

//  Ask the singleton DbiTableProxyRegistry for the DbiCascader.
  const DbiCascader& cascader
              = DbiTableProxyRegistry::Instance().GetCascader();

//  Define the table.
  string tableDescr = "(SEQNO INT,   SUBSYSTEM  INT, PEDESTAL FLOAT,"
                      " GAIN1 FLOAT, GAIN2 FLOAT )";
//  Ask the cascader to find a database that will accept it.
  Int_t dbNoTemp = cascader.CreateTemporaryTable("DbiDemoData1",
                                                 tableDescr);
```

```
if ( dbNoTemp < 0 ) {
   cout << "No  database to will accept temporary tables. " << endl;
}
```

You pass `CreateTemporaryTable` the name of the table and its description. The description is a parenthesised comma separated list. It follows the syntax of the MYSQL CREATE TABLE command, see section 17.8.

In principle not every database in the cascade will accept temporary tables so the cascader starts with the highest priority one and works done until it finds one, returning its number in the cascade. It returns -1 if it fails. For this to work properly the first entry in the cascade must accept it so that it will be taken in preference to the true database. It is recommended that the first entry be the temp database, for everyone has write-access to that and write-access is needed to create even temporary tables. So a suitable cascade might be:-

```
setenv ENV_TSQL_URL "mysql://pplx2.physics.ox.ac.uk/temp;\
                     mysql://pplx2.physics.ox.ac.uk/offline"
```

Having found a database and defined the new or replacement table, you can now create a `DbiWriter` and start writing data as describe in section 17.6. You have to make sure that the DbiWriter will output to the correct database which you can either do by specifying it using the 5th arg of its constructor:-

```
DbiWriter(const ContextRange& vr,
          Int_t aggNo,
          Dbi::Task task = 0,
          TimeStamp versionDate = TimeStamp(),
          UInt_t dbNo = 0);
```

or alternatively you can set it after construction:-

```
DbiWriter<DbiDemoData1> writer(range,aggNo);
writer.SetDbNo(dbNoTemp);
```

As soon as the table has been defined it will, as explained above, completely replace any permanent table in the same database with the same name. However, if there is already data in the cache for the permanent table then it may satisfy further requests for data. To prevent this from happening you can clear the cache as described in the next section.

> *Do NOT write permanent data to any temporary database for it could end up being used by anyone who includes the database for temporary tables. Database managers may delete any permanent tables in temporary databases without warning in order to prevent such problems.*

17.6.7 Clearing the Cache

Normally you would not want to clear the cache, after all its there to improve performance. However if you have just created a temporary table as described above, and it replaces an existing table, then clearing the cache is necessary to ensure that future requests for data are not satisfied from the now out of date cache. Another reason why you may want to clear the cache is to study database I/O performance.

Although this section is entitled Clearing the Cache, you cannot actually do that as the data in the cache may already be in use and must not be erased until its clients have gone away. Instead the data is marked as stale, which is to say that it will ignored for all future requests. Further, you don't clear the entire cache, just the cache associated with the table that you want to refresh. Each table is managed by a `DbiTableProxy` that owns a `DbiCache`. Both `DbiWriter` and `DbiResultPtr` have a `TableProxy` method to access the associated `DbiTableProxy`. The following code fragment shows how to set up a writer and mark its associated cache as stale:-

```
DbiWriter<DbiDemoData1> writer(range,aggNo);
writer.SetDbNo(dbNoTemp);
writer.TableProxy().GetCache()->SetStale();
```

17.7 ASCII Flat Files and Catalogues

17.7.1 Overview

ASCII flat files and catalogues provide a convenient way to temporarily augment a database with additional tables under your control. A flat file is a file that contains, in human readable form, the definition of a table and its data. It can be made an entry in a cascade and, by placing before other entries allows you to effectively modify the database just for the duration of a single job. As has already been explained, for each Main Data Table there is also an auxiliary Context Range Table, so you need 2 entries in the cascade for each table you want to introduce. The problem with this scheme is that, if introducing a number of tables, the cascade could get rather large. To avoid this catalogues are used. A catalogue is actually nothing more that a special ASCII flat file, but each row of its data is a URLs for another ASCII flat file that becomes part of the same cascade entry. In this way a single cascade entry can consist of an arbitrary number of files.

17.7.2 Flat Files

An ASCII flat file defines a single database table.

Format

The format is sometimes referred to as Comma Separated Value (CSV). Each line in the file corresponds to a row in the table. As you might suspect, values are separated by commas, although you can add additional white space (tabs and spaces) to improve readability (but heed the caution in section 17.7.4). The first row is special, it contains the column names and types. The types must valid MySQL types, see table 17.1 for some examples. If the special row is omitted or is invalid then the column names are set to C1, C2, ... etc. and all types are set to string (TEXT). Here is a simple example of a CSV file:-

```
SeqNo int, Pedestal float, SubSystem int, Gain1 float, Gain2 float
1,    1.0,    0,    10.,    100.
1,    1.1,    1,    11.,    110.
1,    1.2,    2,    12.,    120.
1,    1.3,    3,    13.,    130.
```

Its in a convention to use the file extension .csv, but it is not compulsory.
If any value is a string or a date, it *must* be delimited by double quotes.

URL

The database URL is based on the standard one extended by adding the suffix

```
#absolute-path-to-file
```

For example:-

```
mysql://coop.phy.bnl.gov/temp#/path/to/MyTable.csv
```

The table name is derived from the file name after stripping off the extension. In this example, the table name will be MyTable

17.7.3 Catalogues

These are special types of ASCII Flat File. Their data are URLs to other flat files. You cannot nest them i.e. one catalogue cannot contain a URL that is itself catalogue.

Format

The first line of the file just contains the column name "name". The remaining lines are URLs of the flat files. Here is a simple example:-

```
name
file:/home/dyb/work/MyData.csv
file:/home/dyb/work/MyDataVld.csv
file:$MY_ENV/MyDataToo.csv
file:$MY_ENV/MyDataTooVld.csv
```

This catalogue defines two tables MyData and MyDataToo each with its associated auxiliary validity range table. Note that files names must be absolute but can begin with an environmental variable.

URL

The URL is identical to any other flat file with one additional constraint: the extension must be .cat or .db. For example:

```
mysql://coop.phy.bnl.gov/dyb_offline#/home/dyb/work/MyCatalogue.db
```

17.7.4 Example

The stand-alone testing of the Database Interface includes an example of an ASCII Catalogue. The URL of the cascade entry is:-

```
mysql://coop.phy.bnl.gov/dyb_test#\$DATABASEINTERFACE_ROOT/DbiTest/scriptsDemoASCIICatalogue.db
```

If you look at the file:-

```
\$DATABASEINTERFACE_ROOT/DbiTest/scripts/DemoASCIICatalogue.db
```

you will see it contains 4 lines, defining the tables `DEMOASCIIDATA` (a Detector Descriptions table) and `DEMOASCIICONFIG` (Algorithm Configurations table):-

```
file:$DBITESTROOT/scripts/DEMOASCIIDATA.csv
file:$DBITESTROOT/scripts/DEMOASCIIDATAVld.csv
file:$DBITESTROOT/scripts/DEMOASCIICONFIG.csv
file:$DBITESTROOT/scripts/DEMOASCIICONFIGVld.csv
```

In both cases, the auxiliary validity range table defines a single validity range, although there is no reason why it could not have defined any number. For the `DEMOASCIIDATA`, there are 5 rows, a header row followed by 4 rows of data:-

```
SEQNO INT, UNWANTED INT, PEDESTAL FLOAT, SUBSYSTEM INT, GAIN1 FLOAT, GAIN2 FLOAT
1,99,1.0,0,10.,100.
1,99,1.1,1,11.,110.
1,99,1.2,2,12.,120.
1,99,1.3,3,13.,130.
```

For the `DEMOASCIICONFIG` table, there are only two rows:-

```
SEQNO INT, CONFIGSTRING TEXT
1,"mybool=1 mydouble=1.23456789012345678e+200 mystring='This is a string' myint=12345"
```

Caution: Note, don't have any white space between the comma and the leading double quote of the configuration string.

17.8 MySQL Crib

This provides the absolute bare minimum to install, manage and use a MySQL database in the context of the DatabaseInterface.

17.8.1 Introduction

The following are useful URLs:-

- MySQL home page:-

  http://www.mysql.com/

- from which you can reach a documentation page:-

  http://www.mysql.com/documentation/index.html

- and the downloads for 3.23:-

  http://www.mysql.com/downloads/mysql-3.23.html

A good book on MySQL is:-

MySQL by Paul DuBois, Michael Widenius. New Riders Publishing; ISBN: 0-7357-0921-1

17.8.2 Installing

See:-

https://wiki.bnl.gov/dayabay/index.php?title=Database
https://wiki.bnl.gov/dayabay/index.php?title=MySQL_Installation

17.8.3 Running mysql

mysql is a utility, used both by system administrators and users to interact with MySQL database. The command syntax is:-

```
mysql [-h host_name] [-u user_name] [-pyour_pass]
```

if you are running on the server machine, with you Unix login name and no password then:-

```
mysql
```

is sufficient. To exit type:-

```
\q
```

Note: most mysql commands are terminated with a semi-colon. If nothing happens when you type a command, the chances are that mysql is still waiting for it, so type it and press return again.

17.8.4 System Administration

This also has to be done as root. As system administrator, MySQL allows you to control access, on a user by user basis, to databases. Here are some example commands:-

```
create database dyb_offline;
grant all on    dyb_offline.*       to  smart@coop.bnl.phy.gov
grant all on    dyb_offline.*       to  smart@"%"
grant select    dyb_offline.Boring  to  dumb@coop.bnl.phy.gov
\q
```

- The first lines creates a new database called dyb_offline. With MySQL you can have multiple databases.
- The next two lines grants user smart, either logged in locally to the server, or remotely from anywhere on the network all privileges to all tables in that database.
- The next line grants user dumb, who has to be logged in locally, select (i.e. read) access to the table Boring in the same database.

17.8.5 Selecting a Database

Before you can use mysql to create, fill or examine a database table you have to tell it what database to use. For example:-

```
use dyb_offline
```

'use' is one of the few commands that does not have a trailing semi-colon.

17.8.6 Creating Tables

The following commands create, or recreate, a table and display a description of it:-

```
drop table if exists DbiDemoData1;
create table DbiDemoData1(
    SeqNo      int,
    SubSystem  int,
    Pedestal   float,
    Gain1      float,
    Gain2      float
 );
describe DbiDemoData1;
```

See table 17.1 for a list of MySQL types that the DatabaseInterface currently supports.

17.8.7 Filling Tables

The following commands add data from the file DemoData1.dat to an existing table:-

```
load data local infile 'DemoData1.dat' into table DbiDemoData1;
```

Each line of the file corresponds to a row in the table. Columns should be separated with tabs. Table 17.2 shows typical formats of the various data types.

MySQL Type	Table Row Type
CHAR	a
TINYINT	-128
SMALLINT	-32768
INT or INTEGER	-2147483647
FLOAT	-1.234567e-20
DOUBLE	1.23456789012345e+200
TEXT	'This is a string'
DATETIME	'2001-12-31 04:05:06'

Table 17.2: Example data formats.

17.8.8 Making Queries

Here is a sample query:-

```
select * from DbiDemoData2Validity where
    TimeStart <= '2001-01-11 12:00:00'
and TimeEnd    > '2000-12-22 12:00:00'
and SiteMask & 4
order by TimeStart desc
;
```

17.9 Performance

17.9.1 Holding Open Connections

Connections to the database are either permanent i.e. open all the time or temporary i.e. they are closed as soon as a I/O operation is complete. A connection is made permanent if:-

- Connecting to a ASCII flat file database as re-opening such a database would involve re-loading all the data.
- Temporary data is written to the database for such data would be lost if the connection were closed.

In all other cases the connection is temporary so as to minimise resources (and in the case ORACLE resources that have to be paid for!). For normal operations this adds little overhead as typically there are several major database reads at the start of a production job after which little or no further database I/O occurs. However if you require the connection to remain open throughout the job then you can force any entry in the cascade to be permanent. The following code sets entry 0 in the cascade to have a permanent connection:-

```
#include "DatabaseInterface/DbiCascader.h"
#include "DatabaseInterface/DbiTableProxyRegistry.h"


//  Ask the singleton DbiTableProxyRegistry for the DbiCascader.
  const DbiCascader& cascader
               = DbiTableProxyRegistry::Instance().GetCascader();
// Request that entry 0 is permanently open.
  cascader.SetPermanent(0);
```

Note that this won't open the connection but will prevent it from closing after its next use.

If you want all connections to remain open this can be set through the configuration parameter MakeConnectionsPermanent. See section 17.3.2.

17.9.2 Truncated Validity Ranges

Standard context specific queries are first trimmed to a time window to limit the number of Vld records that have to be analysed. Having established the best data, a further 4 calls to query the Vld table is made to determine the full validity. For data with long validities, these extra calls are worthwhile as they can significantly increase the lifetime of the results. However there are two cases where these should not be use:-

- For data that changes at high frequency (minutes or hours rather than days) it may waste time doing the extra searches, although the results would be valid.
- For sparse aggregation - see 17.2.2. The algorithm opens up the window on the basis of the aggregates present at the supplied context so won't take account of aggregates not present and might over-estimate the time window.

The following `DbiResultPtr` methods support this request:-

```
DbiResultPtr(const Context& vc,
             Dbi::Task task = Dbi::kDefaultTask,
             Dbi::AbortTest abortTest = Dbi::kTableMissing,
             Bool_t findFullTimeWindow = true);

DbiResultPtr(const string& tableName,
             const Context& vc = Dbi::fgDefaultContext,
             Dbi::Task task = Dbi::kDefaultTask,
             Dbi::AbortTest abortTest = Dbi::kTableMissing,
             Bool_t findFullTimeWindow = true);
UInt_t NewQuery(Context vc,
             Dbi::Task task=0,
             Bool_t findFullTimeWindow = true);
```

It is selected by passing in the value `false` for `findFullTimeWindow`.

17.9.3 Timing

DbiTimerManager is a static object that provides performance printout when enabled. By default it is enabled but can be disabled by:-

```
DbiTimerManager::gTimerManager.Enable(false);
```

Chapter 18

Database Maintanence

18.1 Introduction

The DatabaseMaintenance package produces a single binary application: **dbmjob** that provides very basic database maintenance support. Specifically its current function is only as a tool to distribute data between databases.

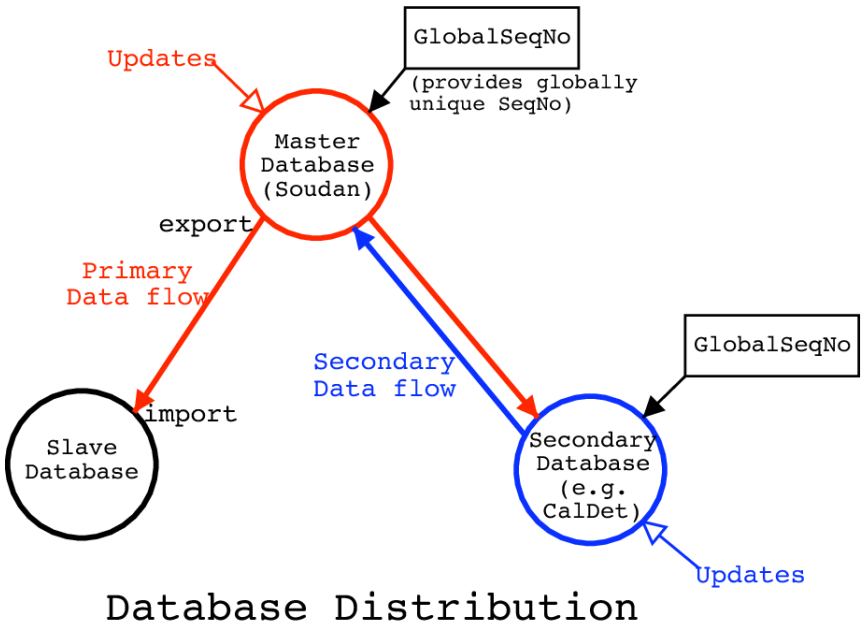

Figure 18.1:

The flow of data is shown schematically in diagram 18.1. At the heart of the system is the Master Database at Soudan. Most database updates enter the database realm here. At regular intervals dbmjob is used to export all recently updated data and these export files are distributed to all other databases where the data is imported if not already present. This is done by the local database manager again using dbmjob. These primary data flows are shown in red.

Smaller amounts of data come from secondary databases e.g. at CalDet and these are exported up to the Master Database where they join other updates for distribution.

This system relies on the ability to:-

- Record the insertion date so that updates can be incremental.
- Uniquely identify data so that it is not accidentally duplicated if attempting import more than once. For example updates to a secondary database might be reflected back if exporting all recent changes. However such data is ignored as duplicated data when resubmitted to the Master.

dbmjob exploits the fact that all Dbi compliant database tables come in pairs, the main data table and an auxiliary validity range table. The auxiliary table records insertion dates and have globally unique SeqNos (Sequence Numbers). The diagram shows how globally unique numbers are assigned. Every database that is a source of data has a `GlobalSeqNo` table that is used to generate sequence numbers. Each time one is allocated the count is incremented in the table. For each database the table operates in a different range of numbers hence ensuring that all are unique. dbmjob moves data in "Validity Packets" i.e. a single row in the auxiliary table and all its associated data rows. The insertion date and SeqNo on the auxiliary row allow dbmjob to support incremental updates and avoid data duplication.

All this implies a very important restriction on dbmjob:-

> *dbmjob can only distribute Dbi compliant database tables i.e. ones that come in pairs, the main data table and an auxiliary validity range table.*

18.2 Building and Running dbmjob

18.2.1 Building

The DatabaseMaintenance package is a standard Framework package and the dbmjob application is build in the standard way:-

```
cd $SRT_PUBLIC_CONTEXT    %$
gmake DatabaseMaintenance.all
```

18.2.2 Running

Before running, a Database cascade must be defined using the `ENV_TSQL_*` variables as described in **??**. Alternatively use the -d, -u and -p switches that are also described there or use the `ENV_TSQL_UPDATE_*` (e.g. `ENV_TSQL_UPDATE_USER`) set of variables. Where they exist, they will take precedence over the equivalent `ENV_TSQL_*` variable. This allows for a safe read-only setting of the `ENV_TSQL_*` variables that can be shared by a group, with just the local database manager also having the `ENV_TSQL_UPDATE_*` set for write-access. Note that the job switches take priority over everything else.

To run, just type:-

```
dbmjob
```

dbmjob enters interactive mode. For help type `Help` and to quit type `Quit`. The following illustrate simple exporting and importing. For more detail consult the Help command.

Exporting Data

dbmjob always exports data from the first database in the cascade.

To export data use the `Export` command. The syntax is:-

```
Export {--Since <date>}  <table>  <file>
```

This exports the contents of <table> into <file> which can subsequently be imported into another database using the Import command. <table> can be a specific table e.g. PlexPixelSpotToStripEnd or * for all tables. For example:-

```
Export * full_backup.dat
Export -since "2001-09-27 12:00:00" PlexPixelSpotToStripEnd update.dat
```

The first updates the entire database whilst the second just records updates to PlexPixelSpotToStripEnd since midday on the 27 September 2001.

Importing Data

By default dbmjob always imports into the first database in the cascade but this can be overridden.

To Import data use the Import command. The syntax is:-

```
Import {--Test } {--DatabaseNumber <no>} <file>
```

This imports the contents <file> into the database. The insertion dates in the file's validity records are replaced by the current date and time so that the insertion dates in the database reflect the local insertion date. Any SeqNo already present will be skipped but the associated data is compared to the corresponding entries in the database to confirm that they are identical, neglecting differences in insertion dates. For example:-

```
Import full_backup.dat
Export --DatabaseNumber 1  update.dat
Import --Test full_backup.dat
```

The first updates the first database (Cascade number 0) whilst the second updates the second database in the cascade. The last does not import at all but still does comparisons so is a convenient way to compare a database to an import file.

Chapter 19

Database Tables

19.1 AdMass

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GAdMass.spec[1]

```
Masses of detector liquids as measured from filling data
```

name	dbtype	codetype	description	code2db
AdNo	int(11)	int	AD Number (1-8)	
GdMass	double	double	GdLS mass in AD in kg	
GdUnct	double	double	GdLS mass uncertainty in kg	
GdBlind	tinyint	int	Indicate whether GdLS mass is blind (1) or not (0)	
LsMass	double	double	LS mass in AD in kg	
LsUnct	double	double	LS mass uncertainty in kg	
MoMass	double	double	MO mass in AD in kg	
MoUnct	double	double	Mo mass uncertainty in kg	

Table 19.1: DBI Table specification for class GAdMass which corresponds to table AdMass

[1] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GAdMass.spec

19.2 AdWpHvMap

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GAdWpHvMap.spec[2]

High voltage cable map table:

```
mysql&gt; describe AdWpHvMap;
+-------------+------------+------+-----+---------+-------+
| Field       | Type       | Null | Key | Default | Extra |
+-------------+------------+------+-----+---------+-------+
| SEQNO       | int(11)    | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11)    | NO   | PRI | NULL    |       |
| LocationId  | char(6)    | YES  |     | NULL    |       |
| HvChannelId | int(11)    | YES  |     | NULL    |       |
| DcBoard     | tinyint(1) | YES  |     | NULL    |       |
| DcChannel   | tinyint(2) | YES  |     | NULL    |       |
+-------------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

name	dbtype	codetype	description	code2db
LocationId	char(6)	string	PMT Location ID	
HvChannelId	int(11)	DayaBay::HvChannelId	Packed HV Channel ID	.fullPackedData()
DcBoard	tinyint(1)	int	Decoupler Board #	
DcChannel	tinyint(2)	int	Decoupler Channel #	

Table 19.2: DBI Table specification for class GAdWpHvMap which corresponds to table AdWpHvMap

[2] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GAdWpHvMap.spec

19.3 AdWpHvSetting

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GAdWpHvSetting.spec[3]

```
Table of high voltage requested values:

mysql&gt; describe AdWpHvSetting;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| SEQNO       | int(11) | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11) | NO   | PRI | NULL    |       |
| HvChannelId | int(11) | YES  |     | NULL    |       |
| HvSetting   | float   | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

name	dbtype	codetype	description	code2db
HvChannelId	int(11)	DayaBay::HvChannelId	Packed HV Channel ID	.fullPackedData()
HvSetting	float	float	Requested HV Value	

Table 19.3: DBI Table specification for class GAdWpHvSetting which corresponds to table AdWpHvSetting

[3] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GAdWpHvSetting.spec

19.4 AdWpHvToFee

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GAdWpHvToFee.spec[4]

```
Map from HV board/channel to FEE channel ID

mysql&gt; describe AdWpHvToFee;
+--------------+---------+------+-----+---------+-------+
| Field        | Type    | Null | Key | Default | Extra |
+--------------+---------+------+-----+---------+-------+
| SEQNO        | int(11) | NO   | PRI | NULL    |       |
| ROW_COUNTER  | int(11) | NO   | PRI | NULL    |       |
| HvChannelId  | int(11) | YES  |     | NULL    |       |
| FeeChannelId | int(11) | YES  |     | NULL    |       |
+--------------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

name	dbtype	codetype	description	code2db
HvChannelId	int(11)	DayaBay::HvChannelId	Packed HV Channel ID	.fullPackedData()
FeeChannelId	int(11)	DayaBay::FeeChannelId	Packed FEE Channel ID	.fullPackedData()

Table 19.4: DBI Table specification for class GAdWpHvToFee which corresponds to table AdWpHvToFee

[4] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GAdWpHvToFee.spec

19.5 CableMap

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCableMap.spec[5]

```
The Cable Map provides the data for lookups between a sensorID and a
channelID.  The ID numbers are packed integers ready for consumption
by the sensor classes from Detectors.h and channel classes from
Electronics.h from the Conventions package.  This table handles
sensors and electronics for PMTs (AD and Pool) and RPCs.

Read context must explicitly give: Site, SimFlag and DetectorId/SubSite
Write context must explicitly give: SiteMask, SimMask and SubSite
```

name	dbtype	codetype	description	code2db
SensorId	int(11)	DayaBay::DetectorSensor	Packed Sensor ID	.fullPackedData()
ChannelId	int(11)	DayaBay::ElecChannelId	Packed Channel ID	.fullPackedData()

Table 19.5: DBI Table specification for class GCableMap which corresponds to table CableMap

[5] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCableMap.spec

19.6 CableMapFix

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCableMapFix.spec[6]

```
Testing fix for :dybsvn:‘ticket:948‘

The Cable Map provides the data for lookups between a sensorID and a
channelID.  The ID numbers are packed integers ready for consumption
by the sensor classes from Detectors.h and channel classes from
Electronics.h from the Conventions package.  This table handles
sensors and electronics for PMTs (AD and Pool) and RPCs.

Read context must explicitly give: Site, SimFlag and DetectorId/SubSite
Write context must explicitly give: SiteMask, SimMask and SubSite
```

name	dbtype	codetype	description	code2db
SensorId	int(11)	DayaBay::DetectorSensor	Packed Sensor ID	.fullPackedData()
ChannelId	int(11)	DayaBay::ElecChannelId	Packed Channel ID	.fullPackedData()

Table 19.6: DBI Table specification for class GCableMapFix which corresponds to table CableMap

[6] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCableMapFix.spec

19.7 CalibFeeGainConv

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibFeeGainConv.spec[7]

docstring

name	dbtype	codetype	description	c
ChannelId	int(11)	DayaBay::FeeChannelId	unique id of this fee channel	.
Status	tinyint(4)	int	status of this gain ratio	
FineCoarseRatio	float	double	gain ratio of fine over coarse adc	
FineCoarseRatioErr	float	double	error of gain ratio	
FRLineSlope	float	double	slope of partial linear fit to fine range	
FRLineErr	float	double	error of fine range slope	
CRLineSlope	float	double	slope of partial linear fit to coarse range	
CRLineErr	float	double	error of coarse range slope	

Table 19.7: DBI Table specification for class GCalibFeeGainConv which corresponds to table CalibFeeGainConv

[7] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibFeeGainConv.spec

19.8 CalibFeeSpec

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibFeeSpec.spec[8]

docstring

name	dbtype	codetype	description
ChannelId	int(10) unsigned	DayaBay::FeeChannelId	Electronics channel ID number
Status	int(10) unsigned	int	Channel status
AdcPedestalHigh	double	double	Measured high-gain Pedestal ADC value
AdcPedestalHighSigma	double	double	high-gain Pedestal ADC sigma
AdcPedestalLow	double	double	Measured low-gain Pedestal ADC value
AdcPedestalLowSigma	double	double	low-gain Pedestal ADC sigma
AdcThresholdHigh	double	double	Channel threshold, as measured in ~ADC
AdcThresholdLow	double	double	Channel threshold, as measured in ~ADC

Table 19.8: DBI Table specification for class GCalibFeeSpec which corresponds to table CalibFeeSpec

[8] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibFeeSpec.spec

19.9 CalibFeeSpecCleanup

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibFeeSpecCleanup.spec[9]

docstring

name	dbtype	codetype	description
ChannelId	int(10) unsigned	DayaBay::FeeChannelId	Electronics channel ID number
Status	int(10) unsigned	int	Channel status
AdcPedestalHigh	double	double	Measured high-gain Pedestal ADC value
AdcPedestalHighSigma	double	double	high-gain Pedestal ADC sigma
AdcPedestalLow	double	double	Measured low-gain Pedestal ADC value
AdcPedestalLowSigma	double	double	low-gain Pedestal ADC sigma
AdcThresholdHigh	double	double	Channel threshold, as measured in ~ADC
AdcThresholdLow	double	double	Channel threshold, as measured in ~ADC

Table 19.9: DBI Table specification for class GCalibFeeSpecCleanup which corresponds to table CalibFeeSpecCleanup

[9] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibFeeSpecCleanup.spec

19.10 CalibPmtFineGain

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtFineGain.spec[10]

docstring

name	dbtype	codetype	description	code2db
ChannelId	int(11)	DayaBay::FeeChannelId	unique id of the channel connected to a pmt	.fullPac
SpeHigh	float	double	SPE mean ADC value (high gain)	
SpeHighError	float	double	error in SPE Mean ADC value (high gain)	
SigmaSpeHigh	float	double	SPE 1-sigma peak width (high gain)	
SpeHighFitQual	float	double	SPE fit quality in chi2/ndf (high gain)	

Table 19.10: DBI Table specification for class GCalibPmtFineGain which corresponds to table CalibPmtFineGain

[10] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtFineGain.spec

19.11 CalibPmtHighGain

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtHighGain.spec[11]

docstring

name	dbtype	codetype	description	code2db
SensorId	int(11)	DayaBay::DetectorSensor	unique id of this pmt	.fullPacke
SpeHigh	float	double	SPE mean ADC value (high gain)	
SpeHighError	float	double	error in SPE Mean ADC value (high gain)	
SigmaSpeHigh	float	double	SPE 1-sigma peak width (high gain)	
SpeHighFitQual	float	double	SPE fit quality in chi2/ndf (high gain)	

Table 19.11: DBI Table specification for class GCalibPmtHighGain which corresponds to table CalibPmtHighGain

[11] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtHighGain.spec

19.12 CalibPmtHighGainFake

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtHighGainFake.spec[12]

```
Expediant table/class used to fake the old table, see :dybsvn:‘ticket:1228
```

name	dbtype	codetype	description	code2db
SensorId	int(11)	DayaBay::DetectorSensor	unique id of this pmt	.fullPacke
SpeHigh	float	double	SPE mean ADC value (high gain)	
SpeHighError	float	double	error in SPE Mean ADC value (high gain)	
SigmaSpeHigh	float	double	SPE 1-sigma peak width (high gain)	
SpeHighFitQual	float	double	SPE fit quality in chi2/ndf (high gain)	

Table 19.12: DBI Table specification for class GCalibPmtHighGainFake which corresponds to table CalibPmtHighGainFake

[12] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtHighGainFake.spec

19.13 CalibPmtHighGainPariah

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtHighGainPariah.spec[13]

```
Handling of input table not conforming to SOP norms,  see :dybsvn:‘ticket:1228
```

name	dbtype	codetype	description	code2db
SensorId	int(11)	DayaBay::DetectorSensor	unique id of this pmt	.fullPacke
SpeHigh	float	double	SPE mean ADC value (high gain)	
SpeHighError	float	double	error in SPE Mean ADC value (high gain)	
SigmaSpeHigh	float	double	SPE 1-sigma peak width (high gain)	
SpeHighFitQual	float	double	SPE fit quality in chi2/ndf (high gain)	

Table 19.13: DBI Table specification for class GCalibPmtHighGainPariah which corresponds to table CalibPmtHighGainPariah

[13] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtHighGainPariah.spec

19.14 CalibPmtLowGain

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtLowGain.spec[14]

docstring

name	dbtype	codetype	description	code2db
SensorId	int(11)	DayaBay::DetectorSensor	unique id of this pmt	.fullPackedDat
SpeLow	float	double	SPE mean ADC value (low gain)	
SpeLowError	float	double	error in SPE Mean ADC value (low gain)	

Table 19.14: DBI Table specification for class GCalibPmtLowGain which corresponds to table CalibPmtLowGain

[14] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtLowGain.spec

19.15 CalibPmtPedBias

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtPedBias.spec[15]

docstring

name	dbtype	codetype	description
SensorId	int(11)	DayaBay::DetectorSensor	packed sensor ID of this PMT
Amp_b	double	double	y-intercept of ADC dependence of Gaussian amplitude
Amp_m	double	double	slope of ADC dependence of Gaussian amplitude
Mean_amp	double	double	amplitude of Gaussian portion of GausLine for pulses's
Mean_mean	double	double	mean of Gaussian portion of GausLine for pulse's
Mean_sigma	double	double	sigma of Gaussian portion of GausLine for pulses's
Mean_offset	double	double	offset of Gaussian portion of GausLine for pulses&#39
Mean_b	double	double	y-intercept of linear portion of GausLine for pulses&#39
Mean_m	double	double	slope of linear portion of GausLine for the pulses&#39
Sigma_amp	double	double	amplitude of Gaussian portion of GausLine for pulses&#39
Sigma_mean	double	double	mean of Gaussian portion of GausLine for pulse's
Sigma_sigma	double	double	sigma of Gaussian portion of GausLine for pulses's
Sigma_offset	double	double	offset of Gaussian portion of GausLine for pulses&#39
Sigma_b	double	double	y-intercept of linear portion of GausLine for pulses&#39
Sigma_m	double	double	slope of linear portion of GausLine for the pulses&#39
A0_b	double	double	y-intercept of line used to model ADC dependence of co
A0_m	double	double	slope of line used to model ADC dependence of constant
A1_b	double	double	y-intercept of line used to model ADC dependence of li
A1_m	double	double	slope of line used to model ADC dependence of linear t
A2_b	double	double	y-intercept of line used to model ADC dependence of qu
A2_m	double	double	slope of line used to model ADC dependence of linear c
A3_b	double	double	y-intercept of line used to model ADC dependence of cu
A3_m	double	double	slope of line used to model ADC dependence of linear c

Table 19.15: DBI Table specification for class GCalibPmtPedBias which corresponds to table CalibPmtPedBias

[15] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtPedBias.spec

19.16 CalibPmtSpec

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtSpec.spec[16]

docstring

name	dbtype	codetype	description	code2db
PmtId	int(11)	int	-	
Describ	varchar(27)	string	String of decribing PMT position	
Status	tinyint(4)	int	Status check on the PMT	
SpeHigh	float	double	Single photoelectron mean ADC value (high gain)	
SigmaSpeHigh	float	double	Single p.e. 1-sigma peak width (high gain)	
SpeLow	float	double	ADC per P.E. ratio for low gain ADC channel	
TimeOffset	float	double	Relative transit time offset	
TimeSpread	float	double	Transit time spread	
Efficiency	float	double	Absolute efficiency	
PrePulseProb	float	double	Probability of prepulsing	
AfterPulseProb	float	double	Probability of afterpulsing	
DarkRate	float	double	Dark Rate	

Table 19.16: DBI Table specification for class GCalibPmtSpec which corresponds to table CalibPmtSpec

[16] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtSpec.spec

19.17 CalibPmtTimOff

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtTimOff.spec[17]

docstring

name	dbtype	codetype	description	code2db
ChannelId	int(11)	DayaBay::FeeChannelId	unique id of the channel connected to this pmt	.fullPac
Offset	float	double	Time offset (ns)	
OffsetError	float	double	Time offset error (ns)	

Table 19.17: DBI Table specification for class GCalibPmtTimOff which corresponds to table CalibPmtTimOff

[17] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtTimOff.spec

19.18 CalibPmtTiming

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtTiming.spec[18]

docstring

name	dbtype	codetype	description	code2db
ChannelId	int(11)	DayaBay::FeeChannelId	unique id of the channel connected to this pmt	.fullPa
Status	tinyint(4)	int	Status of this entry	
Par0	float	double	Parameter #0 for time correction	
Par1	float	double	Parameter #1 for time correction	
Par2	float	double	Parameter #2 for time correction	
Par3	float	double	Parameter #3 for time correction	
Par4	float	double	Parameter #4 for time correction	
Par5	float	double	Parameter #5 for time correction	
FitQual	float	double	Fit quality (could be chi2/ndf, for instance)	

Table 19.18: DBI Table specification for class GCalibPmtTiming which corresponds to table CalibPmtTiming

[18] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibPmtTiming.spec

19.19 CalibRpcSpec

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibRpcSpec.spec[19]

```
Calibration data for RPC layers

.. warning:: the attributes ‘PanelRow‘, ‘PanelColumn‘ and ‘Layer‘ duplicate information within ‘RpcSens
```

name	dbtype	codetype	description	code2db
RpcSensorId	int	DayaBay::RpcSensor	ID of RPC sensor for layer	.fullPackedData()
Efficiency	float	double	Efficiency of one RPC layer	
EfficiencyError	float	double	Efficiency error of one RPC layer	
NoiseRate	float	double	Noise rate of one RPC layer	
NoiseRateError	float	double	Noise rate error of one RPC layer	

Table 19.19: DBI Table specification for class GCalibRpcSpec which corresponds to table CalibRpcSpec

[19] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibRpcSpec.spec

19.20 CalibSrcEnergy

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCalibSrcEnergy.spec[20]

```
docstring

This is for energy scale calibration constants from calibration sources.
```

name	dbtype	codetype	description
SourcePeakType	int(11)	int	source:dybgaudi/trunk/DataModel/Conventions/Conventions/Calibrat
XSrcPosition	float	double	X position of calibration source
YSrcPosition	float	double	Y position of calibration source
ZSrcPosition	float	double	Z position of calibration source
PEPeak	float	double	peak PE value of Ge source
PEPeakUnc	float	double	uncertainty in peak value (%)
PEPeakFitQuality	float	double	quality of peak fit

Table 19.20: DBI Table specification for class GCalibSrcEnergy which corresponds to table CalibSrcEnergy

[20] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCalibSrcEnergy.spec

19.21 CoordinateAd

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCoordinateAd.spec[21]

```
Coordinates for AD support platforms in meters, see :docdb:‘6375‘ TABLE II.

.. rubric:: Detector convention in Offline and :docdb:‘6375‘

==========  ============== ======
AD name     fullPackedData  AdNo
==========  ============== ======
DayaBayAD1  16842752        1
DayaBayAD2  16908288        2
LingAoAD1   33619968        3
LingAoAD2   33685504        4
FarAD1      67174400        5
FarAD2      67239936        7
FarAD3      67305472        8
FarAD4      67371008        6
==========  ============== ======
```

name	dbtype	codetype	description
AdNo	tinyint	int	AdNo defined in :docdb:‘6375‘ Figure. 1
Detector	int	DayaBay::Detector	source:dybgaudi/trunk/DataModel/Conventions/Conventions/Detecto
X	double	double	coordinate X in meters
Y	double	double	coordinate Y in meters
Z	double	double	coordinate Z in meters

Table 19.21: DBI Table specification for class GCoordinateAd which corresponds to table CoordinateAd

[21] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCoordinateAd.spec

19.22 CoordinateReactor

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GCoordinateReactor.spec[22]

```
Coordinates for reactors, see :docdb:‘6375‘ TABLE II.

.. rubric:: convention in Offline and :docdb:‘6375‘

=========  ========== ======
name       ReactorId  :docdb:‘6375‘
=========  ========== ======
DayaBayA   0x01        D1
DayaBayB   0x02        D2
LingAoIA   0x04        L1
LingAoIB   0x08        L2
LingAoIIA  0x10        L3
LingAoIIB  0x20        L4
=========  ========== ======
```

name	dbtype	codetype	description	code
ReactorId	int	int	source:dybgaudi/trunk/DataModel/Conventions/Conventions/Reactor.h	
X	double	double	coordinate X in meters	
Y	double	double	coordinate Y in meters	
Z	double	double	coordinate Z in meters	

Table 19.22: DBI Table specification for class GCoordinateReactor which corresponds to table CoordinateReactor

[22] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GCoordinateReactor.spec

19.23 DaqCalibRunInfo

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDaqCalibRunInfo.spec[23]

```
Calibration run information recorded in DAQ database from IS/ACU
This information can also be accessed from raw data file recorded as

* :dybgaudi:‘DaqFormat/FileReadoutFormat/FileTraits.h‘

References:

* :docdb:‘3442‘
* :docdb:‘3603‘
```

name	dbtype	codetype	description
RunNo	int(11)	int	Run number
DetectorId	int(11)	int	0xPQ, P:site, Q:pit number
AdNo	int(11)	int	AD Number: 1-8
SourceIdA	int(11)	int	source:dybgaudi/trunk/DataModel/Conventions/Conventions/Calibration.h
ZPositionA	int(11)	int	Z position in mm of ACU A relative to target center
SourceIdB	int(11)	int	source in ACU B
ZPositionB	int(11)	int	Z position in ACU B
SourceIdC	int(11)	int	source in ACU C
ZPositionC	int(11)	int	Z position in ACU C
Duration	int(11)	int	Duration of DAQ run in seconds
LedNumber1	int(11)	int	ID number of the first LED being pulsed (1-6)
LedNumber2	int(11)	int	ID number of the second LED being pulsed (1-6)
LedVoltage1	int(11)	int	Voltage in mV for first LED
LedVoltage2	int(11)	int	Voltage in mV for second LED
LedFreq	int(11)	int	LED pulsing frequency in Hz
LedPulseSep	int(11)	int	Seperation time in ns for double LED run
LtbMode	int(11)	int	LTB trigger mode (True = Forced)
HomeA	int(11)	int	True if the selected source in ACU A is at home
HomeB	int(11)	int	True if the selected source in ACU B is at home
HomeC	int(11)	int	True if the selected source in ACU C is at home

Table 19.23: DBI Table specification for class GDaqCalibRunInfo which corresponds to table DaqCalibRunInfo

[23] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDaqCalibRunInfo.spec

19.24 DaqRawDataFileInfo

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDaqRawDataFileInfo.spec[24]

docstring

name	dbtype	codetype	description	code2db
RunNo	int(10) unsigned	int	-	
FileNo	int(10) unsigned	int	-	
FileName	tinytext	string	-	
StreamType	varchar(32)	string	-	
Stream	varchar(32)	string	-	
FileState	varchar(32)	string	-	
FileSize	int(11)	int	-	
CheckSum	varchar(64)	string	-	
TransferState	varchar(32)	string	-	

Table 19.24: DBI Table specification for class GDaqRawDataFileInfo which corresponds to table DaqRawDataFileInfo

[24] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDaqRawDataFileInfo.spec

19.25 DaqRunInfo

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDaqRunInfo.spec[25]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	-	
TriggerType	bigint(20)	int	-	
RunType	varchar(32)	string	-	
DetectorMask	int(11)	int	-	
PartitionName	varchar(255)	string	-	
SchemaVersion	int(11)	int	-	
DataVersion	int(11)	int	-	
BaseVersion	int(11)	int	-	

Table 19.25: DBI Table specification for class GDaqRunInfo which corresponds to table DaqRunInfo

[25] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDaqRunInfo.spec

19.26 DataQualityDetector

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDataQualityDetector.spec[26]

docstring

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
DetectorId	int(11)	int	Detector ID	
SinglesRate	float	double	Singles Rate	
IBDRate	float	double	IBD rate (number)	
SPNRate	float	double	Spallation neutron rate	
MuonRate	float	double	Muon Rate	

Table 19.26: DBI Table specification for class GDataQualityDetector which corresponds to table DataQualityDetector

[26] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDataQualityDetector.spec

19.27 DataQualityGoodRun

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDataQualityGoodRun.spec[27]

docstring

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
PmtHealth	tinyint	int	PMT condition of this file	
RpcHealth	tinyint	int	RPC condition of this file	
TriggerHealth	tinyint	int	Trigger condition of this file	
Singles	tinyint	int	Singles Rate normal or not	
IBD	tinyint	int	IBD rate (number) normal or not	
SPN	tinyint	int	Spallation neutron rate normal or not	
Reactor	tinyint	int	Reactor normal or not	
GOOD	tinyint	int	Indicate whether a good run for physics analysis or not	

Table 19.27: DBI Table specification for class GDataQualityGoodRun which corresponds to table DataQualityGoodRun

[27] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDataQualityGoodRun.spec

19.28 DataQualityPmt

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDataQualityPmt.spec[28]

`docstring`

name	dbtype	codetype	description
RunNo	int(11)	int	Run Number
FileNo	int(11)	int	File Number
PmtId	int(11)	DayaBay::DetectorSensor	Packed PMT ID
Status	tinyint	int	Fitting Resutls Status
Chi2ndf	float	double	Chi2/ndf for gain fitting
Gain	float	double	Single photoelectron mean ADC value (high gain)
GainErr	float	double	Gain Error
DarkRate	float	double	Dark Rate
DarkRateErr	float	double	Dark Rate Error
ElecNoiseRate	float	double	Electronics Noise Rate
ElecNoiseRateErr	float	double	Electronics Noise Rate Error
PreAdc	float	double	Pre Adc
PreAdcErr	float	double	Pre Adc Error

Table 19.28: DBI Table specification for class GDataQualityPmt which corresponds to table DataQualityPmt

[28] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDataQualityPmt.spec

19.29 DataQualityRpc

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDataQualityRpc.spec[29]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
RpcSensorId	int	DayaBay::RpcSensor	ID of RPC sensor for layer	.fullPackedData()
Efficiency	float	double	Efficiency of one RPC layer	
EfficiencyErr	float	double	Efficiency error of one RPC layer	
NoiseRate	float	double	Noise rate of one RPC layer	
NoiseRateErr	float	double	Noise rate error of one RPC layer	

Table 19.29: DBI Table specification for class GDataQualityRpc which corresponds to table DataQualityRpc

[29] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDataQualityRpc.spec

19.30 DataQualityTrigger

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDataQualityTrigger.spec[30]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
TrigRate	float	double	Trigger Rate	
FlasherRate	float	double	Flasher Rate	
BlockTrigFrac	float	double	Blocked Trigger Fraction	

Table 19.30: DBI Table specification for class GDataQualityTrigger which corresponds to table DataQualityTrigger

[30] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDataQualityTrigger.spec

19.31 DcsAdPmtHv

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsAdPmtHv.spec[31]

```
PMT High Voltage monitoring table for AD::

mysql&gt; describe DcsAdPmtHv  ;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| SEQNO       | int(11)      | NO   | PRI |         |                |
| ROW_COUNTER | int(11)      | NO   | PRI | NULL    | auto_increment |
| ladder      | tinyint(4)   | YES  |     | NULL    |                |
| col         | tinyint(4)   | YES  |     | NULL    |                |
| ring        | tinyint(4)   | YES  |     | NULL    |                |
| voltage     | float        | YES  |     | NULL    |                |
| pw          | tinyint(4)   | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
7 rows in set (0.07 sec)
```

name	dbtype	codetype	description	code2db
Ladder	tinyint(4)	int	PMT ladder	
Column	tinyint(4)	int	PMT column	
Ring	tinyint(4)	int	PMT ring	
Voltage	float	float	PMT Voltage	
Pw	tinyint(4)	int	PMT Power ON/OFF	

Table 19.31: DBI Table specification for class GDcsAdPmtHv which corresponds to table DcsAdPmtHv

[31] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsAdPmtHv.spec

19.32 DcsAdTemp

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsAdTemp.spec[32]

```
AD Temperature monitoring table::

mysql&gt; describe DcsAdTemp  ;
+-------------+---------+------+-----+---------+----------------+
| Field       | Type    | Null | Key | Default | Extra          |
+-------------+---------+------+-----+---------+----------------+
| SEQNO       | int(11) | NO   | PRI |         |                |
| ROW_COUNTER | int(11) | NO   | PRI | NULL    | auto_increment |
| Temp_PT1    | float   | YES  |     | NULL    |                |
| Temp_PT2    | float   | YES  |     | NULL    |                |
| Temp_PT3    | float   | YES  |     | NULL    |                |
| Temp_PT4    | float   | YES  |     | NULL    |                |
+-------------+---------+------+-----+---------+----------------+
6 rows in set (0.08 sec)

DBI read must explicitly give: Site, SubSite/DetectoId
DBI write must explicitly give: SiteMask, SubSite
```

name	dbtype	codetype	description	code2db
Temp1	float	float	AD?_temp_pt1	
Temp2	float	float	AD?_temp_pt2	
Temp3	float	float	AD?_temp_pt3	
Temp4	float	float	AD?_temp_pt4	

Table 19.32: DBI Table specification for class GDcsAdTemp which corresponds to table DcsAdTemp

[32] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsAdTemp.spec

19.33 DcsAdWpHv

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsAdWpHv.spec[33]

```
PMT High Voltage monitoring table::

mysql&gt; describe DcsAdWpHv;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| SEQNO       | int(11) | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11) | NO   | PRI | NULL    |       |
| LocationId  | char(6) | YES  |     | NULL    |       |
| Voltage     | float   | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
```

name	dbtype	codetype	description	code2db
LocationId	char(6)	string	PMT Location ID	
Voltage	float	float	PMT Voltage	

Table 19.33: DBI Table specification for class GDcsAdWpHv which corresponds to table DcsAdWpHv

[33] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsAdWpHv.spec

19.34 DcsAdWpHvShunted

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsAdWpHvShunted.spec[34]

```
PMT High Voltage monitoring table::

mysql&gt; describe DcsAdWpHv;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| SEQNO       | int(11) | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11) | NO   | PRI | NULL    |       |
| LocationId  | char(6) | YES  |     | NULL    |       |
| Voltage     | float   | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
```

name	dbtype	codetype	description	code2db
LocationId	char(6)	string	PMT Location ID	
Voltage	float	float	PMT Voltage	

Table 19.34: DBI Table specification for class GDcsAdWpHvShunted which corresponds to table DcsAdWpHvShunted

[34] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsAdWpHvShunted.spec

19.35 DcsMuonCalib

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsMuonCalib.spec[35]

```
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| SEQNO       | int(11) | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11) | NO   | PRI | NULL    |       |
| LedId       | int(11) | YES  |     | NULL    |       |
| Voltage     | float   | YES  |     | NULL    |       |
| Frequency   | float   | YES  |     | NULL    |       |
| ChannelId   | int(11) | YES  |     | NULL    |       |
| ErrorCode   | int(11) | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
7 rows in set (0.02 sec)
```

name	dbtype	codetype	description	code2db
LedId	int	int	IOW_CAL_LED_ID	
Voltage	float	float	IOW_CAL_LED_ID_Voltage	
Frequency	float	float	IOW_CAL_LED_ID_Frequency	
ChannelId	int	int	IOW_CAL_Channel_ID	
ErrorCode	int	int	IOW_CAL_ErrorCode	

Table 19.35: DBI Table specification for class GDcsMuonCalib which corresponds to table DcsMuonCalib

[35] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsMuonCalib.spec

19.36 DcsPmtHv

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsPmtHv.spec[36]

```
PMT High Voltage monitoring table::

mysql&gt; describe DcsPmtHv  ;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| SEQNO       | int(11)      | NO   | PRI |         |                |
| ROW_COUNTER | int(11)      | NO   | PRI | NULL    | auto_increment |
| ladder      | tinyint(4)   | YES  |     | NULL    |                |
| col         | tinyint(4)   | YES  |     | NULL    |                |
| ring        | tinyint(4)   | YES  |     | NULL    |                |
| voltage     | decimal(6,2) | YES  |     | NULL    |                |
| pw          | tinyint(4)   | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
7 rows in set (0.07 sec)
```

name	dbtype	codetype	description	code2db
Ladder	tinyint(4)	int	PMT ladder	
Column	tinyint(4)	int	PMT column	
Ring	tinyint(4)	int	PMT ring	
Voltage	float	float	PMT Voltage	
Pw	tinyint(4)	int	PMT Power ON/OFF	

Table 19.36: DBI Table specification for class GDcsPmtHv which corresponds to table DcsPmtHv

[36] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsPmtHv.spec

19.37 DcsRpcHv

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsRpcHv.spec[37]

```
PMT High Voltage monitoring table::

mysql&gt; describe DcsRpcHv;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| SEQNO       | int(11)     | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11)     | NO   | PRI | NULL    |       |
| LocationId  | char(3)     | YES  |     | NULL    |       |
| VoltagePos  | float       | YES  |     | NULL    |       |
| VoltageNeg  | float       | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
```

name	dbtype	codetype	description	code2db
LocationId	char(3)	string	Location ID	
VoltagePos	float	float	Positive RPC Voltage	
VoltageNeg	float	float	Negative RPC Voltage	

Table 19.37: DBI Table specification for class GDcsRpcHv which corresponds to table DcsRpcHv

[37] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsRpcHv.spec

19.38 DcsWpPmtHv

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDcsWpPmtHv.spec[38]

```
PMT High Voltage monitoring table for water pool
PMT naming in water pool::

    XXXXNN (4 letters + (number or letter) + number)

Where:

* 1st letter: experiment site (D, L, F).
* 2nd letter: water pool (V: outer pool, C: inner pool).
* 3rd letter: PMT direction (I: facing toward the center of the pool, O: facing away from the center of
* 4th letter: wall (A, B, C, D, E, F, G, H), floor treated like another wall, U.
* 1st number/letter: row number counting from bottom up or F for floor PMT within the wall.
* 2nd number: column number counting from left to right when facing the wall standing in the middle of
```

name	dbtype	codetype	description	code2db
Direction	char(1)	char	PMT facing direction	
Wall	char(1)	char	PMT wall	
PmtRow	tinyint(4)	int	PMT row	
PmtColumn	tinyint(4)	int	PMT column	
Voltage	float	float	PMT Voltage	
Pw	tinyint(4)	int	PMT Power ON/OFF	

Table 19.38: DBI Table specification for class GDcsWpPmtHv which corresponds to table DcsWpPmtHv

[38] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDcsWpPmtHv.spec

19.39 Demo

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDemo.spec[39]

```
Simple Class/Table for demonstration of DybDbi/DBI features
```

name	dbtype	codetype	description	code2db
Gain	double	double	demonstration double	
Id	int(11)	int	demonstration identity	

Table 19.39: DBI Table specification for class GDemo which corresponds to table Demo

[39] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDemo.spec

19.40 DemoAgg

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDemoAgg.spec[40]

```
Simple Class/Table for demonstration of DybDbi/DBI features, modified to be an aggregate type with some
template handling for meta key ``aggrow``.
Adding meta key ``aggrow=Id`` to add the ``GetAggregateNo`` method which returns ``m_Id`` to provide th
which must be ``0,1,2,3,4,...``
```

name	dbtype	codetype	description	code2db
Gain	double	double	demonstration double	
Id	int(11)	int	demonstration identity	

Table 19.40: DBI Table specification for class GDemoAgg which corresponds to table DemoAgg

[40] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDemoAgg.spec

19.41 DemoBit

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDemoBit.spec[41]

```
GDemoBit
---------

See :dybgaudi:‘Database/DybDbi/tests/test_demo_bitfield.py‘

.. warning:: Due to a DBI/ROOT reading limitation restrict the bit fields to avoid the most significant
```

name	dbtype	codetype	description	code2db
Gain	double	double	demonstration double	
Id	int(11)	int	demonstration identity	
Mask0	int(11)	Int_t	demonstration BitField	
Mask1	int(11)	Int_t	demonstration BitField	
Mask2	int(11)	Int_t	demonstration BitField	
Mask3	int(11)	Int_t	demonstration BitField	
Mask4	int(11)	Int_t	demonstration BitField	
Mask5	int(11)	Int_t	demonstration BitField	

Table 19.41: DBI Table specification for class GDemoBit which corresponds to table DemoBit

[41] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDemoBit.spec

19.42 DqChannel

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqChannel.spec[42]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
ChannelId	int(11)	DayaBay::FeeChannelId	Unique id of the channel connected to a pmt	.fullPackedDa
Occupancy	float	double	Channel Occupancy	
DAdcMean	float	double	Delta Adc Mean	
DAdcRMS	float	double	Delta Adc RMS	
HvMean	float	double	HV mean value within the file	
HvRMS	float	double	HV RMS value within the file	

Table 19.42: DBI Table specification for class GDqChannel which corresponds to table DqChannel

[42] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqChannel.spec

19.43 DqChannelPacked

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqChannelPacked.spec[43]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
Mask0	int(11)	Int_t	Mask0, bit packing of channel status	
Mask1	int(11)	Int_t	Mask1, bit packing of channel status	
Mask2	int(11)	Int_t	Mask2, bit packing of channel status	
Mask3	int(11)	Int_t	Mask3, bit packing of channel status	
Mask4	int(11)	Int_t	Mask4, bit packing of channel status	
Mask5	int(11)	Int_t	Mask5, bit packing of channel status	
Mask6	int(11)	Int_t	Mask6, bit packing of channel status	

Table 19.43: DBI Table specification for class GDqChannelPacked which corresponds to table DqChannelPacked

[43] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqChannelPacked.spec

19.44 DqChannelStatus

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqChannelStatus.spec[44]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
ChannelId	int(11)	DayaBay::FeeChannelId	Unique id of the channel connected to a pmt	.fullPackedDa
Status	tinyint	int	Channel status (good/bad)	

Table 19.44: DBI Table specification for class GDqChannelStatus which corresponds to table DqChannelStatus

[44] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqChannelStatus.spec

19.45 DqDetector

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqDetector.spec[45]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
TriggerCounts	int(11)	int	Trigger Counts	
FlasherCounts	int(11)	int	Flasher Counts	
MuonCounts	int(11)	int	Muon Counts	
IbdCounts	int(11)	int	IBD Number	
SpnCounts	int(11)	int	Spallation Neutron Number	
BlockTrigCounts	int(11)	int	Blocked Trigger Number	
SpnEnergy	float	double	Spallation Neutron Energy	
SpnEnergySigma	float	double	Sigma of Spallation Neutron Energy	

Table 19.45: DBI Table specification for class GDqDetector which corresponds to table DqDetector

[45] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqDetector.spec

19.46 DqDetectorExt

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqDetectorExt.spec[46]

docstring

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
ADMuonCounts	int(11)	int	AD Muon Counts	
IWSTaggedCounts	int(11)	int	IWS Tagged AD Muon Counts	
OWSTaggedCounts	int(11)	int	OWS Tagged AD Muon Counts	
WSTaggedCounts	int(11)	int	WS Combined Tagged AD Muon Counts	
Var1	float	double	Place Holder	
Var2	float	double	Place Holder	
Var3	float	double	Place Holder	
Var4	float	double	Place Holder	
Var5	float	double	Place Holder	
Var6	float	double	Place Holder	
Var7	float	double	Place Holder	
Var8	float	double	Place Holder	
Var9	float	double	Place Holder	
Var10	float	double	Place Holder	

Table 19.46: DBI Table specification for class GDqDetectorExt which corresponds to table DqDetectorExt

[46] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqDetectorExt.spec

19.47 DqDetectorNew

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqDetectorNew.spec[47]

docstring

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
TriggerCounts	int(11)	int	Trigger Counts	
FlasherCounts	int(11)	int	Flasher Counts	
MuonCounts	int(11)	int	Muon Counts	
IbdCounts	int(11)	int	IBD Number	
SpnCounts	int(11)	int	Spallation Neutron Number	
BlockTrigCounts	int(11)	int	Blocked Trigger Number	
SpnEnergy	float	double	Spallation Neutron Energy	
SpnEnergySigma	float	double	Sigma of Spallation Neutron Energy	
K40Energy	float	double	K40 Peak Energy	
K40EnergyErr	float	double	Error of K40 Peak Energy	
Tl208Energy	float	double	Tl208 Peak Energy	
Tl208EnergyErr	float	double	Error of Tl208 Peak Energy	
PLikeCounts	int(11)	int	Positron Like Event Number	
NLikeCounts	int(11)	int	Neutron Like Event Number	
DtNegCounts	int(11)	int	Number of Events with Dt_trigger<-12.5ns	
DtLargeGapCounts	int(11)	int	Large Dt Gap Event Number	
NHitCountsGt	int(11)	int	NHit Trigger Counts With Energy>0.7MeV	
ESumCountsGt	int(11)	int	ESum Trigger Counts With Energy>0.7MeV	

Table 19.47: DBI Table specification for class GDqDetectorNew which corresponds to table DqDetectorNew

[47] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqDetectorNew.spec

19.48 DqLiveTime

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqLiveTime.spec[48]

`docstring`

name	dbtype	codetype	description	code2d
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
IntegralRunTime	double	double	DAQ Running Time (ms)	
IntegralLiveTimeBlocked	double	double	DAQ runtime with blocked trigger correction (ms)	
IntegralLiveTimeBuffer	double	double	DAQ runtime with buffer full correction (ms)	

Table 19.48: DBI Table specification for class GDqLiveTime which corresponds to table DqLiveTime

[48] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqLiveTime.spec

19.49 DqPmt

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqPmt.spec[49]

docstring

name	dbtype	codetype	description
RunNo	int(11)	int	Run Number
FileNo	int(11)	int	File Number
PmtId	int(11)	DayaBay::DetectorSensor	Packed PMT ID
Status	tinyint	int	Fitting Resutls Status
Chi2ndf	float	double	Chi2/ndf for gain fitting
Gain	float	double	Single photoelectron mean ADC value (high gain)
GainErr	float	double	Gain Error
DarkRate	float	double	Dark Rate
DarkRateErr	float	double	Dark Rate Error
ElecNoiseRate	float	double	Electronics Noise Rate
ElecNoiseRateErr	float	double	Electronics Noise Rate Error
PreAdc	float	double	Pre Adc
PreAdcErr	float	double	Pre Adc Error
AdcMean	float	double	Adc Mean
AdcRMS	float	double	Adc RMS
TdcMean	float	double	Tdc Mean
TdcRMS	float	double	Tdc RMS
FlashingCounts	int(11)	int	Flashing Counts
DNCountsForced	int(11)	int	Dark Noise Counts Calculated from Forced Trigger
Quality	tinyint	int	Quality Flag for a Pmt Channel

Table 19.49: DBI Table specification for class GDqPmt which corresponds to table DqPmt

[49] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqPmt.spec

19.50 DqPmtNew

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqPmtNew.spec[50]

docstring

name	dbtype	codetype	description	c
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
ChannelId	int(11)	DayaBay::FeeChannelId	Unique id of the channel connected to a pmt	.
Status	tinyint	int	Fitting Resutls Status	
Chi2ndf	float	double	Chi2/ndf for gain fitting	
Gain	float	double	Single photoelectron mean ADC value (high gain)	
GainErr	float	double	Gain Error	
DarkRate	float	double	Dark Rate	
DarkRateErr	float	double	Dark Rate Error	
ElecNoiseRate	float	double	Electronics Noise Rate	
ElecNoiseRateErr	float	double	Electronics Noise Rate Error	
PreAdc	float	double	Pre Adc	
PreAdcErr	float	double	Pre Adc Error	
AdcMean	float	double	Adc Mean	
AdcRMS	float	double	Adc RMS	
TdcMean	float	double	Tdc Mean	
TdcRMS	float	double	Tdc RMS	
FlashingCounts	int(11)	int	Flashing Counts	
DNCountsForced	int(11)	int	Dark Noise Counts Calculated from Forced Trigger	
Quality	tinyint	int	Quality Flag for a Pmt Channel	

Table 19.50: DBI Table specification for class GDqPmtNew which corresponds to table DqPmtNew

[50] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqPmtNew.spec

19.51 DqTriggerCounts

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqTriggerCounts.spec[51]

docstring

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
CrossInCounts	int(11)	int	Cross In Trigger Type Events Counts	
PeriodicCounts	int(11)	int	Periodic Trigger Type Events Counts	
CalibCounts	int(11)	int	Calib Trigger Type Events Counts	
RandomCounts	int(11)	int	Random Trigger Type Events Counts	
NhitCounts	int(11)	int	NHit Trigger Type Events Counts	
EsumCounts	int(11)	int	ESum Trigger Type Events Counts	
Rpc2of4Counts	int(11)	int	RPC (2 out of 4) Trigger Type Events Counts	
Rpc3of4Counts	int(11)	int	RPC (3 out of 4) Trigger Type Events Counts	

Table 19.51: DBI Table specification for class GDqTriggerCounts which corresponds to table DqTriggerCounts

[51] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqTriggerCounts.spec

19.52 DqWPMonitoring

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GDqWPMonitoring.spec[52]

`docstring`

name	dbtype	codetype	description	code2db
RunNo	int(11)	int	Run Number	
FileNo	int(11)	int	File Number	
DetectorId	int(11)	int	Detector ID	
MuonNhit	float	double	Muon Pmt Nhit	
MuonNhitErr	float	double	Muon Pmt Nhit Error	
MuonPESum	float	double	Muon Pmt PE Sum	
MuonPESumErr	float	double	Muon Pmt PE Sum Error	

Table 19.52: DBI Table specification for class GDqWPMonitoring which corresponds to table DqWPMonitoring

[52] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GDqWPMonitoring.spec

19.53 EnergyPositionCorr

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GEnergyPositionCorr.spec[53]

`docstring`

name	dbtype	codetype	description	code2db
NonUniformCorr1	float	double	nonuniformity parameter 1	
NonUniformCorr2	float	double	nonuniformity parameter 2	
NonUniformCorr3	float	double	nonuniformity parameter 3	
NonUniformCorr4	float	double	nonuniformity parameter 4	

Table 19.53: DBI Table specification for class GEnergyPositionCorr which corresponds to table EnergyPositionCorr

[53] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GEnergyPositionCorr.spec

19.54 EnergyRecon

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GEnergyRecon.spec[54]

`docstring`

name	dbtype	codetype	description	code2db
PeEvis	float	double	pe per mev value	
PeEvisUnc	float	double	pe per mev uncertainty	

Table 19.54: DBI Table specification for class GEnergyRecon which corresponds to table EnergyRecon

[54] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GEnergyRecon.spec

19.55 FeeCableMap

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GFeeCableMap.spec[55]

```
Data members of instances of the generated class use specialized types, which are specified for each
field by the ``codetype`` column.

  ==============================  ======================================  ==============================
    codetype                        API ref                                       defined
  ==============================  ======================================  ==============================
   ``DayaBay::FeeChannelId``       :py:class:`DybDbi.FeeChannelId`         :conventions:`Electronics.h`
   ``DayaBay::FeeHardwareId``      :py:class:`DybDbi.FeeHardwareId`        :conventions:`Hardware.h`
   ``DayaBay::DetectorSensor``     :py:class:`DybDbi.DetectorSensor`       :conventions:`Detectors.h`
   ``DayaBay::PmtHardwareId``      :py:class:`DybDbi.PmtHardwareId`        :conventions:`Hardware.h`
  ==============================  ======================================  ==============================

This usage is mirrored in the ctor/getters/setters of the generated class.
As these cannot be directly stored into the DB, conversions are performed on writing and reading.

On writing to DB the ``code2db`` defined call is used to convert the specialized type into integers
that can be persisted in the DB.  On reading from the DB the one argument ``codetype`` ctors are used
to convert the persisted integer back into the specialized types.
```

name	dbtype	codetype	description	code2db
FeeChannelId	int(11)	DayaBay::FeeChannelId	Front-end electronics channel	.fullPa
FeeChannelDesc	varchar(30)	string	String of decribing FEE channel	
FeeHardwareId	int(11)	DayaBay::FeeHardwareId	Identify a physical FEE board	.id()
ChanHrdwDesc	varchar(30)	string	String of decribing FEE board position	
SensorId	int(11)	DayaBay::DetectorSensor	PMT Sensor ID in detector	.fullPa
SensorDesc	varchar(30)	string	String of decribing PMT sensor id	
PmtHardwareId	int(11)	DayaBay::PmtHardwareId	Identify a physical PMT position	.id()
PmtHrdwDesc	varchar(30)	string	String of decribing PMT position	

Table 19.55: DBI Table specification for class GFeeCableMap which corresponds to table FeeCableMap

[55] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GFeeCableMap.spec

19.56 GoodRunList

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GGoodRunList.spec[56]

```
List of runs/files suitable for analysis
```

name	dbtype	codetype	description	code2db
RunNo	int	int	Run number	
FileNo	int	int	File sequence number	
StreamId	int	int	Output stream	

Table 19.56: DBI Table specification for class GGoodRunList which corresponds to table GoodRunList

[56] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GGoodRunList.spec

19.57 HardwareID

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GHardwareID.spec[57]

```
The Hardware ID table maps a HardwareId (from Conventions/Hardware.h)
to either a DetectorSensor (Conventions/Detectors.h) or an
ElecChannelId (Conventions/Electronics.h) depending on the harddware
type.

For the code, note the DetectorSensor and ElecChannelId are stored in
the form of the base class DayaBay::Detector
```

name	dbtype	codetype	description	code2db
ChanOrSens	int(11)	DayaBay::Detector	Packed Channel/Sensor ID	.fullPackedData()
HardwareId	int(11)	DayaBay::HardwareId	Packed Hardware ID	.fullPackedData()

Table 19.57: DBI Table specification for class GHardwareID which corresponds to table HardwareID

[57] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GHardwareID.spec

19.58 HardwareIDFix

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GHardwareIDFix.spec[58]

```
Testing fix for :dybsvn:‘ticket:948‘

The Hardware ID table maps a HardwareId (from Conventions/Hardware.h)
to either a DetectorSensor (Conventions/Detectors.h) or an
ElecChannelId (Conventions/Electronics.h) depending on the harddware
type.

For the code, note the DetectorSensor and ElecChannelId are stored in
the form of the base class DayaBay::Detector
```

name	dbtype	codetype	description	code2db
ChanOrSens	int(11)	DayaBay::Detector	Packed Channel/Sensor ID	.fullPackedData()
HardwareId	int(11)	DayaBay::HardwareId	Packed Hardware ID	.fullPackedData()

Table 19.58: DBI Table specification for class GHardwareIDFix which corresponds to table HardwareID

[58] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GHardwareIDFix.spec

19.59 McsPos

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GMcsPos.spec[59]

```
Source position for MCS.

mysql&gt; describe McsPos;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| SEQNO       | int(11) | NO   | PRI | NULL    |       |
| ROW_COUNTER | int(11) | NO   | PRI | NULL    |       |
| PositionR   | float   | YES  |     | NULL    |       |
| PositionZ   | float   | YES  |     | NULL    |       |
| PositionPhi | float   | YES  |     | NULL    |       |
| RunNo       | int(11) | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
6 rows in set (0.02 sec)
```

name	dbtype	codetype	description	code2db
PositionR	float	float	r position of source (mm)	
PositionZ	float	float	z position of source (mm)	
PositionPhi	float	float	phi position of source (degrees)	
RunNo	int(11)	int	run number associated with this position	

Table 19.59: DBI Table specification for class GMcsPos which corresponds to table McsPos

[59] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GMcsPos.spec

19.60 PhysAd

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GPhysAd.spec[60]

```
This table can be read/written using :py:class:‘DybDbi.AdLogicalPhysical‘

(adapted from Dan/Zhimin email 2011-01-17)

There are two ways to identify an AD in the experiment:

   #. Location: SAB-AD1, ..., FAR-AD4
   #. Physical ID: AD1, AD2, ..., AD8



.. rubric:: Convention references

============   ============================================================
 Convention     Reference
============   ============================================================
 DCS            :docdb:‘3198‘
 DAQ            :docdb:‘3442‘ page 6
============   ============================================================

The Offline convention can be found from:
  * :dybgaudi:‘DataModel/Conventions/Conventions/Site.h‘
  * :dybgaudi:‘DataModel/Conventions/Conventions/DetectorId.h‘

Here is a summary of the Location names/IDs that each system uses:

.. rubric:: Site (Name and ID)

====  ==== ============ ======  ==========
DCS   DAQ  Offline      DAQ_ID  Offline_ID
====  ==== ============ ======  ==========
DBNS  DBN  DayaBay      0x10    0x01
LANS  LAN  LingAo       0x20    0x02
FARS  FAR  Far          0x30    0x04
MIDS  MID  Mid          .       0x08
.     .    Aberdeen     .       0x10
SAB   SAB  SAB          0x60    0x20
.     .    PMTBenchTest .       0x40
LSH   .    .            .       .
====  ==== ============ ======  ==========


.. rubric:: Detector/MainSys (Name and ID)
```

[60] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GPhysAd.spec

DCS	DAQ	Offline	DAQ_ID	Offline_ID
AD1	AD1	AD1	0x01	0x01
AD2	AD2	AD2	0x02	0x02
AD3	AD3	AD3	0x03	0x03
AD4	AD4	AD4	0x04	0x04
IWP	WPI	IWS	0x05	0x05
OWP	WPO	OWS	0x06	0x06
RPC	RPC	RPC	0x07	0x07
Muon	.	.	.	.
GAS	.	.	.	.
PMT	.	.	.	.
FEE	.	.	.	.
SIS	.	.	.	.

name	dbtype	codetype	description	code2db
PhysAdId	tinyint	int	ID of physical AD, counts 1 to 8	

Table 19.60: DBI Table specification for class GPhysAd which corresponds to table PhysAd

19.61 QSumCalib

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GQSumCalib.spec[61]

```
Calibration constant for QSum Reconstruction.
```

name	dbtype	codetype	description	code2db
PeYield	float	double	PE yield per MeV	
Uncertainty	float	double	uncertainty of PE yield	

Table 19.61: DBI Table specification for class GQSumCalib which corresponds to table QSumCalib

[61] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GQSumCalib.spec

19.62 SimPmtSpec

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GSimPmtSpec.spec[62]

docstring

name	dbtype	codetype	description	code2db
PmtId	int(11)	DayaBay::DetectorSensor	PMT sensor ID	.sensorId()
Describ	varchar(27)	string	String of decribing PMT position	
Gain	float	double	Relative gain for pmt, mean = 1	
SigmaGain	float	double	1-sigma spread of S.P.E. response	
TimeOffset	float	double	Relative transit time offset	
TimeSpread	float	double	Transit time spread	
Efficiency	float	double	Absolute efficiency	
PrePulseProb	float	double	Probability of prepulsing	
AfterPulseProb	float	double	Probability of afterpulsing	
DarkRate	float	double	Dark Rate	

Table 19.62: DBI Table specification for class GSimPmtSpec which corresponds to table SimPmtSpec

[62] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GSimPmtSpec.spec

19.63 SupernovaTrigger

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GSupernovaTrigger.spec[63]

```
Supernova Trigger Table DocDB:8854

.. rubric:: Detector convention in Offline and AdNo we define

==========  ============== ======
AD name     fullPackedData  AdNo
==========  ============== ======
DayaBayAD1  16842752        1
DayaBayAD2  16908288        2
LingAoAD1   33619968        3
LingAoAD2   33685504        4
FarAD1      67174400        5
FarAD2      67239936        6
FarAD3      67305472        7
FarAD4      67371008        8
==========  ============== ======
```

name	dbtype	codetype	description
detector	int(11)	DayaBay::Detector	source:dybgaudi/trunk/DataModel/Conventions/Conventions/Detec
runNo	int(11)	int	physics run number
fileNo	int(11)	int	raw data file number of a certain physics run
pTriggerNo	int(11)	int	prompt signal trigger number
pEnergy	double	double	prompt signal energy (MeV) after a simple recon
pTime_s	int(11)	int	prompt signal time stamp sec
pTime_nano	int(11)	int	prompt signal time stamp nano
pVertex_x	double	double	prompt signal vertex x position(mm) after a simple recon
pVertex_y	double	double	prompt signal vertex y position(mm) after a simple recon
pVertex_z	double	double	prompt signal vertex z position(mm) after a simple recon
dTriggerNo	int(11)	int	delayed signal trigger number
dEnergy	double	double	delayed signal energy (MeV) after a simple recon
dTime_s	int(11)	int	delayed signal time stamp sec
dTime_nano	int(11)	int	delayed signal time stamp nano
dVertex_x	double	double	delayed signal vertex x position(mm) after a simple recon
dVertex_y	double	double	delayed signal vertex y position(mm) after a simple recon
dVertex_z	double	double	delayed signal vertex z position(mm) after a simple recon

Table 19.63: DBI Table specification for class GSupernovaTrigger which corresponds to table SupernovaTrigger

[63] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GSupernovaTrigger.spec

19.64 TimeLatency

Table specification source dybgaudi/trunk/Database/DybDbi/spec/GTimeLatency.spec[64]

`docstring`

name	dbtype	codetype	description	code2db
Latency	float	double	time latency relative to a specific detector in one EH	

Table 19.64: DBI Table specification for class GTimeLatency which corresponds to table TimeLatency

[64] http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/DybDbi/spec/GTimeLatency.spec

Chapter 20

Bibliography

Bibliography

[1] Reference target needed for g4dyb