

---

# **Offline User Manual**

*Release 17726*

**Offline Group**

August 03, 2012



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Intended Audience . . . . .	3
1.2	Document Organization . . . . .	3
1.3	Contributing . . . . .	3
1.4	Building Documentation . . . . .	3
1.5	Typographical Conventions . . . . .	4
<b>2</b>	<b>Quick Start</b>	<b>5</b>
2.1	Offline Infrastructure . . . . .	5
2.2	Installation and Working with the Source Code . . . . .	5
2.3	Offline Framework . . . . .	6
2.4	Data Model . . . . .	6
2.5	Detector Description . . . . .	6
2.6	Kinematic Generators . . . . .	6
2.7	Detector Simulation . . . . .	6
2.8	Quick Start with Truth Information . . . . .	6
2.9	Electronics Simulation . . . . .	8
2.10	Trigger Simulation . . . . .	8
2.11	Readout . . . . .	8
2.12	Event Display . . . . .	9
2.13	Reconstruction . . . . .	11
2.14	Database . . . . .	11
<b>3</b>	<b>Analysis Basics</b>	<b>13</b>
3.1	Introduction . . . . .	13
3.2	Daya Bay Data Files . . . . .	13
3.3	NuWa Basics . . . . .	31
3.4	NuWa Recipes . . . . .	33
3.5	Cheat Sheets . . . . .	41
3.6	Hands-on Exercises . . . . .	57
<b>4</b>	<b>Offline Infrastructure</b>	<b>59</b>
4.1	Mailing lists . . . . .	59
4.2	DocDB . . . . .	59
4.3	Wikis . . . . .	59
4.4	Trac bug tracker . . . . .	59
<b>5</b>	<b>Installation and Working with the Source Code</b>	<b>61</b>
5.1	Using pre-installed release . . . . .	61

5.2	Installation of a Release . . . . .	62
5.3	Anatomy of a Release . . . . .	62
5.4	Version Control Your Code . . . . .	63
5.5	Technical Details of the Installation . . . . .	63
<b>6</b>	<b>Offline Framework</b>	<b>65</b>
6.1	Introduction . . . . .	65
6.2	Framework Components and Interfaces . . . . .	65
6.3	Common types of Components . . . . .	66
6.4	Writing your own component . . . . .	66
6.5	Properties and Configuration . . . . .	67
<b>7</b>	<b>Data Model</b>	<b>77</b>
7.1	Overview . . . . .	77
7.2	Times . . . . .	78
7.3	Examples of using the Data Model objects . . . . .	79
<b>8</b>	<b>Data I/O</b>	<b>81</b>
8.1	Goal . . . . .	81
8.2	Features . . . . .	81
8.3	Packages . . . . .	82
8.4	I/O Related Job Configuration . . . . .	82
8.5	How the I/O Subsystem Works . . . . .	82
8.6	Adding New Data Classes . . . . .	83
<b>9</b>	<b>Detector Description</b>	<b>91</b>
9.1	Introduction . . . . .	91
9.2	Conventions . . . . .	92
9.3	Coordinate System . . . . .	94
9.4	XML Files . . . . .	95
9.5	Transient Detector Store . . . . .	95
9.6	Configuring the Detector Description . . . . .	95
9.7	PMT Lookups . . . . .	95
9.8	Visualization . . . . .	95
<b>10</b>	<b>Kinematic Generators</b>	<b>97</b>
10.1	Introduction . . . . .	97
10.2	Generator output . . . . .	97
10.3	Generator Tools . . . . .	97
10.4	Generator Packages . . . . .	97
10.5	Types of GenTools . . . . .	98
10.6	Configuration . . . . .	98
10.7	MuonProphet . . . . .	101
<b>11</b>	<b>Detector Simulation</b>	<b>105</b>
11.1	Introduction . . . . .	105
11.2	Configuring DetSim . . . . .	105
11.3	Truth Information . . . . .	106
11.4	Truth Parameters . . . . .	115
<b>12</b>	<b>Electronics Simulation</b>	<b>121</b>
12.1	Introduction . . . . .	121
12.2	Algorithms . . . . .	123
12.3	Tools . . . . .	123
12.4	Simulation Constant . . . . .	124

<b>13</b>	<b>Trigger Simulation</b>	<b>127</b>
13.1	Introduction . . . . .	127
13.2	Configuration . . . . .	127
13.3	Current Triggers . . . . .	128
13.4	Adding a new Trigger . . . . .	128
<b>14</b>	<b>Readout</b>	<b>131</b>
14.1	Introduction . . . . .	131
14.2	ReadoutHeader . . . . .	131
14.3	SimReadoutHeader . . . . .	133
14.4	Readout Algorithms . . . . .	133
14.5	Readout Tools . . . . .	133
<b>15</b>	<b>Simulation Processing Models</b>	<b>135</b>
15.1	Introduction . . . . .	135
15.2	Fifteen . . . . .	135
<b>16</b>	<b>Reconstruction</b>	<b>145</b>
<b>17</b>	<b>Database</b>	<b>147</b>
17.1	Database Interface . . . . .	147
17.2	Concepts . . . . .	147
17.3	Running . . . . .	152
17.4	Accessing Existing Tables . . . . .	156
17.5	Creating New Tables . . . . .	162
17.6	Filling Tables . . . . .	168
17.7	ASCII Flat Files and Catalogues . . . . .	174
17.8	MySQL Crib . . . . .	176
17.9	Performance . . . . .	179
<b>18</b>	<b>Database Maintenance</b>	<b>181</b>
18.1	Introduction . . . . .	181
18.2	Building and Running dbmjob . . . . .	182
<b>19</b>	<b>Bibliography</b>	<b>185</b>
<b>20</b>	<b>Testing Code With Nose</b>	<b>187</b>
20.1	Introduction . . . . .	187
20.2	Using Test Attributes . . . . .	188
20.3	Running Tests Using dybinst . . . . .	190
20.4	Testing nose plugins . . . . .	191
<b>21</b>	<b>Standard Operating Procedures</b>	<b>195</b>
21.1	DB Definitions . . . . .	195
21.2	DBI Very Briefly . . . . .	197
21.3	Rules for Code that writes to the Database . . . . .	202
21.4	Configuring DB Access . . . . .	203
21.5	DB Table Updating Workflow . . . . .	212
21.6	DB Table Writing . . . . .	224
21.7	DB Table Reading . . . . .	231
21.8	Debugging unexpected parameters . . . . .	236
21.9	DB Table Creation . . . . .	239
21.10	DB Validation and Testing . . . . .	244
21.11	DB Administration . . . . .	247
21.12	DB Services . . . . .	248

21.13 DCS tables grouped/ordered by schema . . . . .	250
21.14 Non DBI access to DBI and other tables . . . . .	263
21.15 Scraping source databases into <i>offline_db</i> . . . . .	266
21.16 DBI Internals . . . . .	286
21.17 DBI Overlay Versioning Bug . . . . .	300
<b>22 Admin Operating Procedures for SVN/Trac/MySQL</b>	<b>319</b>
22.1 Env Repository : Admin Infrastructure Sources . . . . .	319
22.2 Source Code Management . . . . .	323
22.3 SSH Setup For Automated Rsync transfers . . . . .	325
<b>23 NuWa Python API</b>	<b>327</b>
23.1 DB . . . . .	327
23.2 DBAUX . . . . .	337
23.3 DBConf . . . . .	340
23.4 DBCas . . . . .	343
23.5 dbsvn - DBI SVN Gatekeeper . . . . .	344
23.6 DybDbiPre . . . . .	348
23.7 DybDbi . . . . .	349
23.8 DybPython . . . . .	414
23.9 DybPython.Control . . . . .	414
23.10 DybPython.dbicnf . . . . .	415
23.11 DbiDataSvc . . . . .	417
23.12 NonDbi . . . . .	417
23.13 Scraper . . . . .	421
23.14 DybTest . . . . .	431
<b>24 Documentation</b>	<b>435</b>
24.1 About This Documentation . . . . .	435
24.2 Todolist . . . . .	446
24.3 References . . . . .	447
<b>25 Unrecognized latex commands</b>	<b>449</b>
<b>26 Indices and tables</b>	<b>451</b>
<b>Bibliography</b>	<b>453</b>
<b>Python Module Index</b>	<b>455</b>
<b>Index</b>	<b>457</b>

**Version** 17726

**Date** August 03, 2012

**PDF** OfflineUserManual.pdf (via reStructuredText and Sphinx)

**Old PDF** main.pdf (direct from latex)



# INTRODUCTION

## 1.1 Intended Audience

This manual describes how Daya Bay collaborators can run offline 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 <sup>1</sup>. Some earlier software was used during the Daya Bay design stage and is documented elsewhere [g4dyb].

## 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 committing `.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
```

---

<sup>1</sup> See chapter *Offline Framework*.

To build the Sphinx derived latex and html renderings of the documentation some non-standard python packages must first be installed, as described [oum:docs](#). 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
```

## 1.5 Typographical Conventions

**This is bold text.**

# QUICK START

## 2.1 Offline Infrastructure

## 2.2 Installation and Working with the Source Code

### 2.2.1 Installing a Release

1. Download `dybinst` <sup>1</sup>.
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 [wiki:Category:Offline\\_Software\\_Releases](#) documents available 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 [wiki:Getting\\_Started\\_With\\_Offline\\_Software](#) 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.

---

<sup>1</sup> <http://dayabay.ihep.ac.cn/svn/dybsvn/installation/trunk/dybinst/dybinst>

## 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
```

## 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<sup>2</sup>. 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();
```

<sup>2</sup> <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 respectively. 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. [fig:evtdsp](#). 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. [fig:evtdsp](#) 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. [fig:evtdsp](#) 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
```

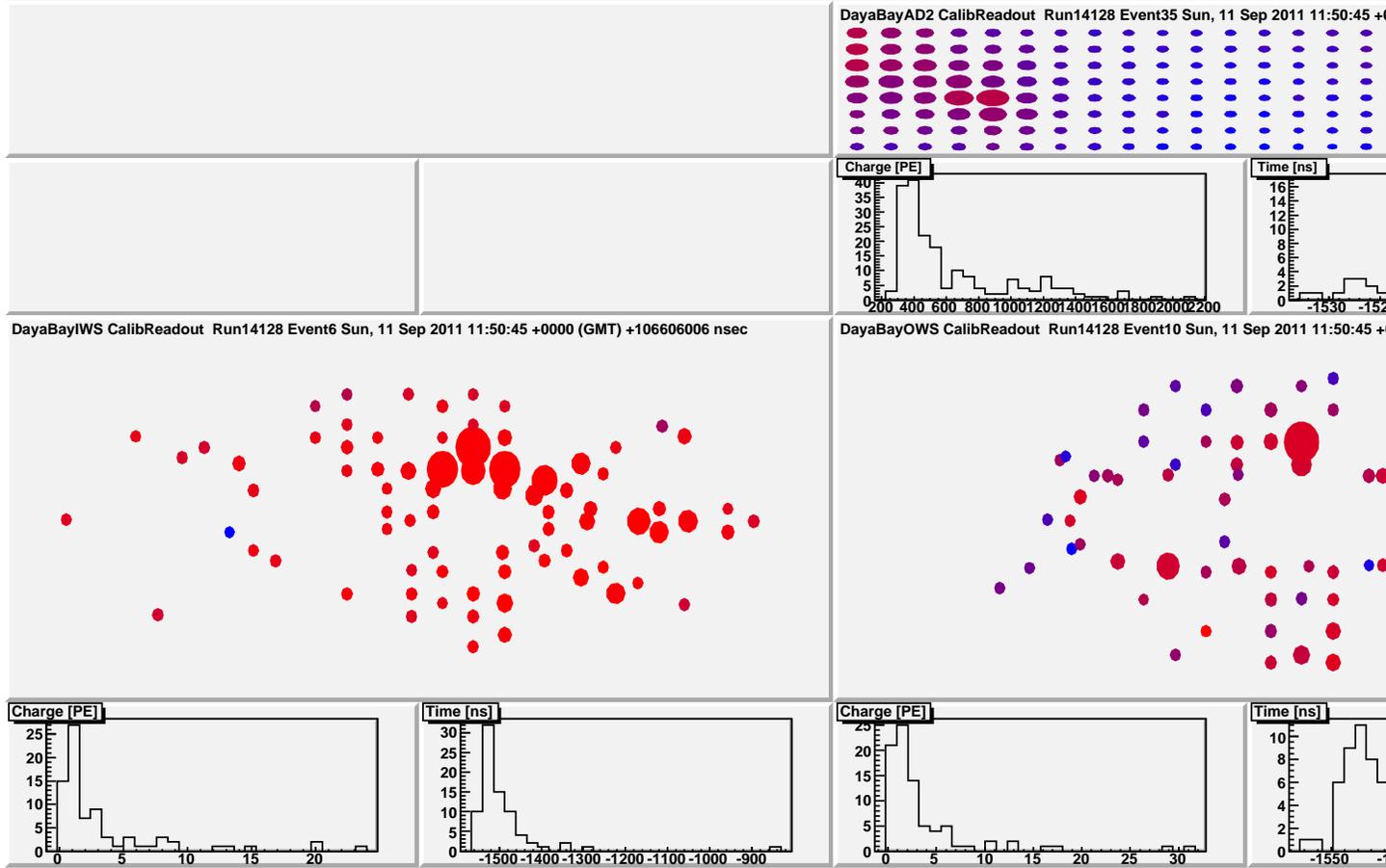


Figure 2.1: fig:evtdsp

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.

```
virtual StatusCode plotPool(DayaBay::Detector det,  
                             double chrg[9][24][2], double time[9][24][2],  
                             const char* chrgunit = 0, const char* timeunit = 0,  
                             const char* info = 0 ) =0;  
  
/// A pause method for user. After this all displayed stuff will be flushed.  
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 [oum:sop/](#)



# 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. *Opening data files*]
- Draw histograms of the data in the file [Sec. *Histogramming data*]
- Use NuWa to do more detailed calculations with the data [Sec. *NuWa Basics*]
- Write your own NuWa analysis module [Sec. *Change an Existing Job Module*]
- Write your own NuWa analysis algorithm [Sec. *Write a Python analysis Algorithm*]
- Select events using *tags* [Sec. *Tag Events in a NuWa File*]
- Add your own data variables to the data file [Sec. *Add Variables to a NuWa File*]
- Filter data based on data path or tag [Sec. *Copy Data Paths to a New File*]

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 *NuWa Basics* 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. *Loading the NuWa software*).

### 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. *fig:tesbrowser*. Event data is found under the path `/Event`, as summarized in Table *Standard paths for Event Data*. 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. *Data File Contents*.

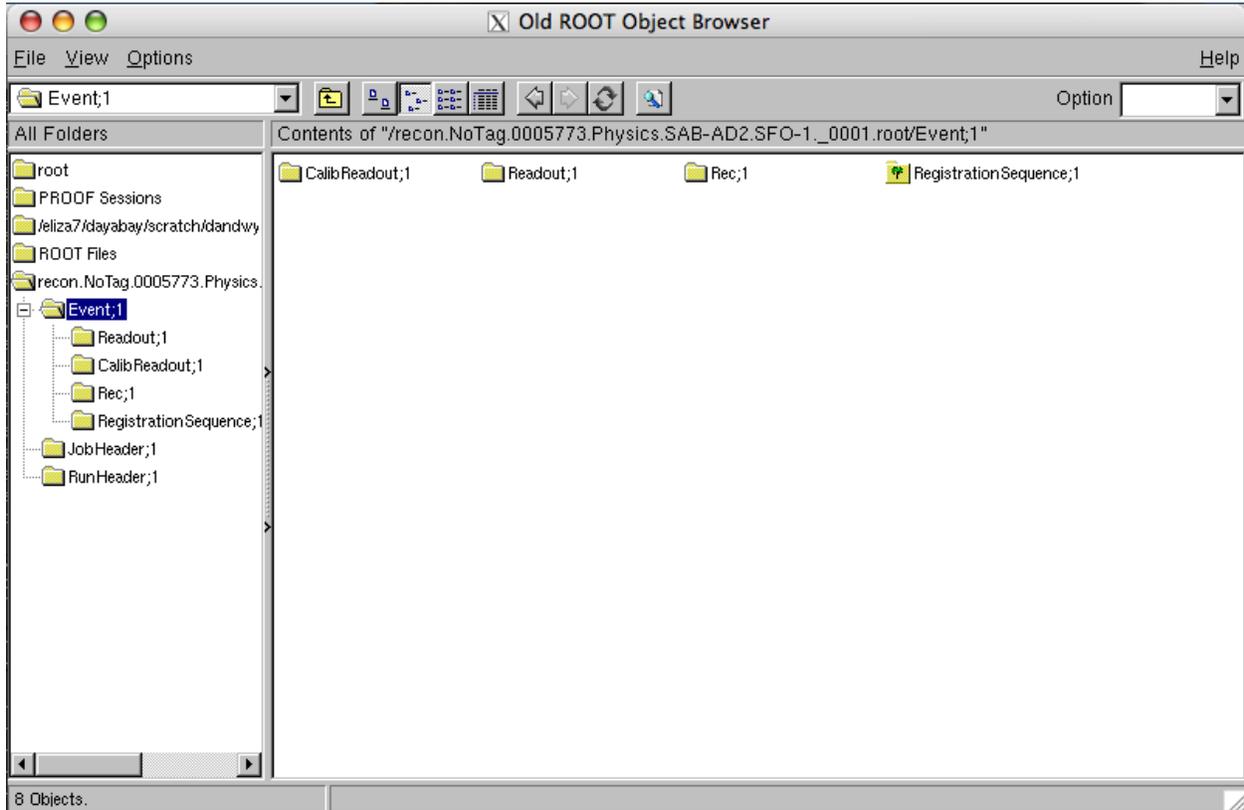


Figure 3.1: **fig:tesbrowser**  
Data File Contents

Table 3.1: Standard paths for Event Data

	<b>Real and Simulated Data</b>	
/Event/Readout	Raw data produced by the experiment	Sec. <i>Raw DAQ Data</i>
/Event/CalibReadout	Calibrated times and charges of PMT and RPC hits	Sec. <i>Calibrated Data</i>
/Event/Rec	Reconstructed vertex and track data	Sec. <i>Reconstructed Data</i>
	<b>Simulated Data Only</b>	
/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	

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 *Standard NuWa Event Data files*. The location of these files on each cluster are listed in Section *Standard Data Files*.

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

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");
```

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 [fig:reconbrowser](#) shows the data contained in a reconstructed event.

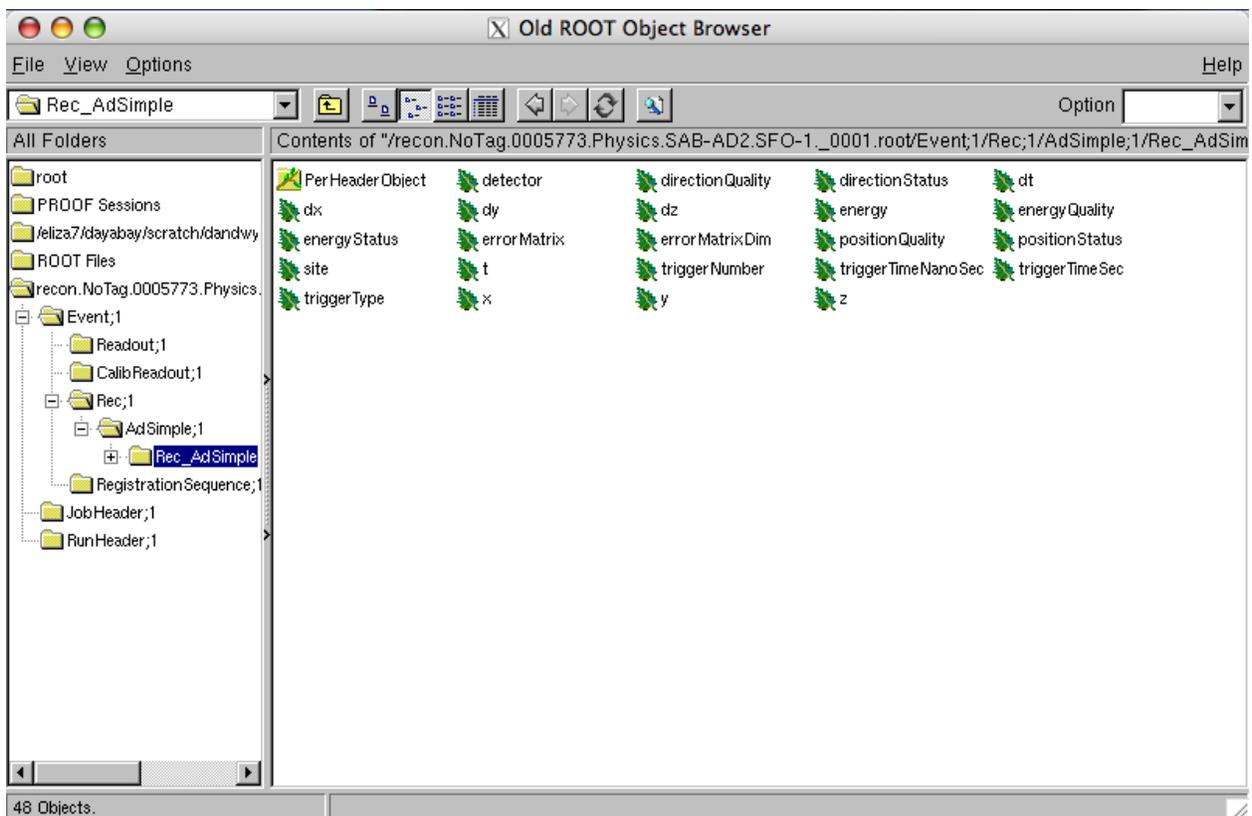


Figure 3.2: **fig:reconbrowser**  
Example Reconstructed Data

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. *fig:reconhists* shows the resulting histograms.

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

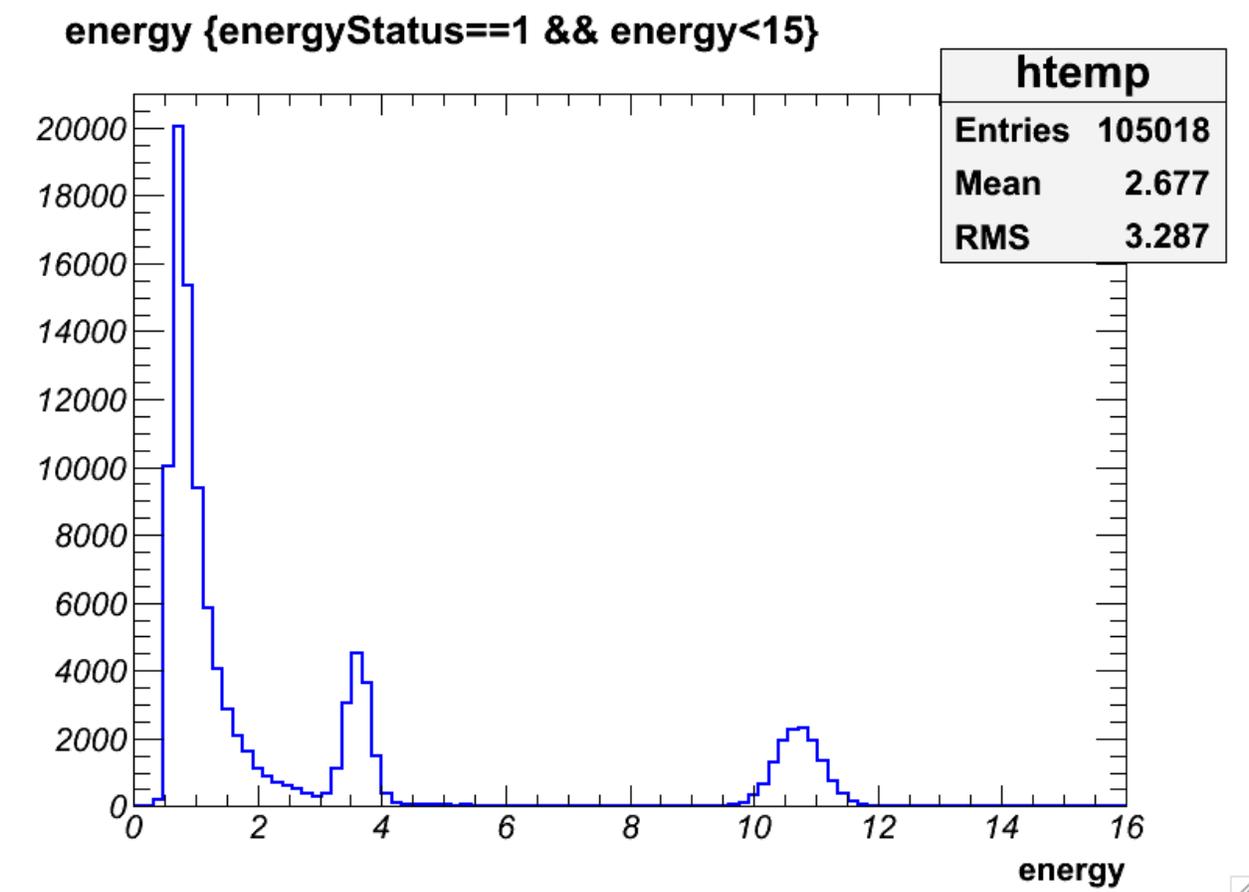


Figure 3.3: *fig:reconhists*

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. *figs:calibhists.*) The charge distribution for a specific 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")
```

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 *fig:chargevstimehist* shows a histogram of the calibrated PMT hit charges versus trigger time<sup>1</sup>. 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.

<sup>1</sup> The trigger time can be converted to a readable Beijing local time format using the lines described in Sec. *Time Axes in ROOT*

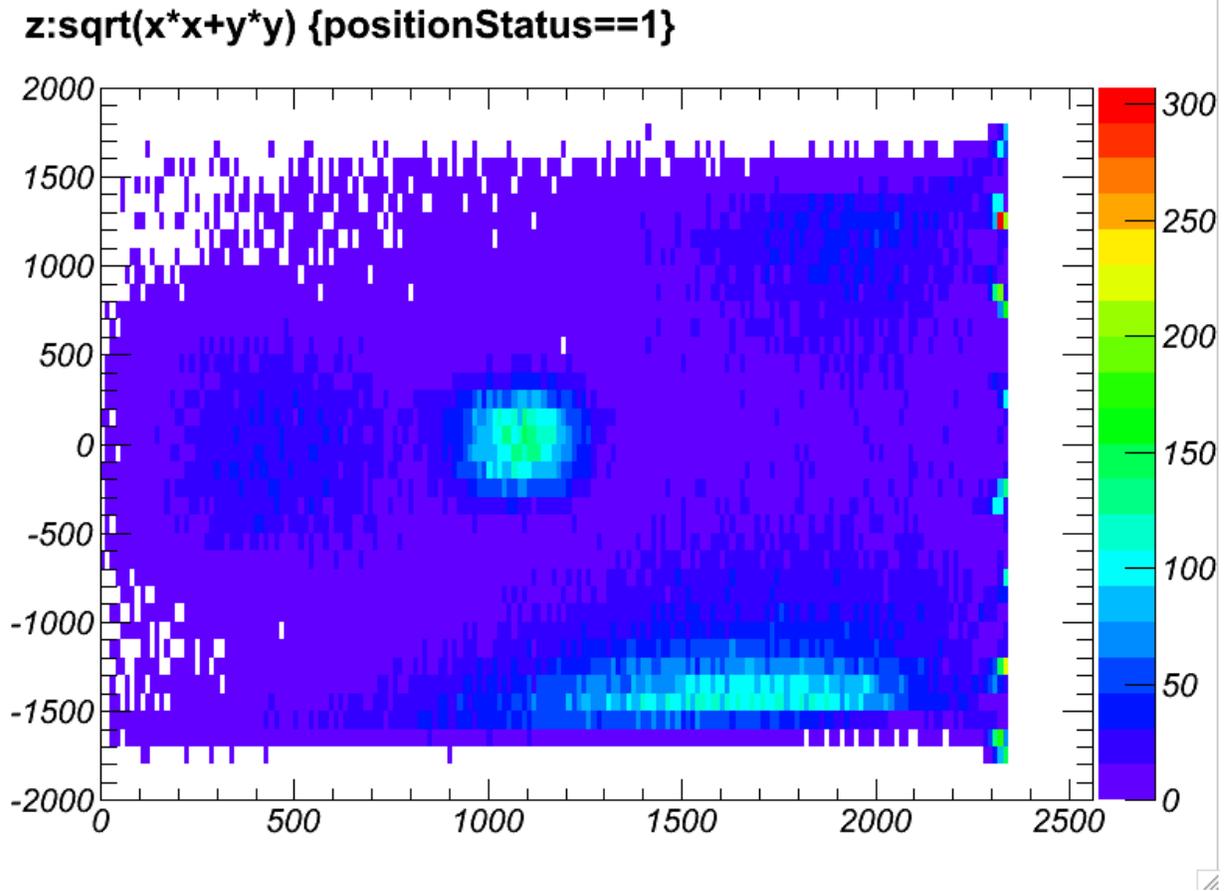


Figure 3.4: **fig:reconhists**  
Example Histograms

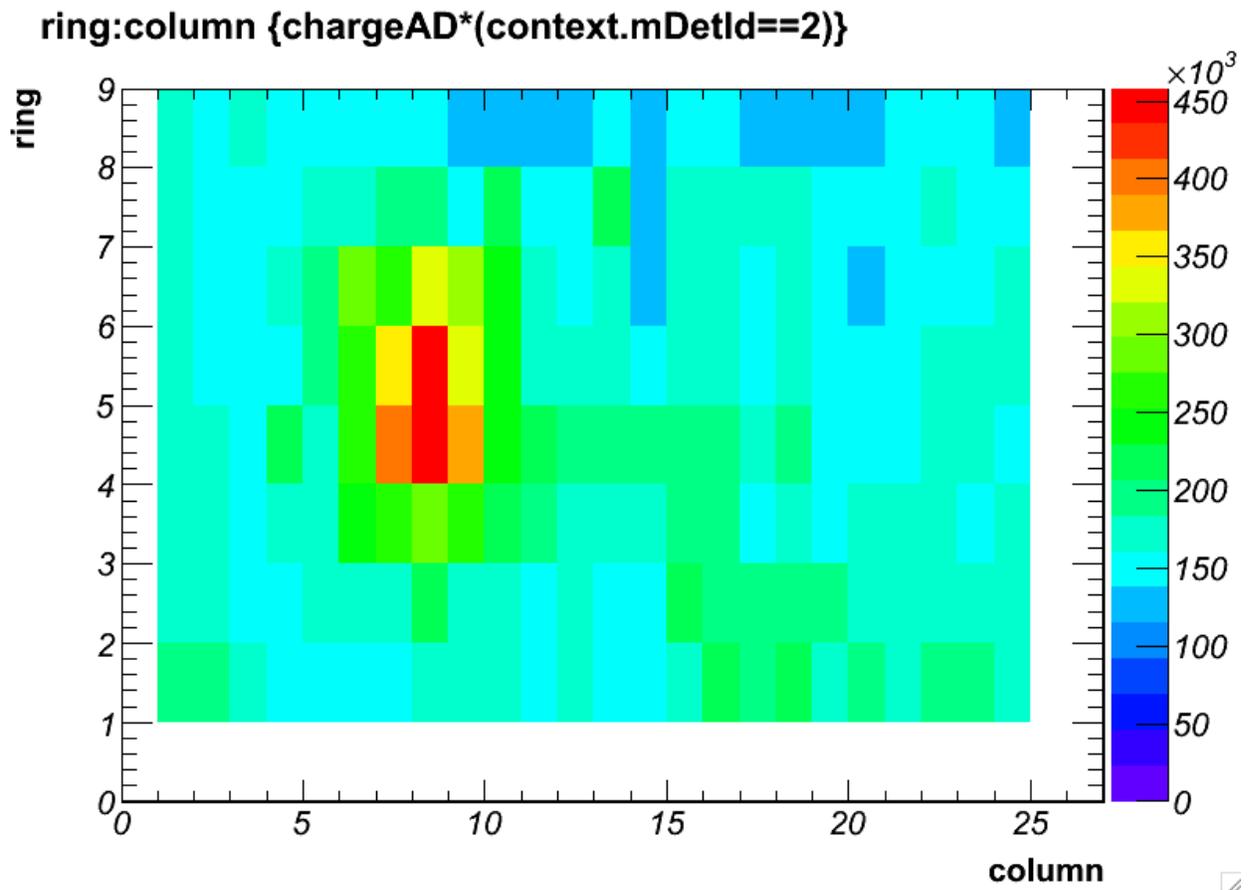


Figure 3.5: fig:calibhists

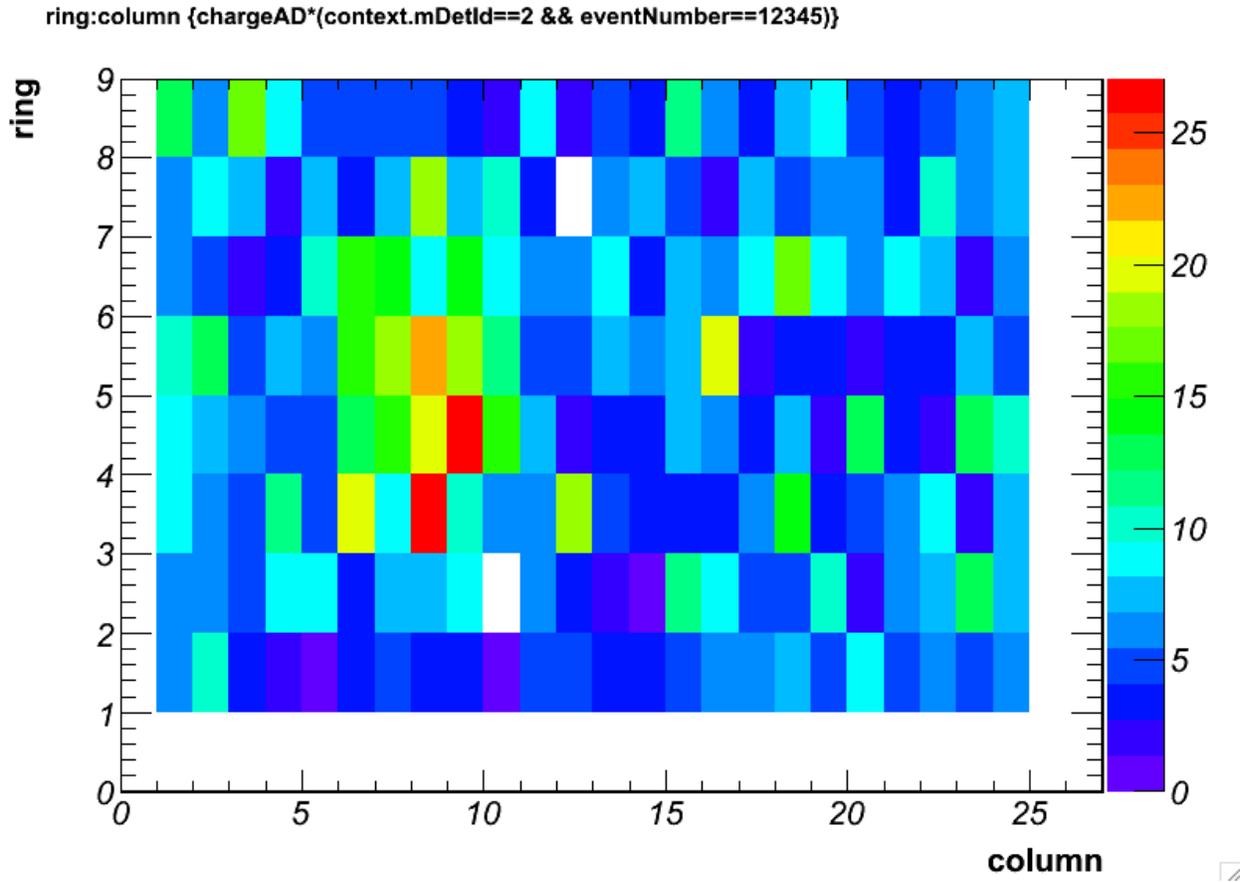


Figure 3.6: **fig:calibhists**  
 The calibrated PMT charge (in photoelectrons) for all events and for an individual event.

```

root[2] CalibReadoutHeader->Draw("chargeAD:triggerTimeSec+triggerTimeNanoSec*1e-9",
    "(detector==2 && ring==4 && column==15 && chargeAD>-3 && chargeAD<7)",
    "colz");
root[3] CalibReadoutHeader->Draw("Sum$(chargeAD):triggerTimeSec+triggerTimeNanoSec*1e-9",
    "detector==2 && Sum$(chargeAD)<1500", "colz");

```

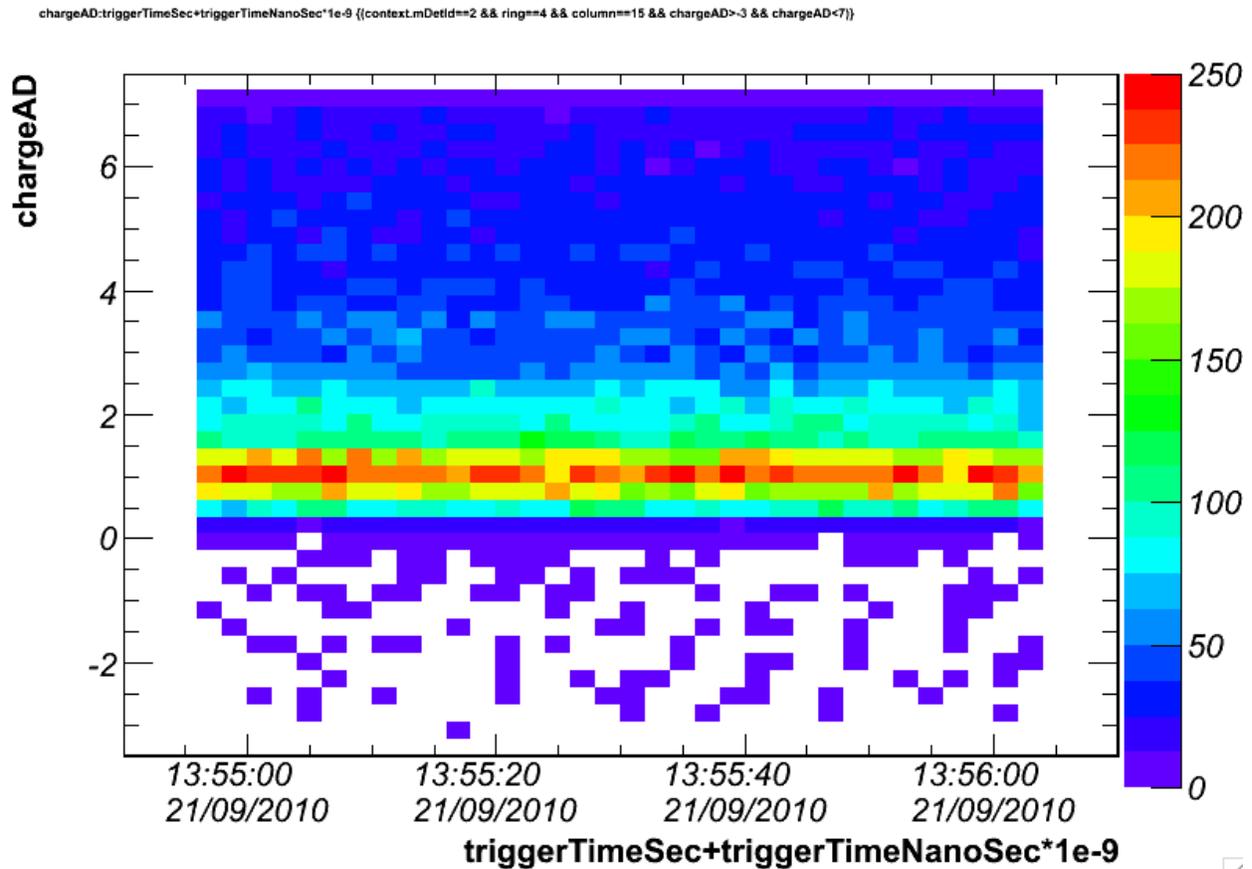


Figure 3.7: fig:chargevstimehist

### 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. [Loading the NuWa software](#). 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. [fig:rawhists.](#)) The selection on `context.mDetId==2` selects the detector AD2; Sec. [Conventions and Context](#) lists the allowed detector and site IDs. If you have a raw `.data` file produced by the DAQ, see section [Conversion from .data](#) 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");

```

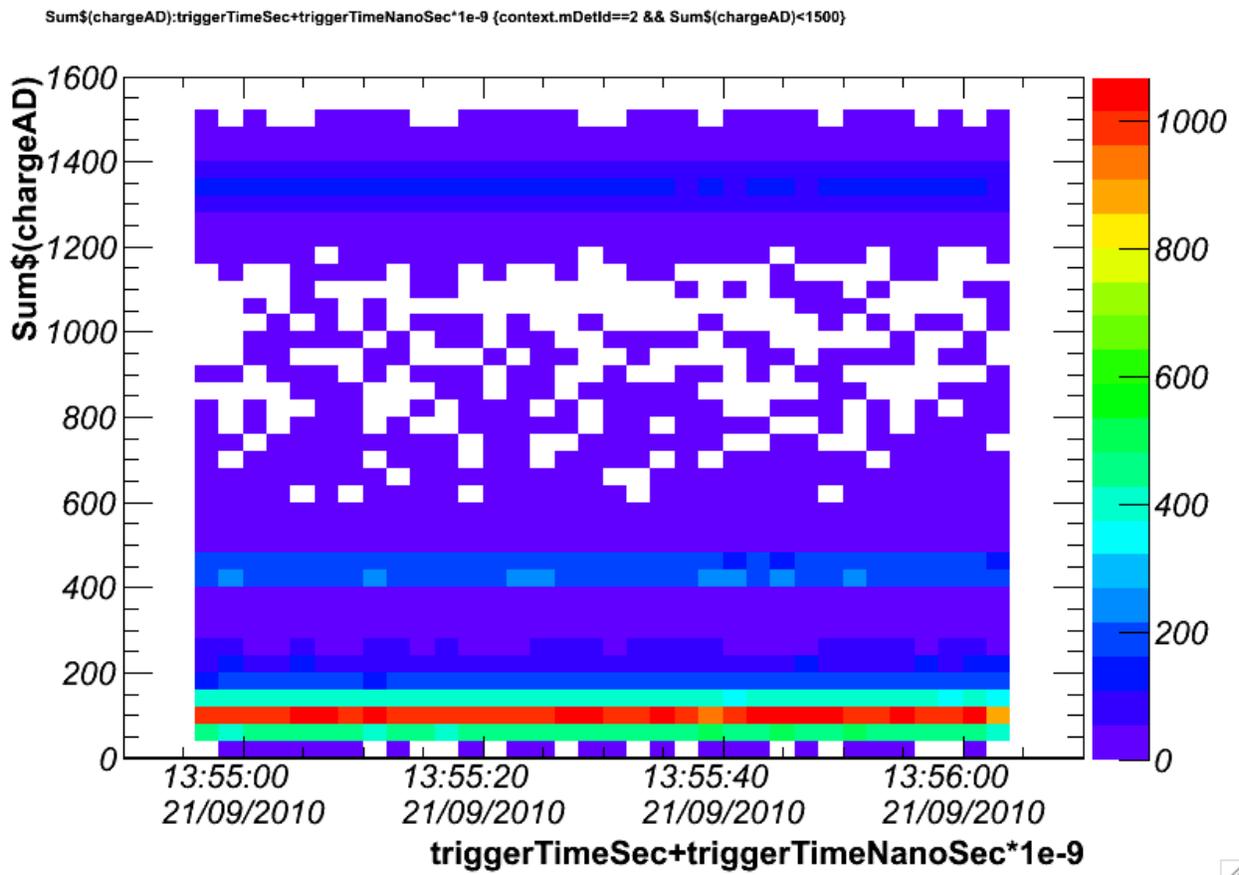


Figure 3.8: `fig:chargevstimehist`

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

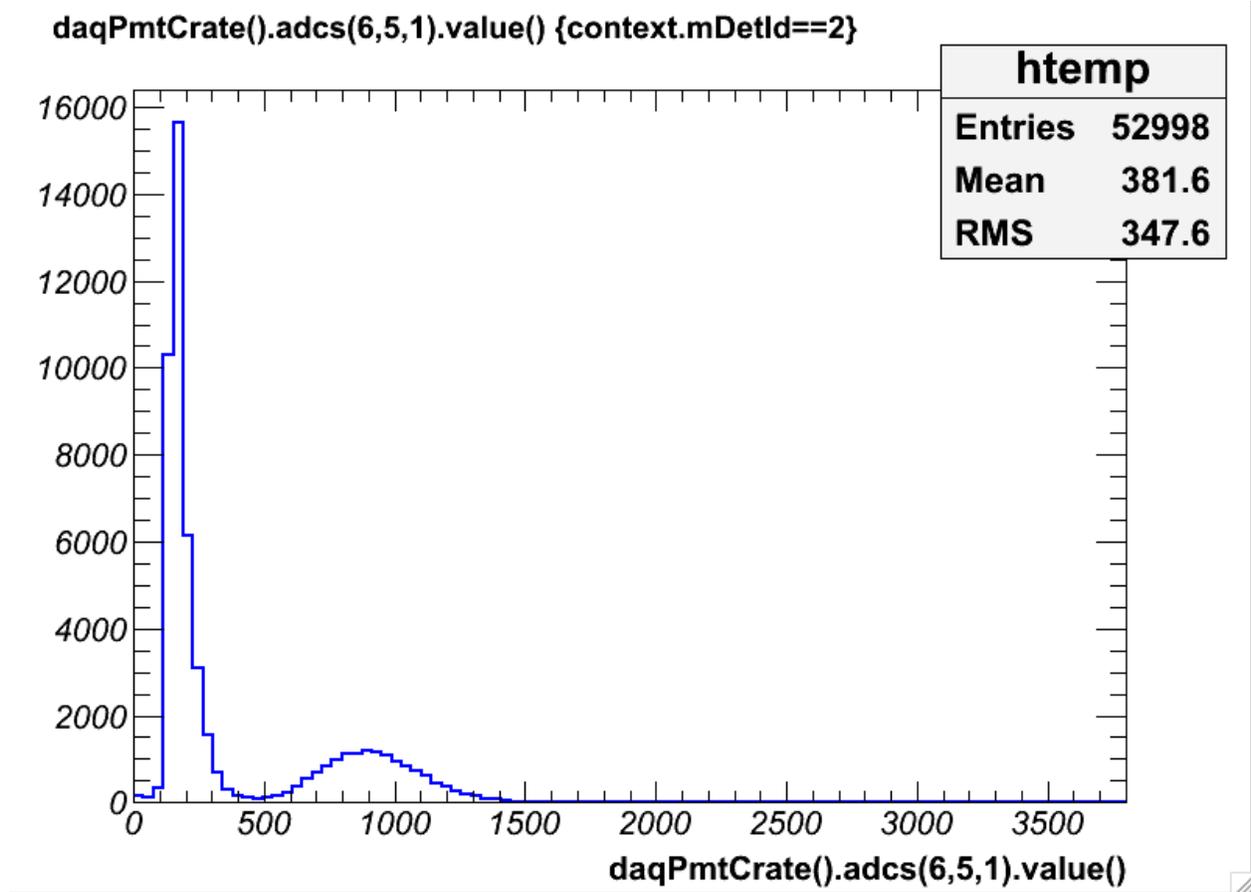
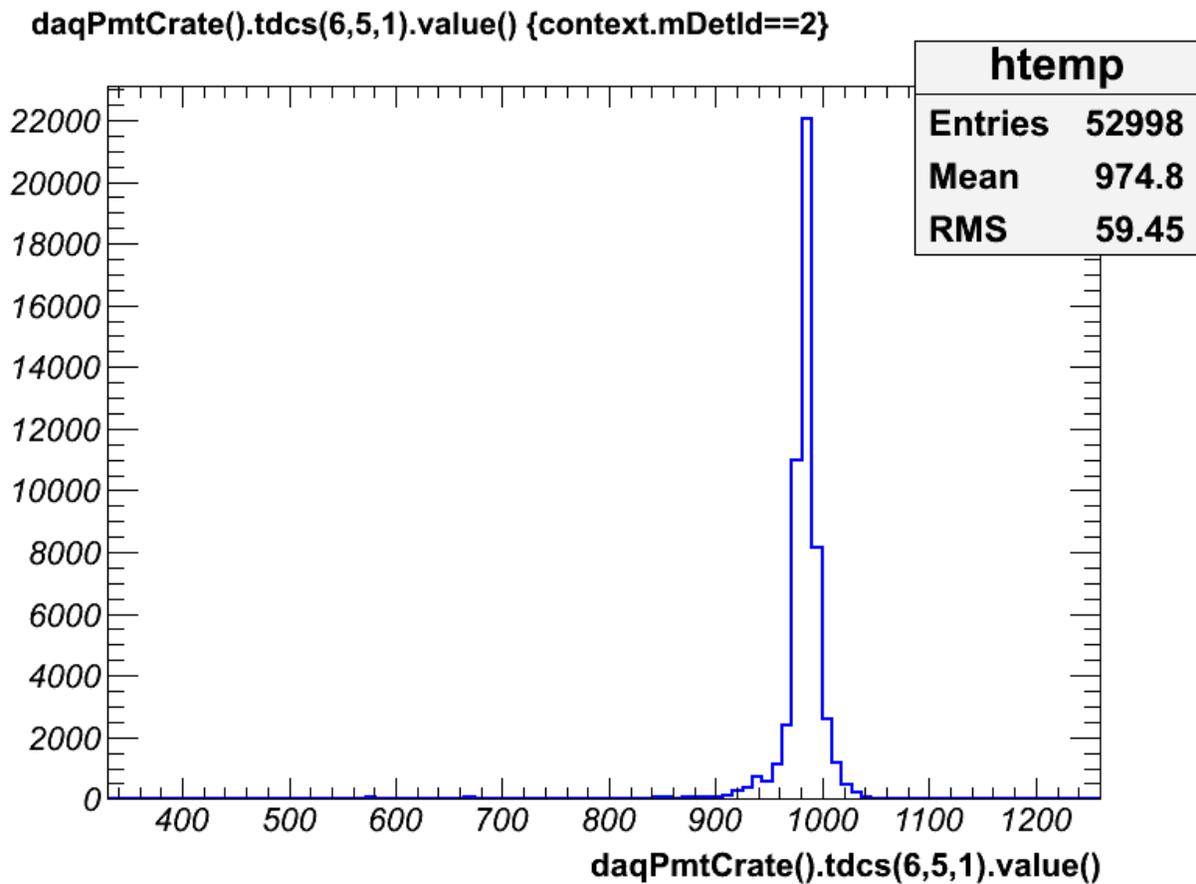


Figure 3.9: fig:rawhists

Figure 3.10: **fig:rawhists**

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 arbitrary 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 `dybgaudi:Tutorial/Quickstart` software package.

- `dybTreeDraw.C`: ROOT script using `TTree::Draw()`
- `dybTreeGetLeaf.C`: ROOT script using `TTree::GetLeaf()`
- `dybTreeSetBranch.C`: ROOT script using `TTree::SetBranchAddresses()`
- `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.

#### 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. In 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::SetBranchAddresses()` 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 reenale execNumber since the tree indexing requires it
reconT.SetBranchStatus("execNumber",kTRUE);
reconT.SetBranchStatus("calibStats.execNumber",kTRUE);

int detector = 0;
reconT.SetBranchStatus("context.mDetId",kTRUE);
reconT.SetBranchAddress("context.mDetId",&detector);

int energyStatus = 0;
```

```

reconT.SetBranchStatus("energyStatus",kTRUE);
reconT.SetBranchAddress("energyStatus",&energyStatus);

float energy = -1;
reconT.SetBranchStatus("energy",kTRUE);
reconT.SetBranchAddress("energy",&energy);

int nHit = -1;
reconT.SetBranchStatus("calibStats.nHit",kTRUE);
reconT.SetBranchAddress("calibStats.nHit",&nHit);

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

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

    // 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);
        }
    }
}

```

### 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:

```
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. *NuWa Basics*).

### 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 `dybgaudi:Tutorial/Quickstart` 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` 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){
// AD#2
chargeVsEnergyAD2H->Fill(energy,nominalCharge/energy);
}
}

} // End loop over AD triggers in the multiplet
} // 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:

```

// Create ‘friendly’ version of data from CalibStats
std::string mainEntriesName = "multiplicity";
std::vector<string> calibVarNames; //variable names to copy from CalibStats
calibVarNames.push_back("MaxQ");
calibVarNames.push_back("NominalCharge");
std::string indexMajorName = "triggerNumber";
std::string indexMinorName = "context.mDetId";
TTree* calibStatsFriendlyT = makeFriendTree(&adCoincT,
                                           &calibStatsT,
                                           mainEntriesName,
                                           calibVarNames,
                                           indexMajorName,
                                           indexMinorName);

if(!calibStatsFriendlyT){
std::cout << "Failed to create friendly tree" << std::endl;
return;
}

// Add new friendly tree to coincidence tree
adCoincT.AddFriend(calibStatsFriendlyT,"calibStats");

```

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

```

// Fill histograms
// AD#1
adCoincT.Draw("calibStats.NominalCharge/e:e>>chargeVsEnergyAD1H",
              "context.mDetId==1 && energyStatus==1 && e>0","colz");

// AD#2

```

```
adCoincT.Draw("calibStats.NominalCharge/e:e>>chargeVsEnergyAD2H",
             "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:

```
TFile outputFile("friendlyCalibStats.root", "RECREATE");
calibStatsFriendlyT.SetDirectory(&outputFile);
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. *Loading the NuWa software*. To install NuWa on another computer, see Sec. *Installing the NuWa software*.

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 *sec:nuwaoptions*. 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. *NuWa Job Modules* 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. *See the history of a NuWa File*]
- Tag a set of events in a NuWa file [Sec. *Tag Events in a NuWa File*]
- Add your own variables to the NuWa file [Sec. *Add Variables to a NuWa File*]
- Copy all the data at a path to a new file [Sec. *Copy Data Paths to a New File*]
- Write tagged data to a new file [Sec. *Write Tagged Data to a New File*]
- Change the configuration of an existing Job Module [Sec. *Change an Existing Job Module*]
- Write your own analysis Algorithm [Python] [Sec. *Write a Python analysis Algorithm*]
- Write your own analysis Algorithm [C++] [Sec. *Write a C++ analysis Algorithm*]
- Modify an existing part of NuWa [C++] [Sec. *Modify Part of NuWa*]

### 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
}
```

**Cached Job Information:**

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

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

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

To add your own tag, follow the steps for modifying an existing python module (section *Write a Python analysis Algorithm*.) Use `dybgaudi:Tagging/UserTagging/python/UserTagging/UserTag/DetectorTag.py` as a starting point. You should add your own tag in the `initTagList` function:

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

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

```
# Get reconstructed data
recHdr = evt["/Event/Rec/AdSimple"]
# Add your calculation / decision here
# ...
#
if tagThisEvent:
    # Keep track of the reconstructed data you are tagging
    self.getTag('MySpecialEvent').setInputHeaders( [recHdr] )
    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:

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

Tags can also be used to produce filtered data sets, as shown in section *Write Tagged Data to a New File*.

### 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 `dybgaudi:Tutorial/Quickstart/python/Quickstart/DtData.py` 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 *fig:userdata*.) The file can also be read back into another NuWa job, and the user data will still be accessible.

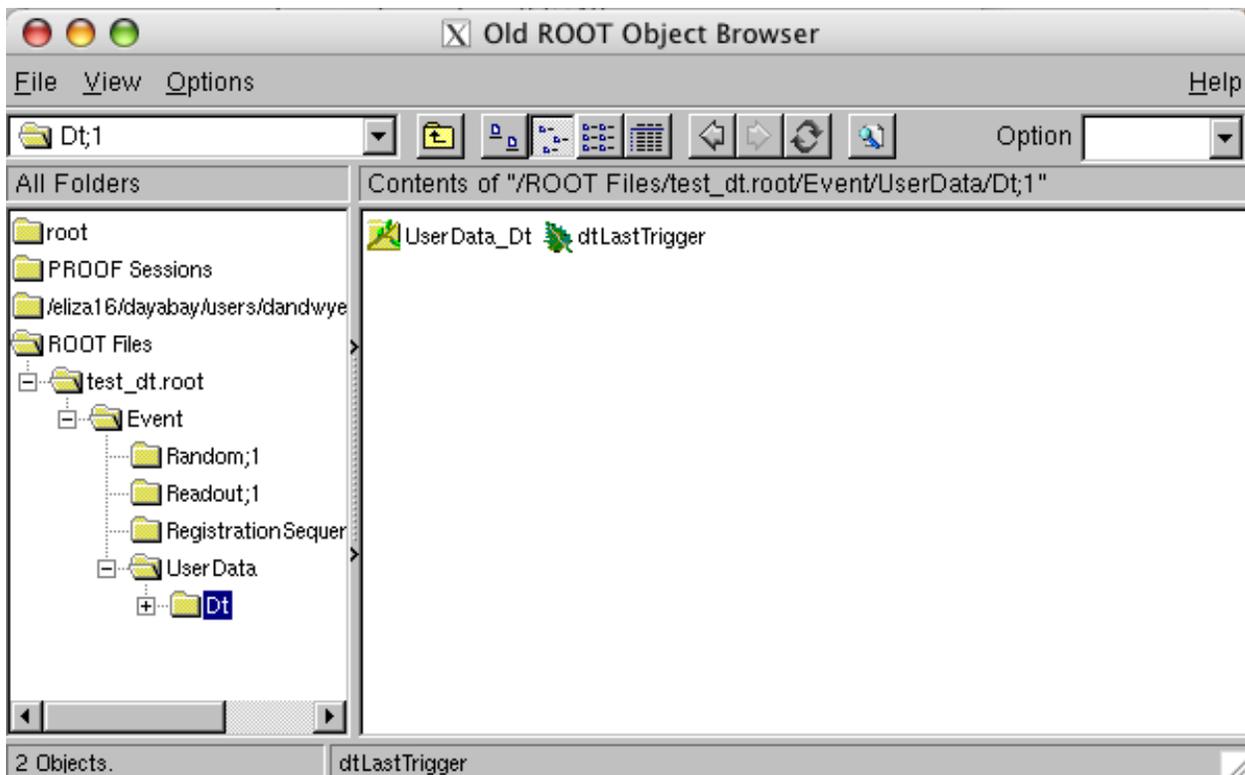


Figure 3.11: *fig:userdata*

To add your own variables, copy and modify the module `dybgaudi:Tutorial/Quickstart/python/Quickstart/DtData.py`. See section *Write a Python analysis Algorithm* 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.

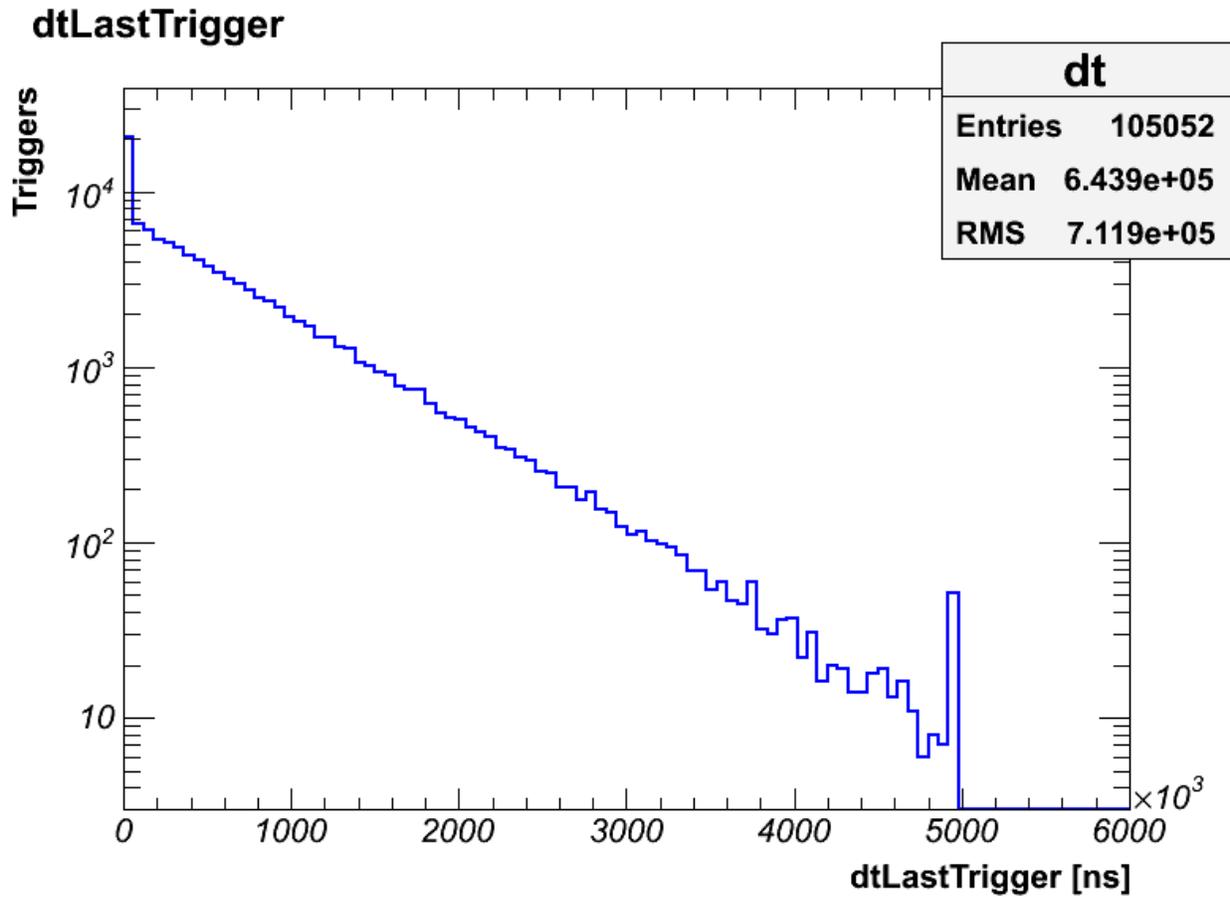


Figure 3.12: **fig:userdata**  
 Example of browsing and histogramming user-defined data in ROOT.

### 3.4.4 Adding User-defined Variables to Tagged Events

The `dybgaudi:Tagging/UserTagging` package provides some convenient tools for simultaneously applying tags and adding user data for those tagged events. Following the example described in section *Tag Events in a NuWa File*, 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')
    myData.set('myInt', 12345)
    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:

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

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

```
shell> nuwa.py -n -1 -m"SimpleFilter.Keep /Event/Rec/AdSimple /Event/Rec/AdQmlf"
-o myRecData.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
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 *Tag Events in a NuWa File*. The following example shows how to create an output file which contains only the events you have tagged as `MySpecialEvents`:

```
shell> nuwa.py -n -1 -m"MySpecialTagger" -m"SmartFilter.Keep /Event/UserTag/MySpecialEvents"
-o mySpecialEvents.NoTag.0005773.Physics.SAB-AD2.SFO-1._0001.root
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 `dybgaudi:Filtering/SmartFilter/python/SmartFilter/Example.py`.

### 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`,

```
shell> mkdir mywork
shell> cd mywork
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()`,

```
def configure( argv=[] ):
    """A description of your module here
    """
    # Most job configuration commands here
    return

def run(app) :
    """Specific run-time configuration"""
    # 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 [[Readout data in NuWa](#)], Calibrated hit data [[Calibrated data in NuWa](#)], Reconstructed data [[Reconstructed data in NuWa](#)].

Remember to commit your new algorithm to SVN! The wiki section [wiki:SVN\\_Repository#Guidelines](#) 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 [Making your own Project](#) 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. Starting 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 *Making your own Project*.

Next, you should checkout the package into your Project:

```
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 *Some Common Services* gives lists some common services. Section *NuWa Services* gives detailed descriptions of the common services.

Table 3.3: Some Common Services

<b>ICableSvc</b>	<b>Electronics cable connection maps and hardware serial numbers</b>
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

Multiple versions of the same service can exist. 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" << endl;
    return StatusCode::FAILURE;
}
```

## 3.5 Cheat Sheets

- Loading the NuWa software
- Installing the NuWa software
- Making your own Project
- Standard Data Files
  - Using the Catalog
- Data File Contents
- Common NuWa Commands
- Conventions and Context
  - Sites
  - Detectors
- Raw DAQ Data
  - Conversion from `.data`
  - Raw data in ROOT
  - Readout data in NuWa
- Calibrated Data
  - Calibrated data in ROOT
  - Calibrated data in NuWa
- Calibrated Statistics Data
  - Calibrated statistics data in ROOT
  - Calibrated statistics data in NuWa
- Reconstructed Data
  - Reconstructed data in ROOT
  - Reconstructed data in NuWa
- Spallation Data
  - Spallation data in ROOT
  - Spallation data in NuWa
- Coincidence Data
  - Coincidence data in ROOT
  - Coincidence data in NuWa
- NuWa Services
- Computer Clusters
- Miscellaneous
  - Time Axes in ROOT

### 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](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](mailto:theta13-offline@dayabay.lbl.gov). Your chance of success will be much greater if you 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 *Write a C++ analysis Algorithm* for more details.

You can also replace an existing NuWa package with you own modified version in `myNuWa/`. See section *Modify Part of NuWa* 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
	<b>Onsite Farm</b>
daq. (.data)	/dyb/spade/rawdata
daq.	??
	<b>PDSF</b>
	(In HPSS Archive)
daq. (.data)	/eliza16/dayabay/nuwaData/exp,sim/dataTag/daq
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
	<b>IHEP</b>
daq. (.data)	
daq.	
recon.	
coinc.	
spall.	
	<b>BNL</b>
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 *Loading the NuWa software*). 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']
>>> DybPython.Catalog.runs[8001]
['/eliza16/dayabay/data/exp/dayabay/2011/TestDAQ/NoTag/0430/daq.NoTag.0008001.Physics.EH1-Merged.SFO-1']
>>> DybPython.Catalog.runs[8002]
['/eliza16/dayabay/data/exp/dayabay/2011/TestDAQ/NoTag/0430/daq.NoTag.0008002.Pedestal.EH1-WPI.SFO-1']
```

For more information, refer to the Catalog description [wiki:https://wiki.bnl.gov/dayabay/index.php?title=Accessing\\_Data\\_in\\_a\\_Warehouse](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
	<b>Real and Simulated Data</b>	
/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
	<b>Simulated Data Only</b>	
/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 [Loading the NuWa software](#)).

```
# Wrap raw DAQ files in ROOT tree:
```

```
shell> nuwa.py -n -l -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 -l -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 -l -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 -l -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 -l -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 -l --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 `dybgaudi:DataModel/Conventions/Conventions/Site.h`.

#### Detectors

The detector ID identifies the detector location within the site.

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 `dybgaudi:DataModel/Conventions/Conventions/DetectorId.h`.

### 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 *Histogramming Raw DAQ data*. 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 -l -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	triggerType()	All active triggers, logically OR'd
type		
trigger	triggerTime().GetSeconds()	Complete trigger time [seconds]
time		
TDC time	tdcs(board,*connector*,*adcGain*).values()	Channel TDC values
ADC	adcs(board,*connector*,*adcGain*).values()	Channel ADC values
charge		
	gains(board,*connector*).values()	Channel ADC Gain (1: Fine ADC, 2: Coarse ADC)
	preAd- cRaws(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

### 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  $\approx$  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
    #  $\sim$ 70 ADC counts (ie. no signal =  $\sim$ 70 ADC, 1 photoelectron
    # =  $\sim$ 90 ADC, 2 p.e. =  $\sim$ 110 ADC, etc.)
    #
    # The ADC peak cycle is a record of the clock cycle which had
    # the 'peak' ADC.
    #
    # ADC Gain: Here is a description of ADC gain for these values
    # Unknown = 0
    # High = 1
    # Low = 2
    #
    adc = channel.adc( hitIdx )
    preAdc = channel.preAdcAvg( hitIdx )
    peakCycle = channel.peakCycle( hitIdx )
    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 possible hits)
time_PSD	For hits in each AD, for time window between -1650 and -1250 ns, $\frac{N_{hit_{-1650,-1450}}}{N_{hit_{-1650,-1250}}}$ .
time_PSD1	For hits in each AD, for time window between -1650 and -1250 ns, $\frac{N_{hit_{-1650,-1500}}}{N_{hit_{-1650,-1250}}}$ .
time_PSD_local_RMS	The RMS of the time of the first hit (also must be within -1650 and -1250) for 5x5 (or 4x5 for 4 columns) PMTs
Q1	The total charge (within -1650 and -1250) of nearby $\pm 3$ columns PMTs (total 7 columns)
Q2	The total charge (within -1650 and -1250) of 4 $\rightarrow$ 9 and -4 $\rightarrow$ -9 columns PMTs (total 12 columns)
Q3	The total charge (within -1650 and -1250) of PMTs for the rest of columns (other than those in Q1 and Q2)
flasher_flag	"1-time_PSD + 1- time_PSD1 + Q3/Q2*2 + nPEMax/nPESum + time_PSD_local_RMS/100"
EarlyCharge	The charge sum in time window $t < -1650$ ns
LateCharge	The charge sum in time window $t > -1250$ ns
NominalCharge	The charge sum in time window $-1650$ ns $< t < -1250$ ns, 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 value in the ring
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. MainPeak cluster
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 cluster, see Doc6926
ColumnKurtosis	Kurtosis of charge weighted distance in the Column dimension for the MainPeak cluster
Kurtosis	Sum of RingKurtosis and ColumnKurtosis
MiddleTimeRMS	RMS of PMT first hit time in the time window (-1650ns, -1250ns). This time window should be centered on the trigger
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 Livetime' is the sum of 'DAQ Running time' and 'DAQ Livetime'.
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 the electronics memory buffers filled immediately preceding the current trigger.
buffer_full_flag	This flag is true if the electronics memory buffers filled immediately preceding this trigger. I

### 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]
position	energyQuality	Measure of fit quality ( $\chi^2$ , likelihood, etc.)
	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
direction	positionQuality	Measure of fit quality ( $\chi^2$ , likelihood, etc.)
	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
error matrix	directionQuality	Measure of fit quality ( $\chi^2$ , likelihood, etc.)
	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")
    return FAILURE

result = reconHdr.recTrigger()

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

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

# Trigger Time: Absolute time of trigger for this raw data

```

```

triggerTime = result.triggerTime()

# Energy information
result.energyStatus()
result.energy()
result.energyQuality()

# Position information
result.positionStatus()
result.position().x()
result.position().y()
result.position().z()
result.positionQuality()

# Direction information, for tracks
result.directionStatus()
result.direction().x()
result.direction().y()
result.direction().z()
result.directionQuality()

# Covariance Matrix, if one is generated
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)

Continued on next page

Table 3.5 – continued from previous page

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.

```

evt = self.evtSvc()

# Access the Coincidence Data Header.
# This is a container for AD coincidence data
coincHdr = evt["/Event/Data/Coinc/AD1Coinc"]
if coincHdr == None:
    self.debug("No coincidence header this cycle")
    return SUCCESS

# Access the Coincidence Data
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.

```

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

# 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



# 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”<sup>1</sup>. 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
```

---

<sup>1</sup> [https://wiki.bnl.gov/dayabay/index.php?title=Cluster\\_Account\\_Setup](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<sup>rd</sup> party binary libraries and header files under PACKAGE/VERSION/ sub directories.

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

**lcgcm** build information for using 3<sup>rd</sup> 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](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](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](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](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`<sup>2</sup> 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](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 `lccgcm/CG_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.

---

<sup>2</sup> <http://git.or.cz/>



# 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<sup>1</sup>. This nick-name lets you

---

<sup>1</sup> Nick-names default to the class name.

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:

**`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

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. *fig:config-layers*. 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 :
  GenName: Name of this generator for book keeping purposes.
  GenTools: Tools to generate HepMC::GenEvents
  GlobalTimeOffset: None
  Location: TES path location for the HeaderObject this algorithm produces.
  ...
```

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:

```
from Gaudi.Configuration import ApplicationMgr
theApp = ApplicationMgr()

from MyPackage.MyPackageConf import MyAlgorithm
ma = MyAlgorithm()
ma.SomeProperty = "harder, faster, stronger"
theApp.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:

```
# gets name 'MyAlgorithm'
generic = MyAlgorithm()
# gets name 'alg1'
```

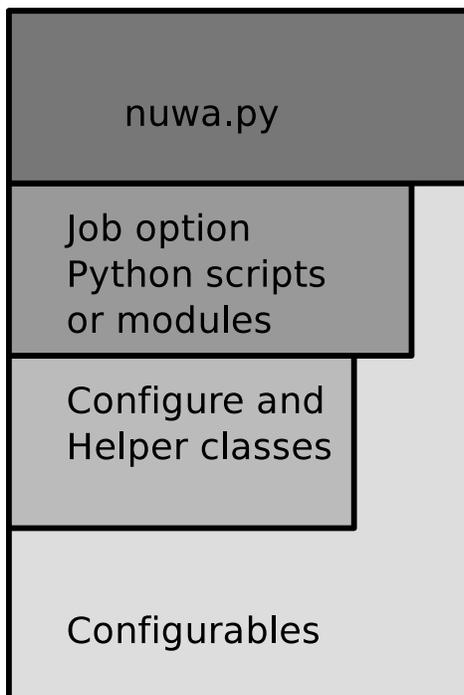


Figure 6.1: **fig:config-layers**  
Cartoon of the layers of configuration code.

```
specific = MyAlgorithm('alg1')

theApp.TopAlg.append(generic)
theApp.TopAlg.append(specific)
# 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 has 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:

```
from GenTools.Helpers import Gun
gunner = Gun()

import GaudiKernel.SystemOfUnits as units
gunner.timerator.LifeTime = int(60*units.second)
# ...
import GenTools
gt = GenTools.Configure("gun", "Particle Gun", helper=gunner)
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:

```
shell> pydoc GenTools.Helpers
Help on module GenTools.Helpers in GenTools:

NAME
    GenTools.Helpers

FILE
    /path/to/NuWa-trunk/dybgaudi/InstallArea/python/GenTools/Helpers.py

DESCRIPTION
    Several helper classes to assist in configuring GenTools. They
    assume geometry has already been setup. The helper classes that
    produce tools need to define a "tools()" method that returns an
    ordered list of what tools it created. Users of these helper classes
    should use them like:

CLASSES
    Gun
    HepEVT
    ...
```

```
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__(self, ...)
|       Construct the configuration. Coustom configured tools can
|       be passed in or customization can be done after construction
|       using the data members:
|
|       .gun
|       .positioner
|       .timerator
|       .transformer
|
|       The GtGenerator alg is available from the .generatorAlg member.
|
|       They can be accessed for additional, direct configuration.
|
|   ...
```

## 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 `imported` 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 `imported`. 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 <sup>2</sup> 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 <sup>3</sup> 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
- 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 <sup>4</sup>:

```
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 ...] \
```

<sup>2</sup> Code is at `dybgaudi/Simulation/GenTools/python/GenTools/Test.py`.

<sup>3</sup> Code is at `tutorial/DivingIn/python/DivingIn/Example.py`

<sup>4</sup> Actual output may differ slightly.

```
[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 specifically 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
                    Load given module and pass optional argument list
-n EXECUTIONS, --executions=EXECUTIONS
                    Number of times to execute list of top level
                    algorithms.
-o OUTPUT, --output=OUTPUT
                    Output filename
-O OUTPUT_STREAMS, --output-streams=OUTPUT_STREAMS
                    Output file map
-I INPUT_STREAMS, --input-streams=INPUT_STREAMS
                    Input file map
-H HOSTID, --hostid=HOSTID
                    Force given hostid
-R RUN, --run=RUN   Set run number
-N EXECUTION, --execution=EXECUTION
                    Set the starting execution number
-V, --visualize    Run in visualize mode
-G DETECTOR, --detector=DETECTOR
                    Specify a non-default, top-level geometry file
```

Each job option `.py` file that you pass on the command line will be evaluated 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:

```
shell> nuwa.py -n1 -m "DybPython.TestMod1 -a foo bar" \  
             -m DybPython.TestMod2 \  
             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 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()
```



# 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<sup>1</sup>. 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  $\mu\text{sec}$  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 Storeaes` 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`.

---

<sup>1</sup> Before 2038 someone had better increase the size what stores the seconds!

**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.

**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 should 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<sup>2</sup>.

---

<sup>2</sup> <http://dayabay.ihep.ac.cn/cgi-bin/DocDB/ShowDocument?docid=3131>



# 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. <sup>1</sup>

---

<sup>1</sup> 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`<sup>2</sup> 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 *The nuwa.py main script* 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.

---

<sup>2</sup> **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<sup>3</sup>. 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/ ↔ 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.

<sup>3</sup> This is actually done in `RootIO/Test/DybStorageAlg`

## 8.6.2 Guidelines for Writing Persistent Data Classes

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

- 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 input 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:

```
#include "RootIOSvc/RootIOTypedCnv.h"
#include "PerGenEvent/PerGenHeader.h"
#include "Event/GenHeader.h"

class GenHeaderCnv : public RootIOTypedCnv<PerGenHeader,
                    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:

```
public:
    static const CLID& classID() {
        return DayaBay::CLID_GenHeader;
    }

    GenHeaderCnv(ISvcLocator* svc);
    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.

```
StatusCodes 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;

    if (m_perSimHeader) return StatusCode::SUCCESS;

    RootIOBaseCnv* other = this->otherConverter(SimHeader::classID());
    if (!other) return StatusCode::FAILURE;

    const RootIOBaseObject* base = other->getBaseObject();
    if (!base) return StatusCode::FAILURE;

    const PerSimHeader* pgh = dynamic_cast<const PerSimHeader*>(base);
    if (!pgh) return StatusCode::FAILURE;

    m_perSimHeader = pgh;

    return StatusCode::SUCCESS;
}
```

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`:

```
package PerGenEvent
version v0

use Context      v*      DataModel
use BaseEvent    v*      DataModel
use GenEvent     v*      DataModel
use ROOT         v*      LCG_Interfaces
use CLHEP        v*      LCG_Interfaces
```

```
use PerBaseEvent v*      RootIO

# public code
include_dirs $(PERGENEVENTROOT)
apply_pattern install_more_includes more="PerGenEvent"
library PerGenEventLib lib/*.cc
apply_pattern linker_library library=PerGenEventLib

# component code
library PerGenEvent components/*.cc
apply_pattern component_library library=PerGenEvent

# dictionary for persistent classes
apply_pattern reflex_dictionary dictionary=PerGenEvent \
    headerfiles=$(PERGENEVENTROOT)/dict/headers.h \
    selectionfile=./dict/classes.xml
```



# 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 hierarchy, 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 *fig:log-phy-touch* 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 <sup>1</sup>.

### 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 hierarchy 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 <sup>2</sup> and continuing counter clockwise if looking down at the AD. The sequential ID number can be calculated by:

column# + 24\*(ring# - 1)

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

---

<sup>1</sup> 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.

<sup>2</sup> Here the X-axis points to the exit of the hall.

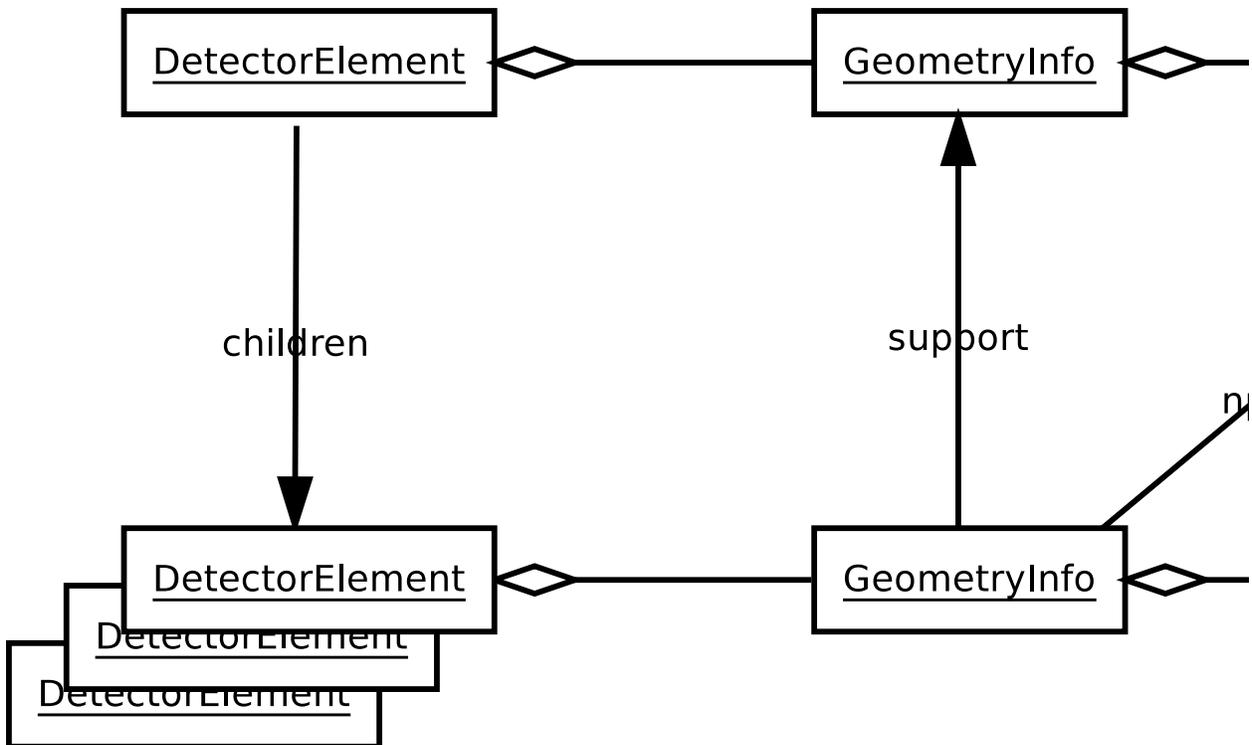


Figure 9.1: **fig:log-phy-touch**  
 Logical, Physical and Touchable volumes.

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** The 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 Steel 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:

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

Gaudi::XYZPoint globalPoint = ...;
IDetectorElement* de = icss->coordSysDE(globalPoint);
if (!de) handle_error();
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.

```
// if in a GaudiAlgorithm can use getDet():
DetectorElement* de = getDet<DetectorElement>("/dd/Structure/DayaBay");
LVVolume* lv = getDet<LVVolume>("/dd/Geometry/AD/lvOIL");

// or if not in a GaudiAlgorithm do it more directly:
IDataProviderSvc* detSvc = 0;
StatusCode sc = service("DetectorDataSvc", detSvc, true);
if (sc.isFailure()) handle_error();

SmartDataPtr<IDetectorElement> topDE(detSvc, "/dd/Structure/DayaBay");
if (!topDE) return handle_error();

// use topDE...

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:

```
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 opening /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 arrow 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.

# 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 fifteen*) 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 *Offline Framework*. 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 multiply 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 half-life \* 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 Gnrt.r.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  $^9\text{Li}$ ,  $^8\text{He}$  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 huge amount of optical photons' propagation 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 radioactive isotope's generation in Geant4 is not very reliable. Photon nuclear process triggered by virtual or real photon, pion-nucleus interaction, nucleon-nucleus interaction, etc. are all possible be responsible to spallation background generation. They are poorly described in Geant4. 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 controlled 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 probability 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.



# 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:

```
import DetSim
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:

```
import DetSim.configure
# only load basic physics
detsim = DetSim.configure(physics_list=DetSim.configure.physics_list_basic)
detsim.historian(trackSelection="...", vertexSelection="...")
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`.<sup>1</sup>

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 [f:simtrack\\_accessors](#) and [f:simvertex\\_accessors](#). 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 [f:simtrack\\_and\\_simvertex](#).

The user may decide *which* vertices or tracks get saved, as described in Sec [Creation Rules](#). If a `SimVertex` is pruned from the output, then any references that should have gone to that `SimVertex` instead point to the `SimVertex` preceding 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

---

<sup>1</sup> 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: **f:simtrack\_accessors**  
**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;
    ...
}
```

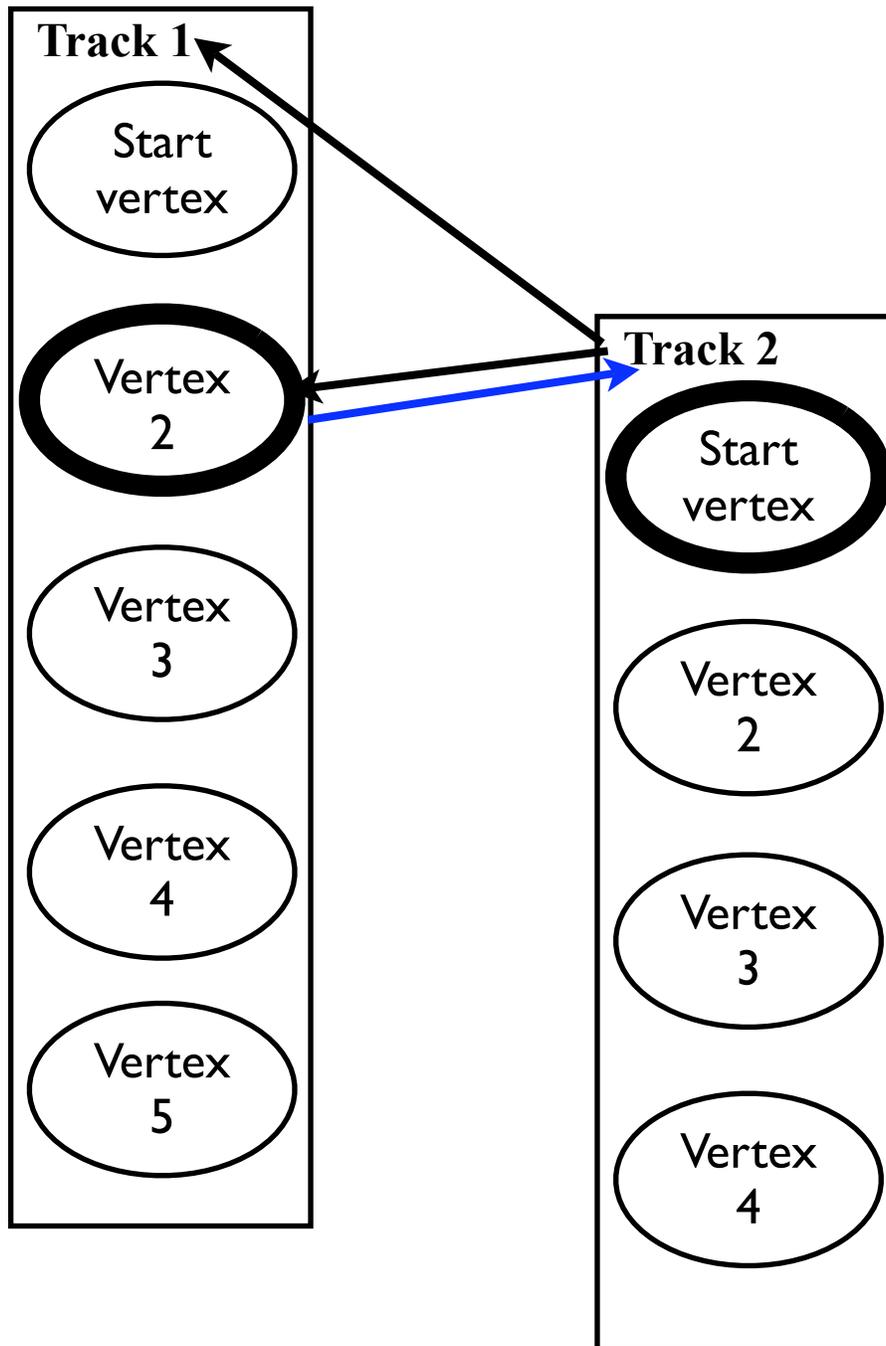
▪

Figure 11.2: **f:simvertex\_accessors**  
**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: **f:simtrack\_and\_simvertex**

**Relationship between SimTrack and SimVertex** Track 1 represents a primary SimTrack, and Track 2 a secondary particle created at the end of Track 1's 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.

particle that the generator makes, and each `SimTrack` is 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 [f:history\\_pruning](#).

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 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 necessary 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 `PARAMETERS` is given in table [t:truthiness\\_parameters](#). Valid `OPERATORS` 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 parameters, `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`,

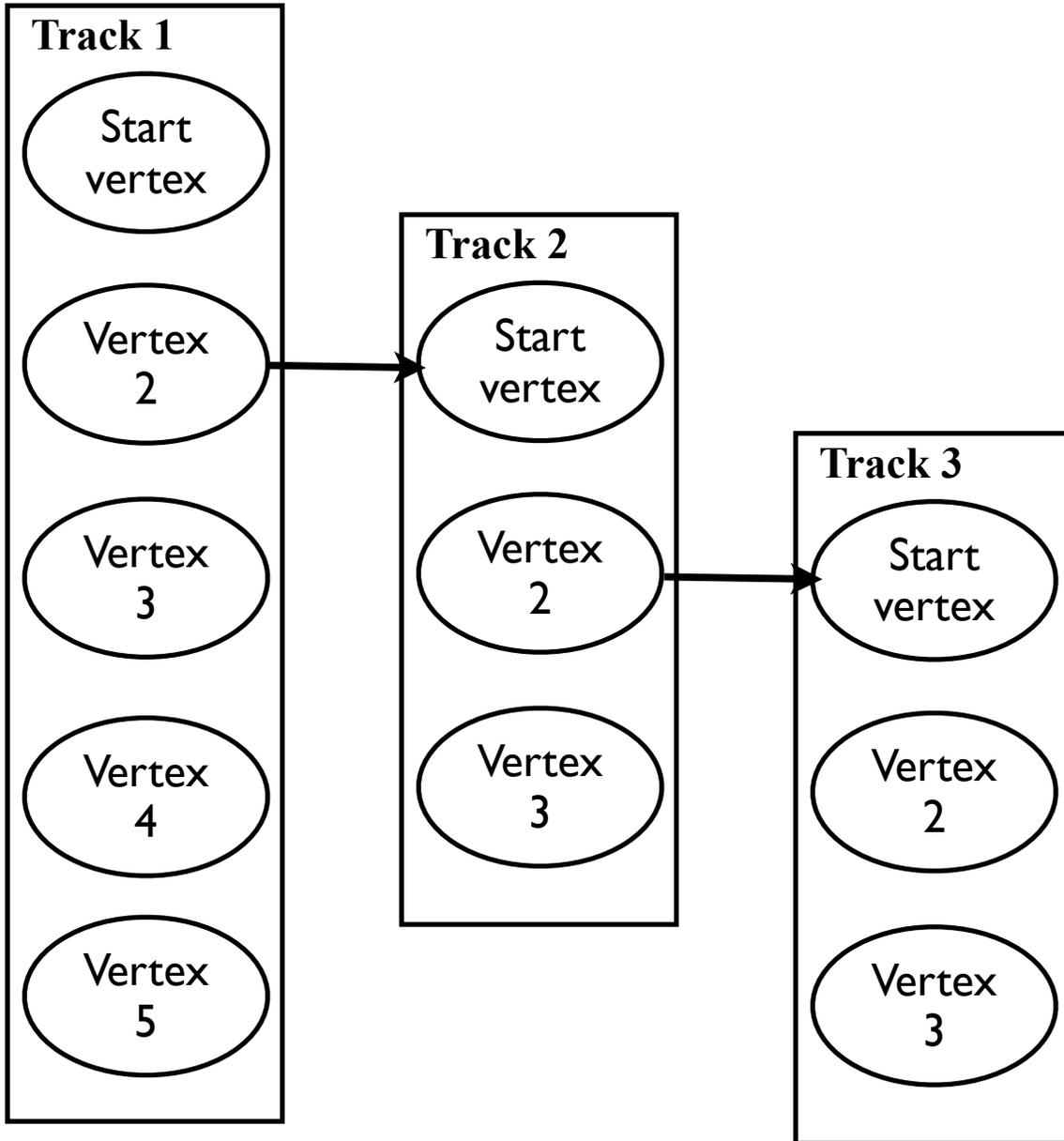


Figure 11.4: f:history\_pruning

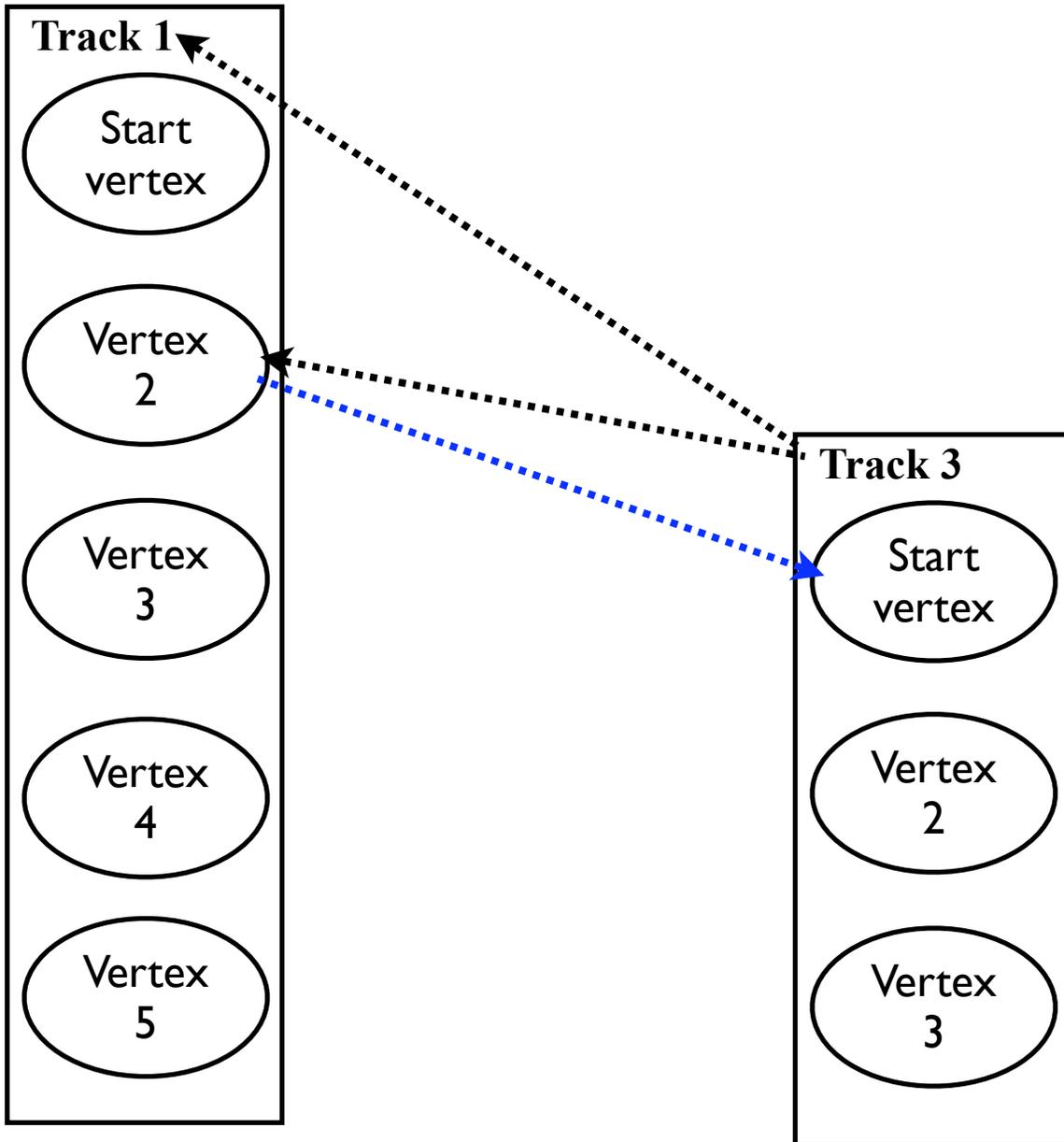


Figure 11.5: **f:history\_pruning**

**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.

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.)

```
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.)

$e^-$	<b>11</b>
$\mu^-$	13
$\gamma$	22
optical photon	20022
neutron	2112
proton	2212
$\pi^0$	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 tricky, 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.

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 Unobservable 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 simulator. 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 *f:simstatistic*.

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 — \*

Figure 11.6: **f:simstatistic**  
**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.
}

["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 *t:truthiness\_parameters* 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. *Creation Rules*. 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 *t:truthiness\_parameters*

### 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 *f:drawhistoryalg\_tracks* and *f:drawhistoryalg\_tracksandvertices*.

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.

**Warning:** latexparser did not recognize : color columnwidth

## 11.4 Truth Parameters

Name & Synonyms	Type	Track	Vertex	Stats	Description
timet	double	X	X	X	Time of the vertex/track start/step
xglobal_x	double	X	X	X	Global X position of the vertex/track start/step
yglobal_y	double	X	X	X	Global Y position of the vertex/track start/step
zglobal_z	double	X	X	X	Global Z position of the vertex/track start/step
rradiuspos_r	double	X	X	X	Global sqrt(X*X+Y*Y) position of the vertex/
lxlocal_xdet_x	double	X	X	X	X Position relative to the local physical volume

Table 11.1 – continued from previous page

lylocal_ydet_y	double	X	X	X	Y Position relative to the local physical volume
lzlocal_zdet_z	double	X	X	X	Z Position relative to the local physical volume
lrlocal_rdet_r	double	X	X	X	$\sqrt{X^2+Y^2}$ position relative to the local p
VolumeVolumeNameLogicalVolume	string	X	X	X	Name of the logical volume of vertex/track star
MaterialMaterialName	string	X	X	X	Name of material at vertex/track start/step
DetectorElementName	double	X	X	X	Name of best-match Detector Element at verte
MatchDetectorElementMatch	double	X	X	X	Level of match for Detector Element. 0=perfec
NicheIdNiche	double	X	X	X	ID number (4-byte) best associated with DetEL
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)
ADAdNumber	double	X	X	X	AD number (1-4)
momentump	double	X	X	X	Momentum at vertex/track start/step
EtotEnergyTotalEnergy	double	X	X	X	Energy at track start or vertex
KEkineticEnergy	double	X	X	X	Kinetic energy at vertex/track start/step
vxdir_xu	double	X	X	X	X-direction cosine
vydir_yv	double	X	X	X	Y-direction cosine
vzdir_zw	double	X	X	X	Z-direction cosine
ProcessType	double	X	X	X	Type of process (see below)
ProcessProcessName	string	X	X	X	Name of current process (via <code>G4VProcess-&gt;</code>
pdgpdgcodeparticle	double	X	X	X	PDG code of particle. Note that opticalphoton-
chargeParticleChargeq	double	X	X	X	Charge of particle
idtrackid	double	X	X	X	Geant TrackID of particle. Useful for debuggin
creatorPdgcreator	double	X	X	X	PDG code for the immediate parent particle
massm	double	X	X	X	Mass of the particle
ParticleName	string	X	X	X	Name of the particle (Geant4 name)
CreatorProcessNameCreatorProcess	string	X	X	X	Name of process that created this particle. (Tra
DetElem inDetectorElement in	custom	X	X	X	Special: matches if the detector element specif
Step_dEdE	double		X	X	Energy deposited in current step
Step_dE_Ionde_ionionization	double		X	X	Energy deposited by ionization in current step
Step_qDEquenched_dEqdE	double		X	X	Quenched energy. Valid only for scintillator
Step_dxStepLengthdx	double		X	X	Step length
Step_dtStepDurationdt	double		X	X	Step duration
Step_dAngledAngle	double		X	X	Change in particle angle before/after this Step
ExE_weighted_x	double		X	X	Energy-weighted global position - x
EyE_weighted_y	double		X	X	Energy-weighted global position - y
EzE_weighted_z	double		X	X	Energy-weighted global position - z
EtE_weighted_t	double		X	X	Energy-weighted global time
qExqE_weighted_xquenched_weighted_x	double		X	X	Quenched energy- weighted global position - x
qEyqE_weighted_yquenched_weighted_y	double		X	X	Quenched energy- weighted global position - y
qEqE_weighted_zquenched_weighted_z	double		X	X	Quenched energy- weighted global position - z
qEtqE_weighted_tquenched_weighted_t	double		X	X	Quenched energy- weighted global time
IsStoppingstopEnd	double		X	X	1 if particle is stopping0 otherwise
IsStartingstartbegin	double		X	X	1 if particle is starting (this is the first step)0 ot
StepNumber	double		X	X	Number of steps completed for this particle
VolumeChangedNew Volume	double		X	X	1 if the particle is entering a new volume0 othe
MaterialChangedNewMaterial	double		X	X	1 if the particle is entering a new material0 oth
ParentPdgAncestorPdgAncestor	double	X	X		PDG code of the last ancestor where a SimTrac
ParentIndirectionAncestorIndirection	double	X	X		Generations passed since the last ancestor was
GrandParentPdgGrandParent	double	X	X		PDG code of the immediate ancestor's ancesto
GrandParentIndirection	double	X	X		Indirection to the immediate ancestor's ancesto
distanceFromLastVertex	double		X		Distance from the last created SimVertex.

**Table 11.1 – continued from previous page**

TimeSinceLastVertex	double		X		Time since the last created SimVertex.
EnergyLostSinceLastVertex	double		X		Energy difference sine the last created SimVert
AngleFromLastVertex	double		X		Change in direction since the last created SimV

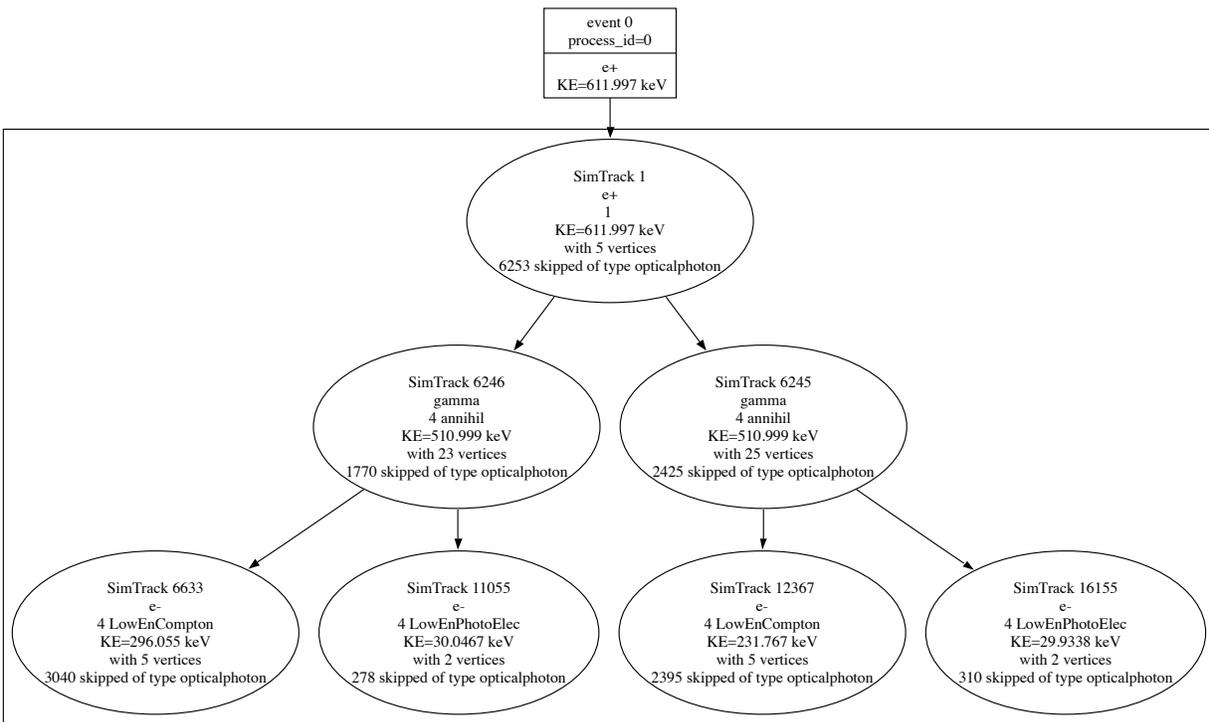
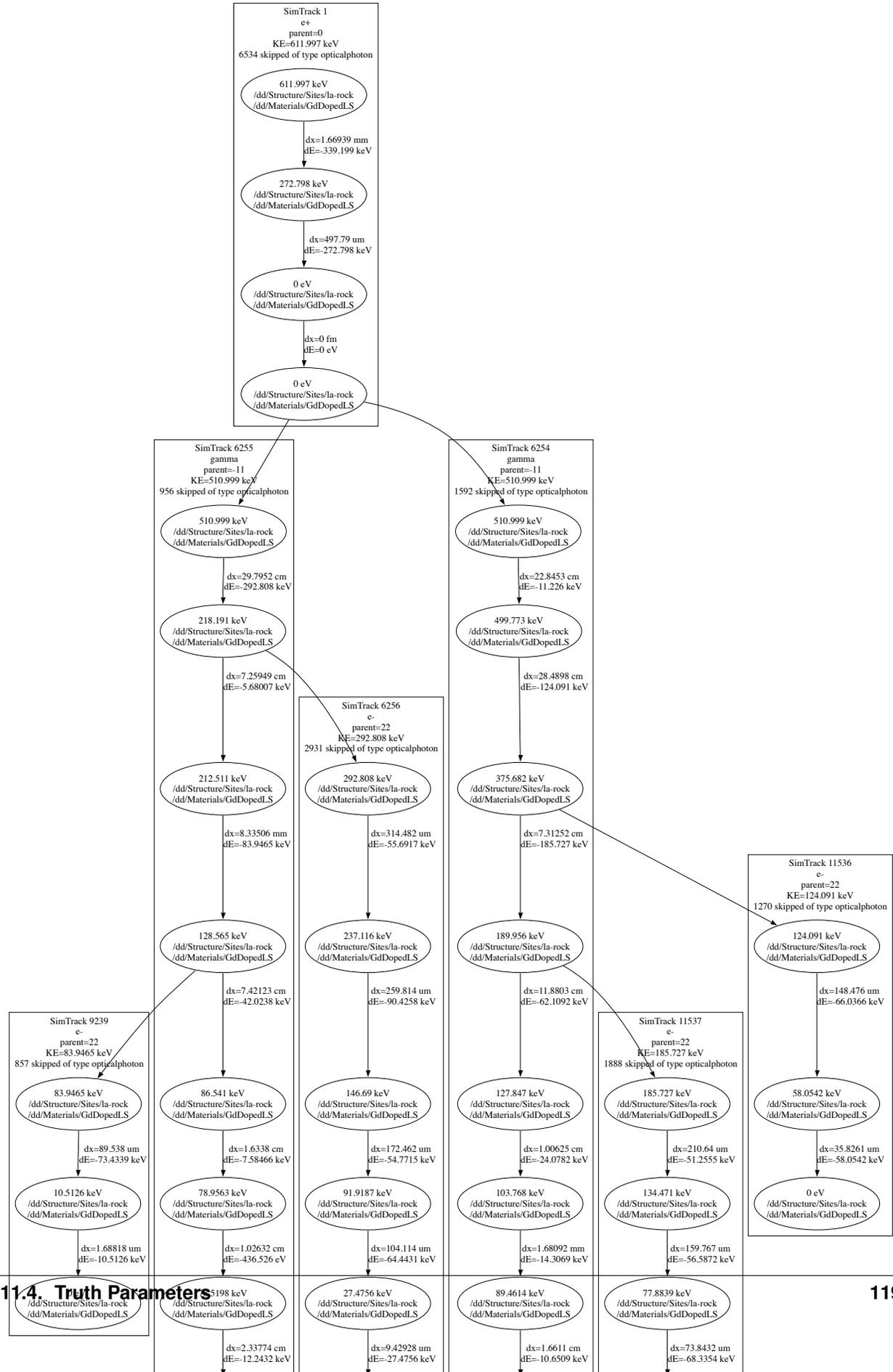


Figure 11.7: `f:drawhistoryalg_tracks`

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.



11.4. Truth Parameters



# 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 *fig-electronics-simchain*. The data model used in ElecSim is summarized in the UML form in figure *fig-electronics-elecsimuml*.

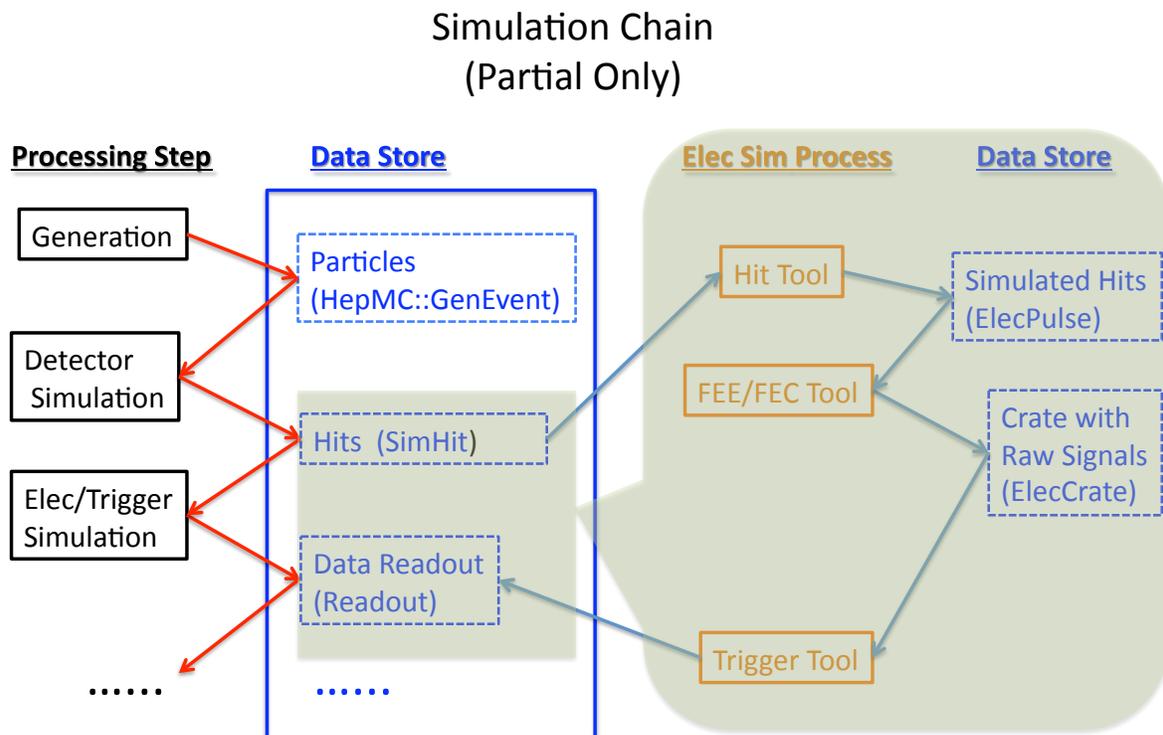


Figure 12.1: **fig-electronics-simchain**  
Electronics Simulation Chain

Location: dybgaudi/DataModel/ElecEvent  
 Current as of: r4061

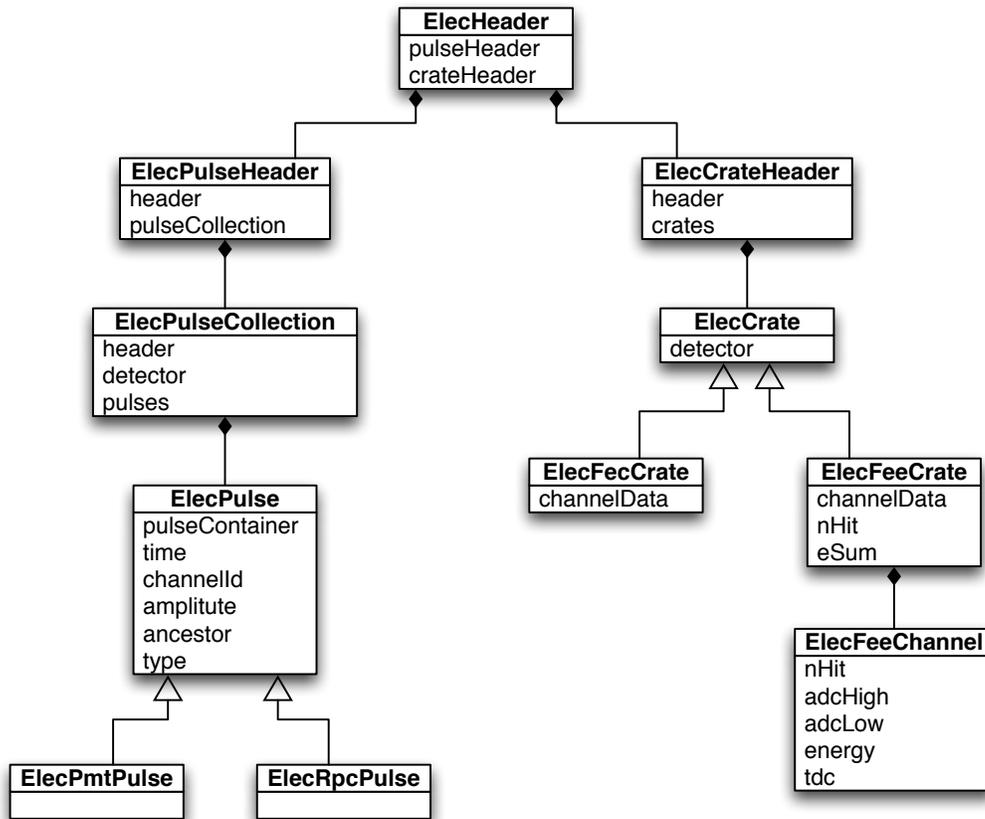


Figure 12.2: fig-electronics-elecsimuml  
 UML for data model in ElecSim.

## 12.2 Algorithms

There are two algorithms. They are listed in table *Algorithms and their properties*.

Table 12.1: Algorithms and their properties.

Algorithm Name	Property	Default
7*EsFrontEndAlg	SimLocation	SimHeaderLocationDefault
2-3	Detectors	DayaBayAD1(2,3,4)
2-3	PmtTool	EsPmtEffectPulseTool
2-3	RpcTool	EsIdealPulseTool
2-3	FeeTool	EsIdealFeeTool
2-3	FecTool	EsIdealFecTool
2-3	MaxSimulationTime	50 us

## 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 summarized in table *Properties declared in EsIdealFeeTool..*

Table 12.2: Properties declared in EsIdealFeeTool.

Property	Default
CableSvcName	StaticCableSvc
SimDataSvcName	StaticSimDataSvc
TriggerWindowCycles	Dayabay::TriggerWindowCycles
NoiseBool	true
NoiseAmp	0.5mV

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 (??).
- 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)**).
- 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**).

$$V(t) = VoltageScale \cdot \frac{(e^{-t/t_0} - e^{-t/t_1})}{(t_1 - t_0)} \tag{12.1}$$

$$t_0 = 3.6ns$$

$$t_1 = 5.4ns$$

### 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 *fig-electronics-npmtgen* shows the flow on how the hitHold signals are generated. One example of two 1 p.e. pulses are shown in figure *fig-electronics-npmtgenexample*.

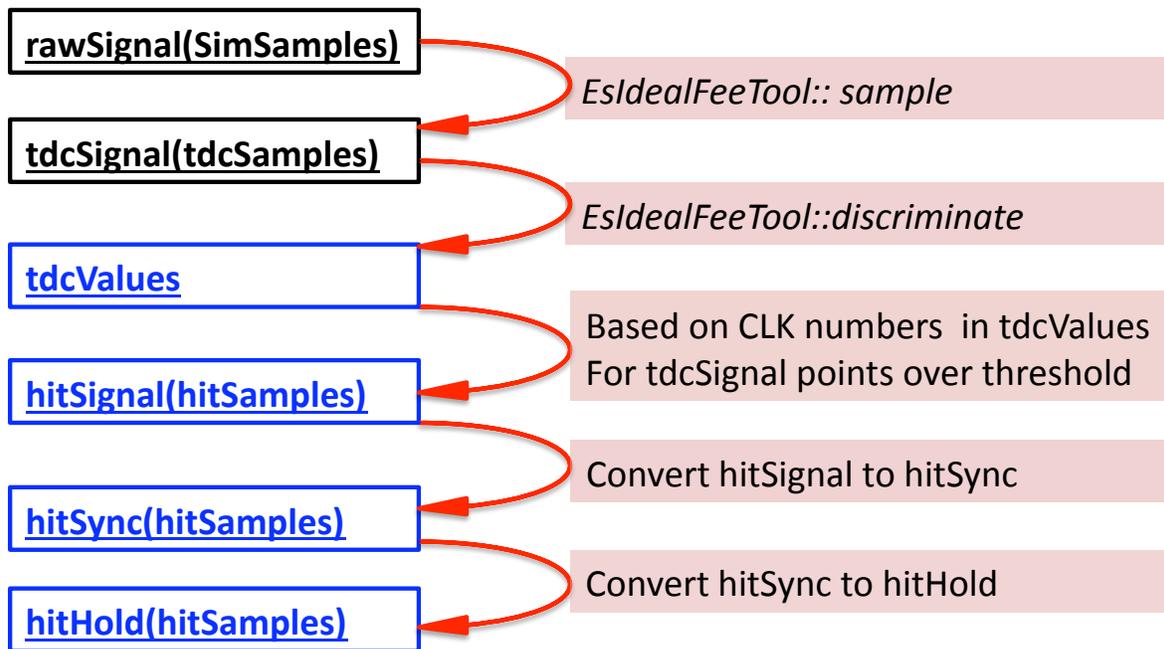


Figure 12.3: **fig-electronics-npmtgen**

: hitHold signal generation sequence. Analog Signals are shown in the black box. And digital signals are shown in blue boxes. On the right hand side, related functions or comments are listed to specify the conversion between different signals.

### ADC Generation

## 12.4 Simulation Constant

Simulation constants based on electronics hardware is defined in `dybgaudi/DataModel/Conventions/Conventions/Electronics.h`. Table *table-elecsim-const* summaries the major variables defined and their hardwired values.



Variable Defined	Value
BaseFrequency	$40 \cdot 1E6$ (hz)
TdcCycle	16
AdcCycle	1
EsumCycle	5
NhitCycle	2
preTimeTolerance	300ns
postTimeTolerance	10us
TriggerWindowCycle	8

**Warning:** latexparser did not recognize : multirow cline

# 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 *fig::simtrigheader*.

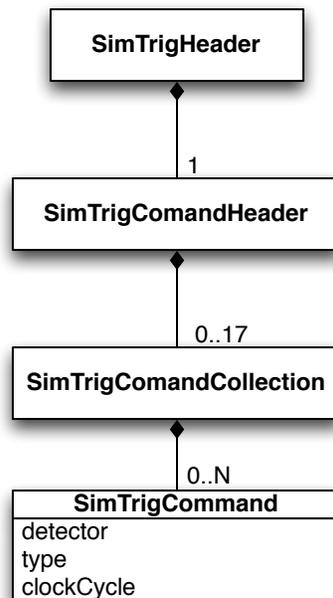


Figure 13.1: **fig::simtrigheader**

SimTrigHeader contains a single SimTrigComandHeader which in turn potentially contains a SimTrigComandCollection for each detector. Each SimTrigComandCollection 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 *TsMultTriggerTool*.

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;
};
```



# 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 satisfy the requirement that only one (1) readout be produced each execution cycle. The details of the header objects are shown in figures *fig::simreadouthead* and *fig::readouthead*

## 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.

**ReadoutEvent**

Modified on Wed Dec 10 2008

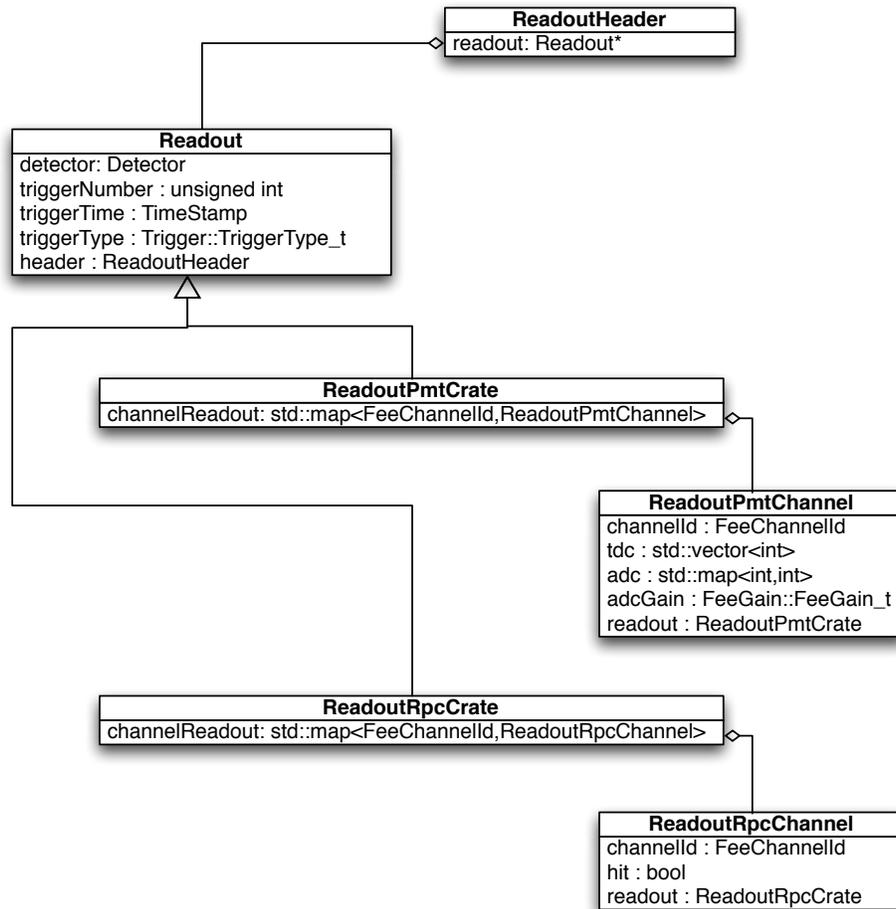


Figure 14.1: **fig::readouthead**

The `ReadoutHeader` contains a single `Readout`. The two flavors of readouts are discussed in *ReadoutHeader*

**SimReadoutEvent**

Modified on Wed Dec 10 2008

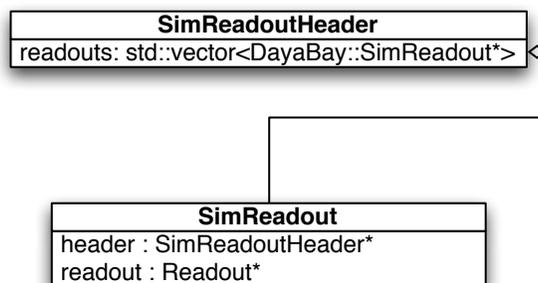


Figure 14.2: **fig::simreadouthead**

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.

## 14.3 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:

### 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
```

# 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. *fig:stages*

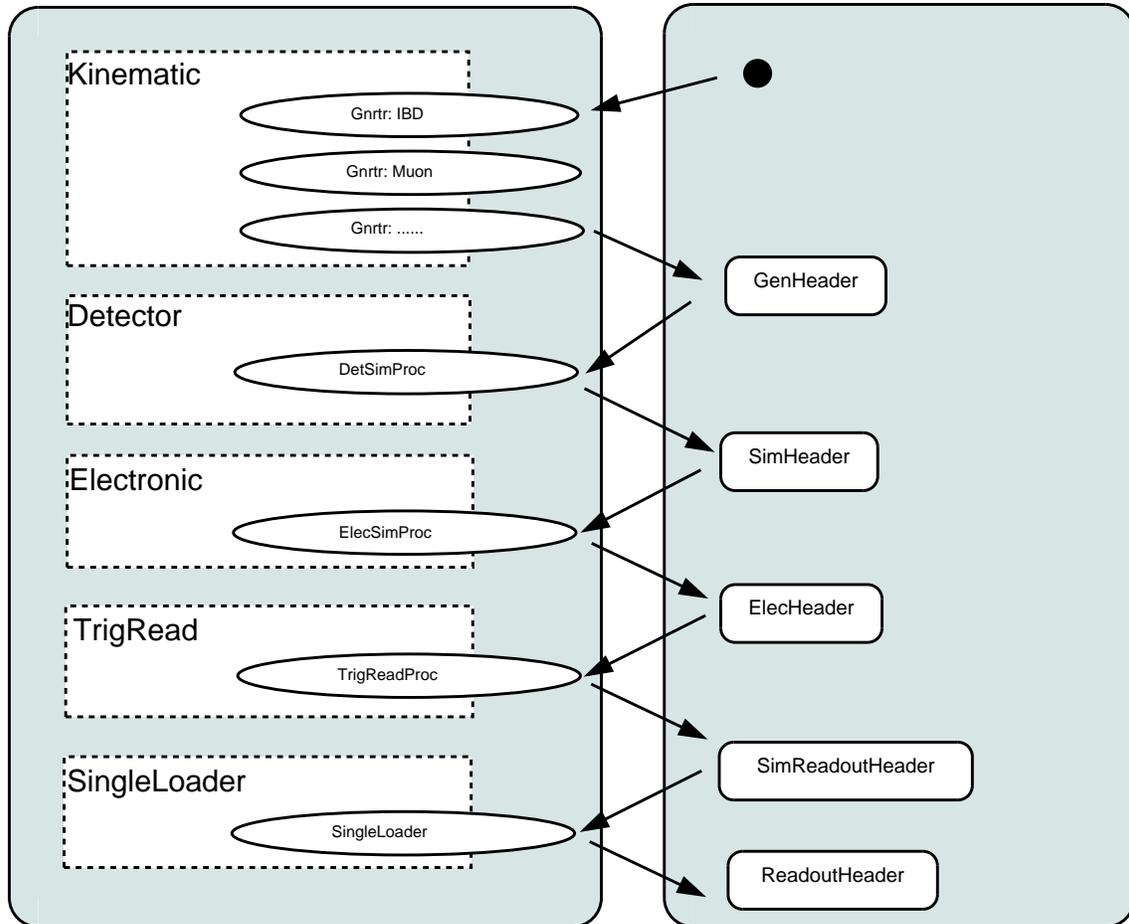


Figure 15.1: **fig:stages**  
Simulation stages.

### 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 simulated MuonProphet muon, a fake hit is created and put into this pipeline. Instead of going into a full electronics 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 observe 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 Gntrt. 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'''

    import DetSim
    ds = DetSim.Configure(physlist=DetSim.physics_list_basic+DetSim.physics_list_nuclear,
                        site="dayabay",
                        use_push_algs = False)

    # QuantumEfficiency*CollectionEfficiency*QEScale = 0.24*1/0.9
    from DetSim.DetSimConf import DsPhysConsOptical
    optical = DsPhysConsOptical()
    #optical.UseScintillation = False
    optical.CerenPhotonScaleWeight = 3.5
    #optical.UseCerenkov = False
    optical.ScintPhotonScaleWeight = 3.5

    from DetSimProc.DetSimProcConf import DetSimProc
    dsp = DetSimProc()
    dsp.ThisStageName = "Detector"
    dsp.LowerStageName = "Kinematic"
    #dsp.OutputLevel = 2
    self.stage_cfg.DetectorSequence.Members.append(dsp)

    ds.historian(trackSelection="(pdg == 2112)",vertexSelection="(pdg == 2112)")

```

```
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.

```
def configureElectronic(self):
    '''Configure the Electronics stage'''

    import ElecSim
    es = ElecSim.Configure(use_push_algs = False)

    from ElecSimProc.ElecSimProcConf import ElecSimProc
    esp = ElecSimProc()
    esp.ThisStageName = "Electronic"
    esp.LowerStageName = "Detector"
    #esp.OutputLevel = 2
    self.stage_cfg.ElectronicSequence.Members.append(esp)

    from ElecSim.ElecSimConf import EsIdealFeeTool
    feetool = EsIdealFeeTool()
    feetool.EnableNonlinearity=False

    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.

```
def configureTrigRead(self):
    '''Configure the Trigger and Readout stage'''
    from TrigReadProc.TrigReadProcConf import TrigReadProc
    tsp = TrigReadProc()
    tsp.ThisStageName = "TrigRead"
    tsp.LowerStageName = "Electronic"
    #tsp.TrigTools = [...]
    #tsp.RoTools = [...]
    #tsp.OutputLevel = 2
    self.stage_cfg.TrigReadSequence.Members.append(tsp)
    return

def configureSingleLoader(self):
    '''Configure the SingleLoader stage'''
    from SingleLoader.SingleLoaderConf import SingleLoader
    sll = SingleLoader()
    sll.ThisStageName = "SingleLoader"
    sll.LowerStageName = "TrigRead"
    #sll.OutputLevel = 2
    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.

```
def configureSim15(self):
    from Stage.StageConf import Sim15
    sim15=Sim15()
    sim15.TopStage=self.opts.top_stage

    from Gaudi.Configuration import ApplicationMgr
    theApp = ApplicationMgr()
    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.

```
def configureLoadingProc(self):
    from LoadingProc.LoadingProcConf import LoadingProc
    load = LoadingProc("LoadingProc.Oxygen18")
    load.StartSec = 0
    load.StartNano = 0
    #load.Distribution = "Exponential"
    load.Distribution = "Periodic"
    load.Rate = 1.0
    assembler_name = "Ox18Assem"
    load.HsAssembler = assembler_name
    load.OutputLevel = 2
    assem = Assembler(toolname = assembler_name,
                      filename = "input.root")

    # This and lower stage
    load.ThisStageName = "Detector"
    load.LowerStageName = ""
    # Add this processor to Gaudi sequencer
    self.stage_cfg.DetectorSequence.Members.append(load)
    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.



# RECONSTRUCTION



# 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

---

**Note:** 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:-

---

**Note:** Merging database tables that have local sequence numbers will require a special procedure to avoid conflicts.

---

**Note:** 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
<code>DBCONF</code>		name of section in config file
<code>DBCONF_URL</code>	<code>mysql://%(host)s/%(database)s</code>	
<code>DBCONF_USER</code>	<code>%(user)s</code>	
<code>DBCONF_PSWD</code>	<code>%(password)s</code>	
<code>DBCONF_HOST</code>	<code>%(host)s</code>	
<code>DBCONF_DB</code>	<code>%(database)s</code>	
<code>DBCONF_PATH</code>	<code>/etc/my.cnf:\$SITEROOT/../../my.cnf: /my.cnf</code>	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 overridden 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 *Holding Open Connections*. 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 [Error Handling](#) and a `Bool_t findFullTimeWindow` see section [Truncated Validity Ranges](#).

## 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 [Error Logging](#). 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  $0 \leq i < \text{GetNumRows}()$ .

*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* dddl = 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 *Truncated Validity Ranges*. 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 *Filling Tables* 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 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](http://home.fnal.gov/%7Edbox/SQL_API_Portability.html)[http://home.fnal.gov/%7Edbox/SQL\\_API\\_Portability.html](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 *MySQL Crib*. 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 *Recommended table row and database column type mappings* gives the recommended mapping between table row, and MySQL column type.

Table 17.1: Recommended table row and database column type mappings

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	

#### Notes

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 [Accessing the Results Table](#) 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 DbResultSet::CurColName() const;
UInt_t DbResultSet::CurColNum() const;
UInt_t DbResultSet::NumCols() const;
DbFieldType DbResultSet::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. `DbFieldType` 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 DbFieldType::IsCompatible(DbFieldType& other) const;
```

and if they are of the same capacity i.e. size:-

```
Bool_t DbFieldType::IsSmaller(DbFieldType& other) const;
```

You can create `DbFieldType` objects e.g:-

```
DbFieldType myFldType(Db::kInt)
```

see enum `Db::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 DbDemoData1" << 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 `DbResultSet` offers are:-

```
UInt_t DbResultSet::CurRowNum() const;
Bool_t DbResultSet::IsExhausted() const;
string DbResultSet::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 implement 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 implement 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 *Concepts* 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 *Concepts*) 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 it 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 *dbi:overlayversiondates*.) 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 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 *Designing a Table Row Class*. 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 DbResultSet::CurColName() const;
UInt_t DbResultSet::CurColNum() const;
UInt_t DbResultSet::NumCols() const;
DbFieldType DbResultSet::CurColFieldType() const;
UInt_t DbResultSet::CurRowNum() const;
string DbResultSet::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 `DbValidityRec` pointer (which is only zero when filling `DbValidityRec` 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 `DbWriter`, 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 `DbCascader` method `CreateTemporaryTable`. You can access the cascader by first locating the singleton `DbTableProxyRegistry` which is in overall charge of the `DatabaseInterface`. The following code fragment shows how you can define a new table for `DbDemoData1`:-

```

#include "DatabaseInterface/DbCascader.h"
#include "DatabaseInterface/DbTableProxyRegistry.h"

...

// Ask the singleton DbTableProxyRegistry for the DbCascader.
const DbCascader& cascader
    = DbTableProxyRegistry::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("DbDemoData1",
                                               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 *MySQL Crib*.

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;\nmysql://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 *Filling Tables*. 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,\n          Int_t aggNo,\n          Dbi::Task task = 0,\n          TimeStamp versionDate = TimeStamp(),\n          UInt_t dbNo = 0);
```

or alternatively you can set it after construction:-

```
DbiWriter<DbiDemoData1> writer(range,aggNo);\nwriter.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);\nwriter.SetDbNo(dbNoTemp);\nwriter.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 than a special ASCII flat file, but each row of its data is a URL 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 *Example*). The first row is special, it contains the column names and types. The types must be valid MySQL types, see table *Recommended table row and database column type mappings* 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.
```

It is 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/> <http://www.mysql.com/>
- from which you can reach a documentation page:-  
<http://www.mysql.com/documentation/index.html> <http://www.mysql.com/documentation/index.html>
- and the downloads for 3.23:-  
<http://www.mysql.com/downloads/mysql-3.23.html>      <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](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 *Recommended table row and database column type mappings* 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 *Example data formats*. shows typical formats of the various data types.

Table 17.2: Example data formats.

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'

### 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 *MakeConnectionsPermanent*.

### 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 *Simple, Compound and Aggregated*. 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);
```

<b>Warning:</b> latexparser did not recognize : href
--

# DATABASE MAINTENANCE

## 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.

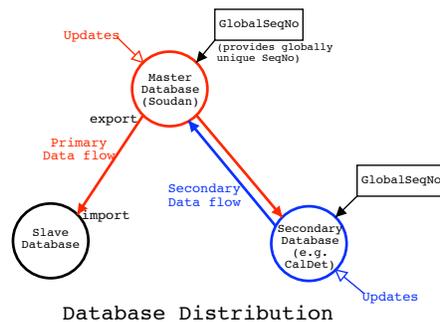


Figure 18.1: **dbm\_db\_distribution\_fig**

The flow of data is shown schematically in diagram *dbm\_db\_distribution\_fig*. 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 *dbi:install*. 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.



# BIBLIOGRAPHY

Bibliography



# TESTING CODE WITH NOSE

## 20.1 Introduction

- References
  - [doc:6280](#) : Encouraging Nose Testing
  - [doc:5258](#) : Your NuWa Testing System
  - [doc:3645](#) : NuWa Offline Software Testing System
  - [doc:3091](#) : NuWa-Trac and Testing System

Many presentations are available describing how to create nosetests, [doc:3645](#) is recommended starting point for the absolute beginner. Also [wiki:Unit\\_Tests](#) provides an excellent introduction.

### 20.1.1 References

#### **[doc:6280](#) : Encouraging Nose Testing**

Make adding tests a zero step process

#### **[doc:5258](#) : Your NuWa Testing System**

Guide to running and creating tests within nose based testing system, allowing NuWa behaviour to be constrained to fulfil the expectations of package experts.

#### **[doc:3645](#) : NuWa Offline Software Testing System**

Demonstrating the ease and usefulness of our software testing system, with the desire to increase its usage.

#### **[doc:3091](#) : NuWa-Trac and Testing System**

Guide to using NuWa-Trac, creating and modifying tickets and running tests, developer guide to adding tests.

## 20.2 Using Test Attributes

As test runs get longer it becomes very useful to control which tests get run in a flexible manner. This functionality is based on the nose `attrib` plugin documented at `nose.plugins.attrib`

- Package Level Nostesting with attributes
- Testing at dybinst level
- Testing at bitten slave level
  - Commit Message controlled deep testing
  - Periodic deep testing based on revision number

### 20.2.1 Package Level Nostesting with attributes

Example based on simple and quick to run `dybgaudi:Database/DybDbi/tests/test_feecablemap.py` for easy checking.

```

from cnf import setup, teardown
from DybDbi import GFeeCableMap

def test_spin():
    r = GFeeCableMap.Rpt()
    print len(r)
    print r[0]
    for i,o in enumerate(r):
        print " %3d feechannelid %d feechanneldesc %s feehardwareid %d sensorid %d sensordesc %s pmt
              ( i, o.feechannelid.fullPackedData() , o.feechanneldesc, o.feehardwareid.id() , o.sens

def test_spin_slowfake():
    r = GFeeCableMap.Rpt()
    print len(r)
    print r[0]
    for i,o in enumerate(r):
        print " %3d feechannelid %d feechanneldesc %s feehardwareid %d sensorid %d sensordesc %s pmt
              ( i, o.feechannelid.fullPackedData() , o.feechanneldesc, o.feehardwareid.id() , o.sens
test_spin_slowfake.minutes = 10

if __name__ == '__main__':
    setup()
    test_spin()
    test_spin_slowfake()
    teardown()

```

#### minutes convention

Assigning an indicative number of minutes to longer running tests allows flexible control over which tests should be run.

The below line assigns a **minutes** attribute to `test_spin_slowfake`.

```
test_spin_slowfake.minutes = 10
```

Subsequently can select which tests using an attribute expression:

```
nosetests -v -A "minutes > 5"
nosetests -v -A "minutes < 5"
```

Real command examples from DybDbi package directory:

- `DBCONF=offline_db nosetests -v -A "minutes > 5"` run only tests for which the attribute expression is true : currently only 1 test
- `DBCONF=offline_db nosetests -v -A "minutes < 5"` run only tests for which the attribute expression is true : currently 270 tests
- `DBCONF=offline_db nosetests -v` run all tests in the package : currently 271 tests
- `DBCONF=offline_db NOSE_EVAL_ATTR="minutes > 5" nosetests -v` using environment controlled attribute setting : runs 1 test
- `DBCONF=offline_db nosetests -v tests/test_feecablemap.py -A "minutes > 5"` runs just `test_spin_slowfake`
- `DBCONF=offline_db nosetests -v tests/test_feecablemap.py -A "minutes < 5"` runs just `test_spin`

## 20.2.2 Testing at dybinst level

Analogously to the above (from `dybsvn:r11731` ) the control can be done at `dybinst trunk tests <pkg-or-alias>` level with:

- `./dybinst trunk tests dybdbi` run all tests in `dybdbi` package, currently 271
- `NOSE_EVAL_ATTR="minutes > 5" ./dybinst trunk tests dybdbi` only tests meeting the expression, currently 1
- `NOSE_EVAL_ATTR="minutes < 5" ./dybinst trunk tests dybdbi` only tests meeting the expression, currently 270

## 20.2.3 Testing at bitten slave level

### Commit Message controlled deep testing

Attribute expressions `attr:<expr>` in svn commit messages like the below are detected and passed to nosetests:

```
example commit message that triggers long tests only attr:"minutes > 10"
example commit message that triggers medium tests attr:"5 < minutes < 10"
```

**Warning:** a build must be triggered within 60 min of the commit time for the `attr:` command to take effect

Protecting quotes is required, eg with:

```
svn ci -m ' attr:"minutes > 5" '
```

Example exercising the machinery, which demonstrates how the bitten-slave access the `SVNLOG_attr` parsed from commit messages.

```
./dybinst -E demo.sh trunk tests dybdbi      # demo.sh contains export statements
```

## Periodic deep testing based on revision number

BUILD\_REVISION is available to dybinst from dybsvn:r11732 and is used to set default attribute expressions that select nosetests from dybsvn:r11738.

build revision	default expression
ends with 00	minutes < 101
ends with 0	minutes < 11
otherwise	minutes < 6

Examples of how to exercise the machinery:

```
BUILD_REVISION=12345 ./dybinst trunk tests dydbdi
BUILD_REVISION=12300 ./dybinst trunk tests dydbdi
BUILD_REVISION=12340 ./dybinst trunk tests dydbdi
```

---

**Note:** commit message attr: commands trump BUILD\_REVISION defaults

---

## 20.3 Running Tests Using dybinst

- Informing dybinst about tests
- Getting the slaves to auto run package tests

### 20.3.1 Informing dybinst about tests

Lists of CMT packages containing tests are configured in installation:dybinst/scripts/dybinst-common.sh,

```
dyb_tests_djaffe="dybalg mdc10b fmcpl1a"
dyb_tests_jetter="elecsim digitizealg"
# add tests here under alias corresponding to your svn username

dyb_tests_db_conditional="dbitest dydbitest dydbdi"    ## conditional on DBCONF sections named after
dyb_tests_default="gaudimessages gentools rootiotest simhistsexample dbvalidate $dyb_tests_djaffe $
dyb_tests_suspects="gentools rootiotest mdc10b"

dyb_tests_db="daqruninfosvc dbidatasvc dbirawdatafilesvc"
dyb_tests_all="$dyb_tests_default $dyb_tests_db dethelpers conventions gendecay"
dyb_tests_failing="detsim"

# these sets define the content of the bitten-slave recipes for configs "dybinst" and "detdesc"
dyb_tests_dybinst="$dyb_tests_default"
dyb_tests_detdesc="xmldetdescchecks"
```

with variables of form dyb\_tests\_<alias> where the alias names djaffe, jetter, suspects, all can be used to refer to the lists of packages. This allows sets of packages to be run with for example:

```
./dybinst trunk tests jetter
./dybinst trunk tests djaffe
./dybinst trunk tests db
./dybinst trunk tests db_conditional
```

No argument corresponds to the default alias, which runs the tests of most of the packages:

```
./dybinst trunk tests
```

### 20.3.2 Getting the slaves to auto run package tests

The bitten-slave follow xml recipes that specify build and test steps to perform. To add tests to the standard set run by the slaves requires these xml recipes to be updated and committed to dybsvn. After modifying `installation:dybinst/scripts/dybinst-common.sh` generate updated bitten-slave xml recipes using commands:

```
./dybinst trunk tests recipe:dybinst
./dybinst trunk tests recipe:opt.dybinst
```

```
cd installation/trunk/dybinst/scripts
svn ci -m "update the slave recipes to include tests for mypkg, mypkgb under the alias mysvnusername"
```

After auto builds have been performed the status of the added test steps run on all the slaves can be seen through the web interface at `build:dybinst` and `build:opt.dybinst`.

## 20.4 Testing nose plugins

- Setup virtualenv sandbox
- Get into the virtualenv
- Interesting Plugins
  - nosepipe
  - insulate

### 20.4.1 Setup virtualenv sandbox

1. Install virtualenv (only this step requires write access to *nuwa* installation):

```
./dybinst trunk external virtualenv
```

2. Get virtualenv into your PATH:

```
cd $SITEROOT/lcgcm/LCG_Interfaces/virtualenv/cmt
cmt config ; . setup.sh
which virtualenv          ## should be the NuWa one
```

3. Create virtual python environment, *spawned* from nuwa python eg:

```
virtualenv ~/v/nose
```

For background info on virtualenv see <http://www.virtualenv.org/en/latest/>

### 20.4.2 Get into the virtualenv

1. Get into the environment:

```
. ~/v/nose/bin/activate
which pip python easy_install ## should all be from ~/v/nose/bin
```

2. install plugin:

```
pip install nosepipe
```

3. list plugins:

```
PYTHONPATH=~/.v/nose/lib/python2.7/site-packages:$PYTHONPATH nosetests -p
```

### 20.4.3 Interesting Plugins

Many 3rd party plugins:

- <http://nose-plugins.jottit.com/>

#### nosepipe

Plugin for the nose testing framework for running tests in a subprocess

- <http://code.activestate.com/pypm/nosepipe/>

Such a feature would be very useful for DBI testing in order to work with different DBI configurations within a single test run. But it is not clear about the granularity control, would want each module of tests to correspond to a separate process.

From the help:

```
PYTHONPATH=~/.v/nose/lib/python2.7/site-packages:$PYTHONPATH nosetests --help
--with-process-isolation
    Enable plugin ProcessIsolationPlugin: Run each test in
    a separate process. [NOSE_WITH_PROCESS_ISOLATION]
```

But looks like not running in py27:

```
(nose) [blyth@belle7 ~]$ PYTHONPATH=~/.v/nose/lib/python2.7/site-packages:$PYTHONPATH nosetests --with-
setup 22041
ERROR
ERROR
ERROR
teardown 22041
```

```
=====
ERROR: test_mp.test_red
-----
Traceback (most recent call last):
  File "/data1/env/local/dyb/external/nose/0.11.4_python2.7/i686-slc5-gcc41-dbg/lib/python2.7/site-pa
    self.runTest(result)
  File "/data1/env/local/dyb/external/nose/0.11.4_python2.7/i686-slc5-gcc41-dbg/lib/python2.7/site-pa
    test(result)
  File "/home/blyth/v/nose/lib/python2.7/site-packages/nosepipe.py", line 152, in __call__
    (request_len, len(data)))
Exception: short message body (want 1416782179, got 207)
```

#### insulate

- <http://code.google.com/p/insulatenoseplugin/wiki/Documentation>

### About this testing section

The documentation is sourced from reStructuredText in `dybgaudi:Documentation/OfflineUserManual/tex/nose`, and html and pdf versions are derived as part of the automated Offline User Manual build. For help with building see *Build Instructions for Sphinx based documentation*



# STANDARD OPERATING PROCEDURES

**Release** 17726

**Date** August 03, 2012

This documentation attempts to provide the practical knowledge needed to perform database operations. Inner details of how DBI works and conceptual background are not covered, these are available at *Database Interface*. A very brief description of some DBI conventions is provided in *DBI Very Briefly*.

The description of DB operations are divided into sections:

1. *DB Definitions* to facilitate communication
2. *DBI Very Briefly*
3. *Rules for Code that writes to the Database*
4. *Configuring DB Access*
5. *DB Table Updating Workflow*
6. *DB Table Writing*
7. *DB Table Reading*
8. *Debugging unexpected parameters*
9. *DB Table Creation*
10. *DB Validation and Testing*
11. *DB Administration*
12. *DB Services*
13. *DCS tables grouped/ordered by schema*
14. *Non DBI access to DBI and other tables*
15. *Scraping source databases into offline\_db*
16. *DBI Internals*
17. *DBI Overlay Versioning Bug*

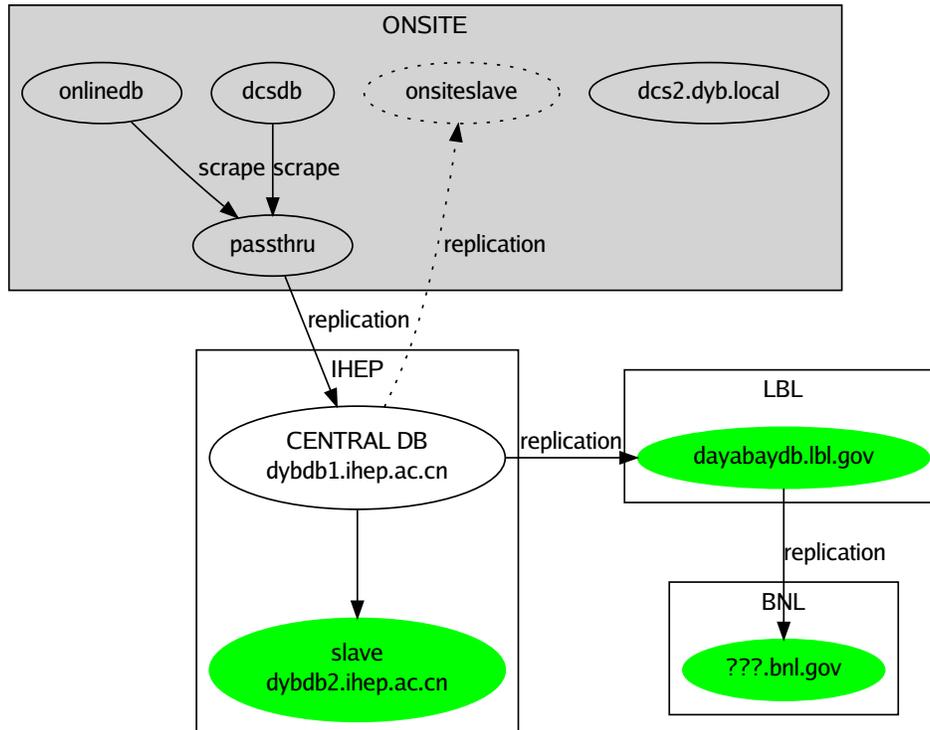
Detailed table of contents:

## 21.1 DB Definitions

For clarity of expression common naming of the various components is useful.

## 21.1.1 Database Topology Diagram

`offline_db` replication data flow



Future plans:

1. Offline DB slave Onsite as well (repairs on same hardware as passthru DB )

### Which Database to read from ?

Use nearest replicated *slave* of `offline_db`, ie `dybdb2.ihep.ac.cn`

### Which Database to write to ?

Your copy of `offline_db`, known as `tmp_offline_db`

Database content and handling is divided into two categories with very different handling:

- monitored DCS/DAQ quantities that are automatically scraped into the `offline_db` by continuously running scripts

- calibration parameters that are calculated based on data taking files and updated in an initially manual manner

The above figure is sourced in `dybgaudi:Documentation/OfflineUserManual/tex/sop/dbdefn.rst`, please commit any corrections/updates to the figure (in `dot/graphviz` language). The figure is gleaned mostly from p9 of `doc:4449 cet091219offline-database.ppt.pdf`

## 21.1.2 Database names

### Section Names or Database names

This documentation refers to databases by their configuration file section names such as `tmp_offline_db` rather than by the actual database name (eg `tmp_username_offline_db`), as this parallels the approach taken by the tools: `db.py` and DBI.

For clarity a few definitions are required

*offline\_db* central DB at IHEP

*tmp\_offline\_db* temporary copies of *offline\_db*

## 21.2 DBI Very Briefly

- Validity Tables
- Validity Table Timestamps
  - How these times fit in
  - Choosing Validity Ranges
  - Rollback and Production
- Using Rollback to Debug/Workaround problem DB entries
- What is TASK for ?
- Features of Clean Validity Tables
  - Overlay Versioning
- DBI Q and A
  - Doesnt TIMEEND of EOT overshadow valid entries when we correct an earlier entry ?
  - How do we make sure not to end up with SEQNO gaps ?
  - If my update has a given SEQNO in my tmp\_offline\_db, will it have the same in the offline\_db ?
  - What are fastforward commits ? Why are they needed ?
  - Or is the offline\_db smart so that it automatically gives it the next number in the sequence ?
  - How can SEQNO be missed?

### 21.2.1 Validity Tables

DBI validity tables are the heart of how DBI operates:

```
mysql> describe TableNameVld ;
```

Field	Type	Null	Key	Default	Extra
SEQNO	int(11)	NO	PRI	NULL	auto_increment
TIMESTART	datetime	NO		NULL	

TIMEEND	datetime	NO		NULL	
SITEMASK	tinyint(4)	YES		NULL	
SIMMASK	tinyint(4)	YES		NULL	
SUBSITE	int(11)	YES		NULL	
TASK	int(11)	YES		NULL	
AGGREGATENO	int(11)	YES		NULL	
VERSIONDATE	datetime	NO		NULL	
INSERTDATE	datetime	NO		NULL	

+-----+-----+-----+-----+-----+-----+

10 rows in set (0.00 sec)

## 21.2.2 Validity Table Timestamps

Each validity entry includes 4 timestamps:

**TIMESTART** start of context range

**TIMEEND** end of context range, often end-of-time

**VERSIONDATE** used by overlay versioning to distinguish otherwise equal validities, overlay versioning usage if signalled by using `versiondate=TimeStamp(0,0)` in writer contexts. allowing easy overriding ... just rewrite with same contextrange to override

**INSERTDATE** the actual insert time, used by rollback to select a snapshot of DB at a chosen time (or times ... this can be a per-table time) This means : **NEVER CHEATED** ... should always be actual UTC now of the `offline_db` update.

### How these times fit in

Stating the obvious, in order to clarify the large numbers of timestamps floating around:

The timestamps embedded into real datafiles and simulation files, form the contexts used to make DBI queries so database validity `TIMESTART/TIMEEND` must be appropriate for those embedded timestamps.

### Choosing Validity Ranges

The choice of validity range should be made as appropriate to the parameters.

In the case of MC production runs which have pre-defined non-overlapping and monotonically increasing time ranges, it is straightforward to choose `TIMESTART`. Where you suspect validity may extend beyond a single production using `TimeStamp.GetEOT()` for `TIMEEND` is the appropriate choice. Subsequent writes can of course override these entries.

### Rollback and Production

The DBI **ROLLBACK** feature is very important for controlled production usage of DBI. A global timestamp or per-table timestamps are defined that all DBI queries incorporate, allowing the DB tables seen by all production jobs(or reruns thereof) to be the same no matter what DB updates are done in the meantime.

Reprocessing an existing dataset following DB updates with improved parameters would entail definition of a new set of rollback dates to benefit from the improved parameters.

Note that these rollback dates pertain **only** to the `INSERTDATE` used. This is orthogonal to the `TIMESTART/TIMEEND` which pertains to the timestamps which are embedded into the files.

Note this presupposes DBI is used appropriately:

1. no deletions
2. no changes to existing entries
3. only additions are permissible

Deletions/changes are only allowed at the initial setup stage.

### 21.2.3 Using Rollback to Debug/Workaround problem DB entries

To verify that a DB update is causing issues or to workaround such problems it is possible to utilise DBI rollback to return to a prior state of all or some of the tables in the DB. This works by applying *INSERTDATE < rollbackdate* cuts

For example setting the rollback date for all tables:

```
DBCONF_ROLLBACK="* = 2011-10-01 08:08:08" nuwa.py ...etc...
```

Single tables:

```
DBCONF_ROLLBACK="CalibPmtSpec = 2011-10-01 08:08:08" nuwa.py ....etc...
```

Multiple tables via comma delimited mappings:

```
DBCONF_ROLLBACK="CalibPmtSpec = 2011-10-01 08:08:08,EnergyRecon = 2011-05-01 08:08:08, " nuwa.py ..
```

Wildcarded sets of tables:

```
DBCONF_ROLLBACK="Cal* = 2011-10-01 08:08:08" nuwa.py ....etc...
```

Combine global setting with table specific ones using comma delimited string:

```
DBCONF_ROLLBACK="* = 2011-10-01 08:08:08,Cal* = 2011-10-01 08:08:08"
```

The above envvar setting approach is bash specific, if you must use inferior shells you will probably need to translate into "setenv DBCONF\_ROLLBACK ... ; nuwa.py ..."

### 21.2.4 What is TASK for ?

TASK is usually left at its default value of zero, greater than zero values are used for testing out non-default algorithms.

### 21.2.5 Features of Clean Validity Tables

1. SEQNO starting from 1 and with no gaps, with maximum corresponding to the LASTUSEDSEQNO
2. Far future times all using TimeStamp.GetEOT() namely 2038-01-19 03:14:07
3. Overlay versioning in use, see below

An example of a clean table with SEQNO = 1:213:

```
mysql> select * from CableMapVld ;
```

SEQNO	TIMESTART	TIMEEND	SITEMASK	SIMMASK	SUBSITE	TASK	AGGREGATE
*1*	2009-03-16 11:27:43	2038-01-19 03:14:07	1	2	2	0	
2	2009-03-16 11:27:43	2038-01-19 03:14:07	2	2	6	0	
3	2009-03-16 11:27:43	2038-01-19 03:14:07	4	2	6	0	
4	2009-03-16 11:27:43	2038-01-19 03:14:07	4	2	5	0	

5	2009-03-16 11:27:43	2038-01-19 03:14:07	4	2	4	0
...						
208	2011-05-23 08:22:19	2038-01-19 03:14:07	2	2	7	0
209	2011-05-23 08:22:19	2038-01-19 03:14:07	4	2	7	0
210	2011-05-23 08:22:19	2038-01-19 03:14:07	1	2	7	0
211	2011-05-23 13:09:43	2038-01-19 03:14:07	2	2	7	0
212	2011-05-23 13:09:43	2038-01-19 03:14:07	4	2	7	0
*213*	2011-05-23 13:09:43	2038-01-19 03:14:07	1	2	7	0

\*213\* rows in set (0.34 sec)

LOCALSEQNO table contains the last used SEQNO for each table, 213 for CableMap:

```
mysql> select * from LOCALSEQNO ;
+-----+-----+
| TABLENAME | LASTUSEDSEQNO |
+-----+-----+
| *          | 0              |
| CalibFeeSpec | 113           |
| CalibPmtSpec | 29            |
| FeeCableMap | 3             |
| CableMap    | 213           | <<<<<<<< 213 <<<<<<<
| HardwareID  | 172           |
+-----+-----+
6 rows in set (0.14 sec)
```

### Overlay Versioning

**VERSIONDATE is more VERSION than DATE**  
 Is better thought of as a VERSION number than rather than a timestamp. Notice the artificial 1 minute jumps in the below VERSIONDATE values.

Overlay versioning is visible by the 1 min differences in VERSIONDATE between overlapping validities. These VERSIONDATE are filled in automatically by DBI when signalled to do so by the special context argument versiondate=TimeStamp(0,0) . As DBI validity queries are done in descending VERSIONDATE order with the SQL: ordered by VERSIONDATE desc this allows updates to prior entries to be simply achieved by re-writing with the same contextrange and with overlay versioning enabled.

Query to find overlapping validities, that are distinguished by VERSIONDATE:

```
mysql> select * from CableMapVld where sitemask=1 and subsite=1 ;
+-----+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART          | TIMEEND          | SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATE |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 14    | 2009-03-16 11:27:43 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 22    | 2009-06-03 21:36:27 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 35    | 2010-12-07 19:14:20 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 57    | 2011-02-08 15:49:51 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 71    | 2011-02-22 12:38:11 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 85    | 2011-02-22 17:08:51 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 99    | 2011-02-22 18:07:45 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 113   | 2011-02-23 10:49:36 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 127   | 2011-03-25 19:31:49 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 143   | 2011-04-01 17:29:23 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
| 159   | 2011-04-18 03:42:40 | 2038-01-19 03:14:07 | 1        | 2        | 1        | 0    |           |
```

```

| 175 | 2011-04-19 23:56:10 | 2038-01-19 03:14:07 | 1 | 2 | 1 | 0 |
| 191 | 2011-05-03 02:35:09 | 2038-01-19 03:14:07 | 1 | 2 | 1 | 0 |
| 207 | 2011-05-05 17:42:22 | 2038-01-19 03:14:07 | 1 | 2 | 1 | 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.09 sec)

```

## 21.2.6 DBI Q and A

### Doesnt TIMEEND of EOT overshadow valid entries when we correct an earlier entry ?

This is the most frequently stated fallacy about DBI. See the above section *Overlay Versioning*. Essentially DBI always orders validities (Vld entries) by VERSIONDATE, **NOT** by INSERTDATE. This means that by virtue of overlay versioning (VERSIONDATE is derived from TIMESTART with minute offsets) you can go back and override a former commit (still using EOT) and not override your recent entries for subsequent times.

### How do we make sure not to end up with SEQNO gaps ?

1. use DBI/DybDbi to prepare updates
2. avoid raw SQL fixes or doing nasty things like editing your ascii catalogs
3. be careful regards re-running updates more that once, you can always start with a fresh tmp\_offline\_db if you do this by mistake
4. check LOCALSEQNO table after updates, it should contain the LASTUSEDSEQNO for your updated tables

### If my update has a given SEQNO in my tmp\_offline\_db, will it have the same in the offline\_db ?

Yes, but it is unwise to do anything based on hardcoded SEQNO

Your table in tmp\_offline\_db is **rdumpcat** into *dybaux* then **rloadcat** into offline\_db in a way that keeps the content exactly the same and SEQNO is preserved. The only thing that is changed is the INSERTDATE, which is fastforwarded to the UTC now date of the actual insert.

### What are fastforward commits ? Why are they needed ?

Fastforward commits are changes to the INSERTDATE validities that are made by the script (*dbaux.py*) that DB managers use to propagate a dybaux catalog commits into offline\_db. After updates are propagated these working copy changes are committed to dybaux.

This fastforwarding of INSERTDATE to the time of the actual offline\_db insert is required to avoid windows of ambiguity between the time the insert is done into tmp\_offline\_db and the time that gets propagated into offline\_db.

### Or is the offline\_db smart so that it automatically gives it the next number in the sequence ?

DBI supplies the next SEQNO in your tmp\_offline\_db, the steps from there to offline\_db simply copy it.

### How can SEQNO be missed?

Either directly by deletions or from failure modes, eg:

1. a re-run that doubles up your SEQNO in LOCALSEQNO, followed by cleanup of Payload and Vld but not LOCALSEQNO entry could result in missing many SEQNO

Automatic and manual validations should pick up such issues.

## 21.3 Rules for Code that writes to the Database

### 21.3.1 Scope of Rules

All code that writes into the Offline Database is required to abide by the regulations. This includes:

1. Calibration writers
2. Automated Scrapers

<b>Warning:</b> code that prepares the parameters is also covered by the rules
--

### 21.3.2 DB Writing Code Management

- code must be reviewed by DB Managers or their delegates

Code reviewers must verify :

- code is housed(and developed) in dybsvn repository CMT packages
- packages have nosetests that can be run by everyone (including the slaves)
- uses DBI (either directly or via DybDbi), no raw SQL
- all times in UTC
- context range end validity times, standardized far future time as `TimeStamp::GetEOT()`
- **uses enums rather than bare integers**
  - if enums do not exist they need to be created
- ... ( more suggestions )

### 21.3.3 Rationale for dybsvn rule

Housing and developing code in dybsvn has several advantages:

- allows referencing the the state of the code with a single integer : the revision number
- easy for all collaborators to see the code that prepared the update now and in the future

Abiding by this rule is a crucial requirement for the creation of reproducible calibration parameters (and by extension reproducible physics results).

### 21.3.4 Recording how DB updates were prepared

Good practices to adopt to record how an update was prepared

- include documentation (in any text based format) alongside code in SVN to provide a record of the algorithm used and any changes to it
- include simple “no argument” scripts in SVN that run your flexible scripts or executables in order to capture the arguments used. These “no argument” simple scripts can be named after the update and will prove useful for subsequent updates.
- ensure that your final values are created with a clean revision (svnversion needs to report an integer without an “M” for modified).
- record the revision number

### 21.3.5 Verification of reproducibility

Although time consuming the best way to ensure that results are reproducible is to test this by

- requesting collaborators from another cluster/continent to duplicate results using just what was obtained from an SVN checkout (at a defined revision) and data files (which presumably have standard naming that allows them to be accessed from different clusters).

### 21.3.6 Testing DB writing code

- development against *offline\_db* is prohibited
- developing against local copies of *offline\_db* is recommended

Follow the example provided by `dybgaudi:Database/DBWriter/tests` which demonstrate best working practices for testing DB Writing code, where every step is fully controlled.

- starts from the vacuum
- creates an empty DB
- creates tables descriptions in the DB
- populates DB with pre-requisite entries (a `DaqRunInfo` row in this case) using `DybDbi`
- invokes the `DBWriter` script in a separate process, using `dybtest.Run`
- does reference comparisons on the output of the script
- does reference comparisons on the `mysqldumped` database that results from the running of the script

This approach allows frequent automated running of the test

## 21.4 Configuring DB Access

- Create Simple DB configuration file
  - Standardized Section Names
  - Section dependent testing
- DBCONF envvar
  - Cascade configuration
  - Configuring access to ascii catalog
  - Using dybaux as ascii catalog
- N ways to set an envvar
  - bash
  - Inferior shells such as tcsh/csh
  - python
- Background Information
  - What is a mysql dump file ?
  - What is a DBI ascii catalog ?
- Hands-On Exercise 1 : Troubleshooting DB connection configuration
  - Check with mysql client
  - Check with db.py
  - Check with DBI
  - DBI error when DBCONF not defined
- Hands On Exercise 2 : Interactive DybDBI
  - Get into ipython
  - Interactively verify connection
  - Interactive Exploration with ipython TAB completion
  - DybDbi with some magic

As both DBI and `db.py` make heavy usage of the mysql configuration file and as this is the primary source of problems for beginners, the below elaborates on how to setup your configuration and troubleshoot problems.

### 21.4.1 Create Simple DB configuration file

#### CAUTION

Keep your configuration file clean and simple with obvious correspondence between section names and DB names.

Create a configuration file in your home directory `~/.my.cnf` containing parameters to connect to relevant databases, for example:

```
[offline_db]
host      = dybdb2.ihep.ac.cn
database = offline_db
user      = dayabay
password  = youknowit

[tmp_offline_db]
host      = dybdb2.ihep.ac.cn
database = tmp_wangzhm_offline_db
user      = wangzhm
password  = plaintestpw

[client]
host      = dybdb2.ihep.ac.cn
```

```
database = tmp_wangzhm_offline_db
user = wangzhm
password = plaintextpw
```

### Section Names

Note that the section names **offline\_db**, **tmp\_offline\_db** do not exactly correspond to DB names, providing useful indirection : but keep it simple to avoid confusion.

**Warning:** At IHEP it is recommended that users connect to the slave machine dybdb2.ihep.ac.cn

The commandline `mysql` client by default reads the `client` section of the configuration file.

**Note:** localhost access

For localhost access, some systems are configured to use a location for the MySQL socket that is different than the hard-coded default of `/tmp/mysql.sock` and defining a “[client]” section will override this configuration. For such systems you must restore the “socket” directive by including it in your `.my.cnf`.

### Standardized Section Names

section	references	role
offline_db	nearest slave copy of master	readonly access to content of central db
tmp_offline_db	temporary copy of offline_db	testing ground for updates, fair game to be dropped

#### transient nature of tmp\_offline\_db

make fresh copy from offline\_db when working on updates : **avoiding merge problems**

Allows:

1. easy communication
2. scripts to have wider applicability, due to common roles
3. testing system to tailor tests based on sections available

### Section dependent testing

The test is only run if all DBCONF sections are available in the configuration file.

```
from DybPython import DBConf
want_conf = 'cascade_0:cascade_1:cascade_2'
has_conf = DBConf.has_config( want_conf )

def setup():
    os.environ['DBCONF'] = want_conf

def test_cascade():
    for dbno in range(3):
        ...
test_cascade.__test__ = has_conf
```

## 21.4.2 DBCONF envvar

DBI uses the configuration section pointed to by the `DBCONF` environment variable. For example:

```
DBCONF=offline_db nuwa.py ...
DBCONF=tmp_offline_db nuwa.py ...
DBCONF=offline_db python -c "from DybDbi import gDbi ; gDbi.Status() "
```

For recommendations on how to set envvars on the commandline and in scripts, see below *N ways to set an envvar*

Further details on `DBCONF` and related envvars are in [doc:5290](#).

### Cascade configuration

Configuring a cascade is achieved by using multiple section names delimited by a colon, for example:

```
DBCONF=tmp_offline_db:offline_db nuwa.py ...
DBCONF=tmp_offline_db:offline_db python -c "from DybDbi import gDbi ; gDbi.Status() "
```

The first section name takes priority in the cascade.

### Configuring access to ascii catalog

A config section like the below with a database value of `dbname#/absolute/path/to/catalog/file.cat` specifies the catalog to use and the database into which temporary tables are loaded:

```
[tmp_offline_db_ascii]
host = your.local.domain
user = joe
password = plaintextpw
database = tmp_joe_offline_db#/home/joe/tmp_offline_db/tmp_offline_db.cat
```

Including such a section name in `DBCONF` allows the content of the catalog to be accessed. For a quick test get into `dybgaudi:Database/DybDbi` and:

```
DBCONF=tmp_offline_db_ascii          python tests/test_feecablemap.py
DBCONF=tmp_offline_db_ascii          python -c "from DybDbi import gDbi ; gDbi.Status() "
DBCONF=tmp_offline_db_ascii:offline_db python -c "from DybDbi import gDbi ; gDbi.Status() "
```

#### mysql temporary tables can be inconvenient

The single session nature of MySQL temporary tables and their evaporation after a single usage means that they cannot be examined with the mysql client. An alternative approach is to use normal **browsable** tables in a non-standard DB and place the corresponding `DBCONF` string at the front of the DBI cascade.

Caveats arising from DBI ascii catalog implementation with MySQL temporary tables:

1. `CREATE_TEMPORARY` permission is required in the specified database
2. the temporary tables only exist for a single session, they are atomically loaded from the catalog at each DBI startup

### Using dybaux as ascii catalog

Note that ascii catalog config can use a URL rather than the absolute path to a checkout:

```
[tmp_offline_db_ascii]
host = your.local.domain
user = joe
password = plaintextpw
database = tmp_joe_offline_db#http://dayabay:youknowit\@dayabay.ihep.ac.cn/svn/dybaux/!svn/bc/5070/c
```

The URL in the above example picks a particular revision of the catalog, to be loaded into temporary tables in the configured DB. This is equivalent to separately checking out dybaux to the desired revision and supplying the absolute path (or envvar prefixed) path in the config section.

### 21.4.3 N ways to set an envvar

#### bash

Pedestrian approach:

```
export DBCONF=tmp_offline_db
python myscript.py
```

Inline:

```
DBCONF=tmp_offline_db ipython
DBCONF=tmp_offline_db python myscript.py
DBCONF=tmp_offline_db nuwa.py ...
DBCONF=tmp_offline_db nosetests -v -s
DBCONF=tmp_offline_db ./dybinst trunk tests dbvalidate
DBCONF=tmp_offline_db ./dybinst trunk tests
DBCONF=tmp_offline_db ./dybinst trunk tests db_conditional
```

#### Inferior shells such as tcsh/csh

```
setenv DBCONF tmp_offline_db
python myscript.py
```

#### python

##### Convenient but Dangerous

The easily overridden `os.environ.setdefault` technique is not appropriate for scripts that write to Databases, but it is the recommended approach for readonly test scripts

```
import os
os.environ['DBCONF'] = "tmp_offline_db"

import os
os.environ.update( DBCONF="tmp_offline_db" )

import os
os.environ.setdefault( 'DBCONF', "tmp_offline_db" )
```

Question : what is the below going to return ?

```
export DBCONF=offline_db
python -c "import os ; os.environ.setdefault('DBCONF','tmp_offline_db') ; print os.environ['DBCONF']"
```

Using the easily overridden approach allows convenient testing against whatever Database or cascade:

```
DBCONF=tmp_offline_db:offline_db ./dybinst trunk tests dybdbi
```

**Warning:** tests that operate beneath DBI, eg `DbiValidate` which connects with `MySQL-python`, have not yet been modified to work in cascade.

## 21.4.4 Background Information

### What is a mysql dump file ?

A text serialisation of a MySQL database that contains the SQL commands necessary to recreate the table structure and content. They are complex and not well suited to human consumption.

### What is a DBI ascii catalog ?

DBI ascii catalogs are a serialization of database tables composed of a directory structure containing `.csv` files and `.cat` files to link them together:

```
/path/to/<catname>/
  <catname>.cat
  CalibFeeSpec/
    CalibFeeSpec.csv
    CalibFeeSpecVld.csv
  CalibPmtSpec/
    CalibPmtSpec.csv
    CalibPmtSpecVld.csv
  ...
  LOCALSEQNO/
    LOCALSEQNO.csv
```

DBI ascii catalogs have several advantages over `mysqldump` (`.sql`) files:

1. effectively native DBI format that can be used in ascii cascades allowing previewing of future database before real updates are made
2. very simple/easily parsable `.csv` that can be read by multiple tools
3. very simple diffs (DBI updates should be contiguous additional lines), unlike `mysqldump`, this means efficient storage in SVN
4. no-variants/options that change the format (unlike `mysqldump`)
5. no changes between versions of mysql

`Mysqldump` serialization has the advantage of being easily usable remotely.

## 21.4.5 Hands-On Exercise 1 : Troubleshooting DB connection configuration

- Check with `mysql` client
- Check with `db.py`
- Check with DBI
- DBI error when `DBCONF` not defined

### DIY: Configuration Setup and Troubleshooting

Create your `~/my.cnf` with 2 sections: `offline_db` and `client`, work through the below steps to verify your config. **Everyone** can do this, no extra permissions required.

**Warning:** Protect `~/my.cnf` with `chmod go-rwx` and **never** commit it into a repository

Approaches to isolating connection problems.

### Check with `mysql` client

Verify that the `mysql` client can connect and check you are talking to the expected DB:

```
echo status | mysql  ## only the client section of the config
```

### Check with `db.py`

Verify that `db.py` (a sibling of `nuwa.py`) can connect using the `client` section

`db.py` client check

```
dbconf : reading config from section "client" obtained from ['/etc/my.cnf', '/home/blyth/my.cnf']
{'VERSION()': '4.1.22-log', 'CURRENT_USER()': 'root@belle7.nuu.edu.tw', 'DATABASE()': 'offline_db_2'}
```

Verify that `db.py` can connect using other sections of the config:

`db.py` `offline_db` check

```
dbconf : reading config from section "offline_db" obtained from ['/etc/my.cnf', '/home/blyth/my.cnf']
{'VERSION()': '5.0.45-community-log', 'CURRENT_USER()': 'dayabay@%', 'DATABASE()': 'offline_db', 'O'}
```

### DIY: Determine row counts for all tables

Use another `db.py` command: `count`, also check `db.py --help` or `oum:api/db/`

### Check with DBI

Verify that DBI (and `DybDbi`) can connect. Do not be concerned regarding the Closed status mentioned in the output, the connection is opened when needed:

```
DBCONF=client python -c "from DybDbi import gDbi ; gDbi.Status() "
DybDbi ctor
DybDbi activating DbitableProxyRegistry
Using DBConf.Export to prime environment with : from DybPython import DBConf ; DBConf.Export('cli
dbconf : reading config from section "client" obtained from ['/etc/my.cnf', '/home/blyth/my.cnf']
dbconf:export_to_env from /etc/my.cnf:$SITEROOT/../../my.cnf:~/my.cnf section client
```

```
Successfully opened connection to: mysql://cms01.phys.ntu.edu.tw/offline_db_20110103
This client, and MySQL server (MySQL 4.1.22-log) does support prepared statements.
DbiCascader Status:-
Status    URL
```

```
Closed          0 mysql://cms01.phys.ntu.edu.tw/offline_db_20110103
```

Similarly test other sections of the config with:

```
DBCONF=offline_db python -c "from DybDbi import gDbi ; gDbi.Status() "
```

### DBI error when DBCONF not defined

To connect to a database with DBI (and thus DybDbi) requires the `DBCONF` envvar to be defined. If it is not defined or is invalid you will see an abort with error message.

```
( unset DBCONF ; python -c "from DybDbi import gDbi ; gDbi.Status() " ; )
DybDbi activating DbiTableProxyRegistry
Cannot open Database cascade as DBCONF envvar is not defined :
search for "DBCONF" in the Offline User Manual
ABORTING
```

## 21.4.6 Hands On Exercise 2 : Interactive DybDBI

- Get into ipython
- Interactively verify connection
- Interactive Exploration with ipython TAB completion
- DybDbi with some magic

### Get into ipython

Get into NuWa environment and fire up ipython with `DBCONF` defined, with bash:

```
DBCONF=offline_db ipython
```

with (t)csh:

```
setenv DBCONF "offline_db"
ipython
```

### Interactively verify connection

Duplicate the below to verify a DB connection:

```
In [1]: from DybDbi import gDbi
Warning in <TEnvRec::ChangeValue>: duplicate entry <Library.vector<short>=vector.dll> for level 0; i
Warning in <TEnvRec::ChangeValue>: duplicate entry <Library.vector<unsigned-int>=vector.dll> for leve
(Bool_t)1
DybDbi ctor

In [2]: gDbi.Status()
```

```

DybDbi activating DbiParamRegistry
Using DBConf.Export to prime environment with : from DybPython import DBConf ; DBConf.Export('offline
dbconf:export_to_env from $SITEROOT/../../my.cnf:~/my.cnf section offline_db
Successfully opened connection to: mysql://dybdb2.ihep.ac.cn/offline_db
This client, and MySQL server (MySQL 5.0.45-community) does support prepared statements.
DbiCascader Status:-
Status  URL

Closed          0 mysql://dybdb2.ihep.ac.cn/offline_db

DbiCascader Status:-
Status  URL

Closed          0 mysql://dybdb2.ihep.ac.cn/offline_db

```

## Interactive Exploration with `ipython` TAB completion

Use `ipython` tab completion to interactively explore:

```

In [3]: gDbi.<TAB>
gDbi.ClearRollbackDates  gDbi.IsA                gDbi.__class__          gDbi.__ge__
gDbi.ConfigRollback      gDbi.IsActive           gDbi.__delattr__       gDbi.__getattr__
gDbi.GetCascader          gDbi.MakeTimeStamp      gDbi.__dict__          gDbi.__gt__
gDbi.GetOutputLevel      gDbi.SetOutputLevel     gDbi.__doc__           gDbi.__hash__
gDbi.GetRegistry         gDbi.ShowMembers        gDbi.__eq__            gDbi.__init__
gDbi.Instance

```

The name of a object followed by <RETURN> outputs the repr (representation) of the object:

### DIY: call some methods

For example on the cascader instance: `len(gDbi.cascader)` or `gDbi.cascader[0]`

```

In [3]: gDbi.cascader
Out[3]:
DbiCascader numdb 1 authorisingdbno -1 (1st with GLOBALSEQNO)
Closed      offline_db          #0 tmp          closed      mysql://dybdb2.ihep.ac.cn/offline_db

```

```

In [4]: gDbi.cascader.__class__
Out[4]: <class 'DybDbi.DbiCascader'>

```

```

In [5]: gDbi.cascader.<TAB>
gDbi.cascader.AllocateSeqNo      gDbi.cascader.GetURL          gDbi.cascader.__delattr__
gDbi.cascader.CreateStatement    gDbi.cascader.HoldConnections gDbi.cascader.__dict__
gDbi.cascader.CreateTemporaryTable gDbi.cascader.IsA            gDbi.cascader.__doc__
gDbi.cascader.GetAuthorisingDbNo  gDbi.cascader.IsTemporaryTable gDbi.cascader.__eq__
gDbi.cascader.GetConnection       gDbi.cascader.ReleaseConnections gDbi.cascader.__format__
gDbi.cascader.GetDbName           gDbi.cascader.SetAuthorisingEntry gDbi.cascader.__ge__
gDbi.cascader.GetDbNo             gDbi.cascader.SetPermanent     gDbi.cascader.__getattr__
gDbi.cascader.GetNumDb            gDbi.cascader.ShowMembers      gDbi.cascader.__getitem__
gDbi.cascader.GetStatus           gDbi.cascader.TableExists      gDbi.cascader.__gt__
gDbi.cascader.GetStatusAsString   gDbi.cascader.__assign__       gDbi.cascader.__hash__
gDbi.cascader.GetTableDbNo        gDbi.cascader.__class__        gDbi.cascader.__init__

```

DybDbi with some magic

**DIY: taste some ipython magic**

enter ? or ?? after classnames eg DybDbi.DBConf?

Explore what DybDbi provides:

In [1]: import DybDbi

In [2]: DybDbi.<TAB>

Display all 117 possibilities? (y or n)

DybDbi.CSV	DybDbi.DbiStatement__del__	DybDbi.GDbiLogEntry	DybDbi.
DybDbi.Context	DybDbi.DbiTableProxy	DybDbi.GDcsAdTemp	DybDbi.
DybDbi.ContextRange	DybDbi.DbiTableProxyRegistry	DybDbi.GDcsPmtHv	DybDbi.
DybDbi.Ctx	DybDbi.DbiValRecSet	DybDbi.GFeeCableMap	DybDbi.
DybDbi.DBCas	DybDbi.DbiValidityRec	DybDbi.GNAMES	DybDbi.
DybDbi.DBConf	DybDbi.Detector	DybDbi.GPhysAd	DybDbi.
DybDbi.Dbi	DybDbi.DetectorId	DybDbi.GSimPmtSpec	DybDbi.
DybDbi.DbiCache	DybDbi.DetectorSensor	DybDbi.LOG	DybDbi.
DybDbi.DbiCascader	DybDbi.DybDbi	DybDbi.Level	DybDbi.
DybDbi.DbiCascader__check	DybDbi.DybDbi__comment	DybDbi.MYSQLDUMP	DybDbi.
DybDbi.DbiCascader__getitem__	DybDbi.GCalibFeeSpec	DybDbi.Mapper	DybDbi.
DybDbi.DbiCascader__repr__	DybDbi.GCalibPmtSpec	DybDbi.NullHandler	DybDbi.
DybDbi.DbiConnection	DybDbi.GDagCalibRunInfo	DybDbi.POST	DybDbi.
DybDbi.DbiConnection__repr__	DybDbi.GDagRawDataFileInfo	DybDbi.ServiceMode	DybDbi.
DybDbi.DbiCtx	DybDbi.GDagRunInfo	DybDbi.SimFlag	DybDbi.
DybDbi.DbiCtx__call__	DybDbi.GDbiConfigSet	DybDbi.Site	DybDbi.
DybDbi.DbiCtx__repr__	DybDbi.GDbiDemoData1	DybDbi.Source	DybDbi.
DybDbi.DbiSqlContext	DybDbi.GDbiDemoData2	DybDbi.TList	DybDbi.
DybDbi.DbiSqlValPacket	DybDbi.GDbiDemoData3	DybDbi.TMap	DybDbi.
DybDbi.DbiStatement	DybDbi.GDbiDemoData4	DybDbi.TObject	DybDbi.

## 21.5 DB Table Updating Workflow

- Objectives
- Workflow Outline
- Rationale for this workflow
- Planning Update Size and Frequency
- workflow steps in detail
  - Copy `offline_db` to `tmp_offline_db`
  - Perform updates and validation on `tmp_offline_db`
  - Early Validations
  - Communicate updates via SVN repository
  - Decoupled Updating Workflow
  - Serialized Updating
  - How the SVN ascii catalog is primed
  - Annotating Updates
  - Pre-commit enforced validation : DBI Gatekeeper
  - Database Managers Propagate updates from `dybaux` into `offline_db`
  - Propagation of multiple commits with `dbaux.py`
  - Post-propagation cross check
  - Quick Validations
- Exceptional Operating Procedures for Major Changes
- Hands On Exercise 3 : Copy Offline DB
- Nosetests of workflow steps
- Development History of Workflow

### 21.5.1 Objectives

The mission critical nature of calibration parameters requires DB updating procedures to be:

1. highly controlled
2. carefully recorded
3. easily reproducible

Also DB updating procedures should be:

1. straightforward and quick

Suggestions for amendments to the workflow steps presented should be made in [dybsvn:ticket:607](#).

### 21.5.2 Workflow Outline

1. Calibration expert obtains values intended to be inserted into `offline_db`
2. Calibration expert makes a temporary copy of the central DB `tmp_offline_db`
3. Calibration expert inserts values into his/her copy of the central DB `tmp_offline_db`
4. Calibration expert validates new values inserted into their `tmp_offline_db`
5. Calibration expert contacts DB Managers (Liang/Simon B) and request update propagated from `tmp_offline_db` into `offline_db`
6. Following successful validation DB Managers propagate updates into Master `offline_db`
7. Master `offline_db` DB is propagated to slaves

### 21.5.3 Rationale for this workflow

Why such caution ? Why not just write directing into `offline_db` ?

1. Avoid inconsistent/conflicting updates
2. Avoid inconsistencies as a result of mysql slave propagation (it may be necessary to briefly halt propagation while updates are made)

### 21.5.4 Planning Update Size and Frequency

Points to bear in mind when planning update size and frequency:

1. not too big to cause handling problems, aim to not exceed ~few MB of csv change (guesswork yet to be informed by experience)
2. not too small, if that necessitates repetition - to avoid manual labor and delays
3. each dybaux commit is loaded into `offline_db` with a single `INSERTDATE`, that means that you cannot distinguish via `ROLLBACK` within a single *commit*

### 21.5.5 workflow steps in detail

#### Section Names or Database names

This documentation refers to databases by their configuration file section names such as `tmp_offline_db` rather than by the actual database name (eg `tmp_username_offline_db`), as this parallels the approach taken by the tools: `db.py` and `DBI`.

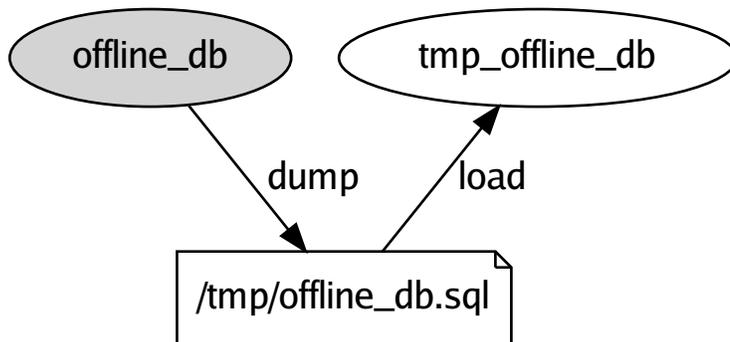
See *Configuring DB Access* for details of configuration file `~/ .my.cnf` creation and troubleshooting.

#### Copy `offline_db` to `tmp_offline_db`

#### Working with a copy

facilitates rapid development without concern for causing damage by providing the option to start over as many times as needed.

The `db.py` script (a sibling of `nuwa.py`) is used to perform the copy, by loading and dumping tables.



Make the copy in 2 steps

```
db.py <sectname> <cmd> ...
```

the first argument references the section in the configuration file, details of `db.py` options and other commands are available at [DybPython.db](http://DybPython.db)

1. Dump selection of `offline_db` tables to be updated into mysqldump file using the `-t/--tselect` option with a comma delimited list of payload table names

```
db.py -t CalibPmtSpec offline_db dump offline_db.sql
db.py -t CableMap,HardwareID offline_db dump offline_db.sql
```

1. Load the mysqldump file into temporary database copy:

```
db.py tmp_offline_db load offline_db.sql
```

Note that no special privileges are needed in `offline_db` but `DATABASE DROP` and `DATABASE CREATE` privileges are needed in `tmp_offline_db`. Also the `tmp_offline_db` does **not** need to be local. Example of `tmp_offline_db` content after the **load** containing just the tables to be updated (and a partial `LOCALSEQNO` table):

```
mysql> show tables ;
+-----+
| Tables_in_tmp_offline_db |
+-----+
| CableMap                  |
| CableMapVld               |
| HardwareID                |
| HardwareIDVld             |
| LOCALSEQNO                |
+-----+
5 rows in set (0.00 sec)

mysql> select * from LOCALSEQNO ;
+-----+
```

```

| TABLENAME | LASTUSEDSEQNO |
+-----+-----+
| *          |          0    |
| HardwareID |          386  |
| CableMap   |          475  |
+-----+-----+
3 rows in set (0.00 sec)

```

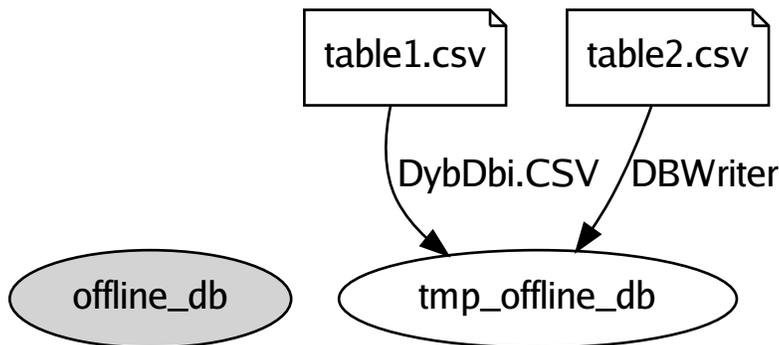
For readonly access to other tables such as DaqRunInfo use DBI cascades configured with a colon delimited DBCONF.

### Perform updates and validation on tmp\_offline\_db

**Warning:** do not attempt to use raw SQL or hand edited .csv

DB Writing must use DBI eg

1. service approach dybgaudi:Database/DBWriter
2. python script using DybDbi, see *DB Table Writing*



#### flexibility unwise

Easily overridden `os.environ.setdefault` not appropriate for Writers see *N ways to set an envvar*

Configure writing scripts with:

```

import os
os.environ['DBCONF'] = 'tmp_offline_db'

```

Or invoke services:

```

DBCONF=tmp_offline_db nuwa.py ...

```

## Early Validations

**Warning:** `dbupdatecheck` currently only contains `dbvalidate`, other packages with tests to be run before and after updates **need to be added**

The standard set of validation tests can be run by Table managers prior to checkin to SVN with:

```
DBCNF=tmp_offline_db:offline_db ./dybinst trunk tests dbupdatecheck
```

`dbupdatecheck` is a alias for a list of packages defined in `installation:dybinst/scripts/dybinst-common.sh`

After table managers commit the candidate update to `dybaux` anyone (with permissions in an available DB) can validate, using:

```
cd ; svn co http://dayabay.ihep.ac.cn/svn/dybaux/catalog/tmp_offline_db
svn up ~/tmp_offline_db
DBCNF=tmp_offline_db_ascii:offline_db ./dybinst trunk tests dbupdatecheck
```

Configuration details in *Configuring access to ascii catalog*

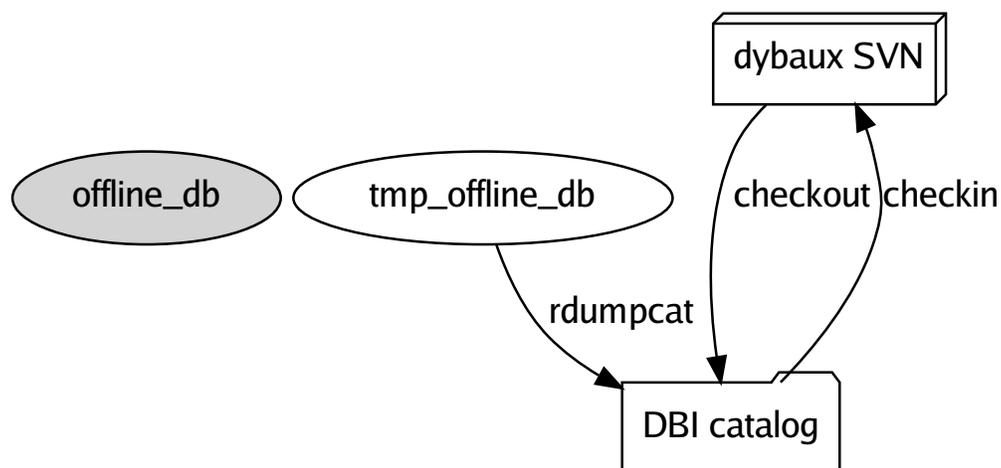
This allows any `os.environ.setdefault nosetest` to be run against the candidate update.

*DB Validation and Testing* includes ideas on update targeted tests.

## Communicate updates via SVN repository

Using an SVN repository to funnel updates has advantages:

1. familiar system for storing the history of updates
2. Easy communication of changes
3. Trac web interface with timeline, presenting the history



Checkout the *tmp\_offline\_db* DBI ascii catalog from SVN repository:

```
mkdir -p ~/dybaux/catalog ; cd ~/dybaux/catalog
svn co http://dayabay.ihep.ac.cn/svn/dybaux/catalog/tmp_offline_db
```

Use *rdumpcat* to export updated database as DBI catalog on top of the SVN checkout, allowing the nature of the update to be checked with *svn diff* etc.:

```
db.py tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
svn status ~/dybaux/catalog/tmp_offline_db
svn diff ~/dybaux/catalog/tmp_offline_db
```

Try to commit the update to SVN:

```
svn ci ~/dybaux/catalog/tmp_offline_db \
-m "New tables for CableSvc see dybsvn:source:dybgaudi/trunk/DybSvc/DybMetaDataSvc/src/DybCableSvc.t
```

For the rationale behind the link see *Annotating Updates*, note that:

1. the link path must start `dybsvn:source:dybgaudi/trunk`
2. when multiple links are included only the first is checked
3. use the Trac search box to check links, without the `dybsvn:` when using the `dybsvn` instance or with it when using `dybaux`

## Decoupled Updating Workflow

### Workflow Commands Essentially Unchanged

Primary difference is the initial **dump** which should now select only the tables being updated using `-t/--tselect` option with a comma delimited list of payload table names.

Features:

1. `db.py` adopts the `-d/--decoupled` option as default from `dybsvn:r14289`
2. `tmp_offline_db` contains only the tables being updated + a partial `LOCALSEQNO` metadata table.
3. partial `LOCALSEQNO` table is merged with the shared *real* one when doing **rdumpcat** into the `dybaux` catalog
4. (in principal) removes updating bottleneck by allowing parallel updating **assuming no cross table dependencies**

Tables are selected on the initial **dump** and subsequent **load** and **rdumpcat** operate on those selected tables:

```
db.py -t CableMap,HardwareID offline_db dump ~/offline_db.sql
db.py tmp_offline_db load ~/offline_db.sql ## clobber
db.py tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
```

Note that the **rdumpcat** must be into into a pre-existing catalog such as `~/dybaux/catalog/tmp_offline_db` as decoupled tables are not viable on their own. Features of decoupled **rdumpcat**:

1. `LOCALSEQNO` entries are merged into the pre-existing `LOCALSEQNO.csv`
2. payload and validity table `.csv` changes must add to existing ones in the catalog
3. no catalog `.cat` file is written, permissible updates cannot change the catalog file

Before doing the **rdumpcat** it is good practice to check `LOCALSEQNO` in `tmp_offline_db` and in the catalog, and be aware of the changes:

```

cat ~/dybaux/catalog/tmp_offline_db/LOCALSEQNO/LOCALSEQNO.csv
TABLENAME char(64),LASTUSEDSEQNO int(11),PRIMARY KEY (TABLENAME)
"*,0
"CableMap",474
"CalibFeeSpec",113
"CalibPmtHighGain",6
"CalibPmtPedBias",1
"CalibPmtSpec",96
"CoordinateAd",1
"CoordinateReactor",1
"FeeCableMap",3
"HardwareID",386
"Reactor",372

```

## Serialized Updating

### Decoupled Updating Is Now Default

Prior to introduction of decoupled updating, updates had to be coordinated due to the shared **LOCALSEQNO** table. This section can be removed once decoupled operation is proven.

**Note:** you need to login to dybaux with your dybsvn identity in order to see commits

In order to coordinate the serialized updating check the dybaux timeline <http://dayabay.ihep.ac.cn/tracs/dybaux/timeline> before making commits there. If you see a recent catalog commit that is not followed up a *fastforward OVERRIDE* commit by one of the DB managers then an update is queued up ahead of you in the final validation stage.

That means:

1. you will need to refresh(dump+load) your tmp\_offline\_db and rerun your updater script after the update ahead is completed (you will see the *fastforward OVERRIDE* commit on the timeline)
2. hold off making your dybaux catalog commit until the 1st step is done
3. continue testing in your tmp\_offline\_db, such that once you have a valid starting point your update is able to proceed smoothly and quickly and does not cause delays in the final validations stage

## How the SVN ascii catalog is primed

### ascii catalog is for communication

The catalog/tmp\_offline\_db exists to facilitate communication and checking of updates. It in no way detracts from the definitive nature of what is in offline\_db. Essentially it is a shared tmp\_offline\_db that may need re-priming following candidate updates for which problems are found at the last hurdle of DB Manager validations.

Direct approach, using **rdumpcat** from offline\_db into ascii catalog:

```

svn co http://dayabay.ihep.ac.cn/svn/dybaux/catalog          ## just empty tmp_offline_db created by zh
db.py offline_db rdumpcat ~/catalog/tmp_offline_db         ## dump non-scraped default subset of table

```

Check machinery and transfers (and prepare a local DB to work with as side effect) by going via local DB `tmp_offline_db`:

```
db.py offline_db dump ~/tmp_offline_db.sql
db.py tmp_offline_db load ~/tmp_offline_db.sql
db.py tmp_offline_db rdumpcat ~/tmp_offline_db_via_local
```

Compare the direct and `via_local` catalogs:

```
diff -r --brief ~/catalog/tmp_offline_db ~/tmp_offline_db_via_local | grep -v .svn
Only in /home/blyth/catalog/tmp_offline_db: tmp_offline_db.cat
Only in /home/blyth/tmp_offline_db_via_local: tmp_offline_db_via_local.cat
```

Add to repository, and commit with override:

```
svn add tmp_offline_db/*
svn status
svn ci -m "initial commit of ascii catalog prepared with {{{db.py offline_db rdumpcat ~/catalog/tmp_
```

### Annotating Updates

When making updates it is required that brief documentation is provided in a text file housed in `dybsvn`. Appropriate locations for the documentation are:

- package containing the code that prepares the update (this code must be kept in `dybsvn`, see *Rules for Code that writes to the Database*).
- package containing the service that uses the updated tables

Expected features for the annotation of an update:

1. brief summary of nature/motivation, a few lines only (refer to more detailed descriptions)
2. include date of update
3. refer to related docdb documents
4. refer to related database tables
5. refer to `dybsvn` packages updated, name revision numbers where appropriate
6. refer to related tickets

In order to associate the annotation with the `dybaux` commit of the candidate DB update, it is required that the commit message provides a revisioned Trac Link that points at the updated document containing the annotation.

In the above example, the revisioned Trac link points to a real example of an annotation document.

- <dybsvn:source:dybgaudi/trunk/DybSvc/DybMetaDataSvc/src/DybCableSvc.txt@12349>

### Pre-commit enforced validation : DBI Gatekeeper

The `dybaux` repository is configured to perform validations prior to allowing the commit to proceed. When commits are denied the validation error is returned. Validations are implemented in python module `DybPython.dbsvn`, currently:

1. Only expected tables are touched (LOCALSEQNO + DBI pair)
2. Only row additions are allowed, no deletions or alterations
3. Commit message includes valid revisioned `dybsvn` Trac Link, precisely identifying code/documentation for the update

Intended additions:

1. verify use of `versiondate=TimeStamp(0,0)` signaling overlay dates

Pre-commit validations must be quick and self-contained as cannot run tests on SVN server.

### Test Locally

Validations can be run locally using `DybPython.dbsvn` script.

## Database Managers Propagate updates from `dybaux` into `offline_db`

After re-running validations as described in *Early Validations* are found to be successful DB managers can perform updates first on their `tmp_offline_db` and then on the central `offline_db`.

Prepare a fresh `tmp_offline_db`:

```
db.py offline_db dump ~/offline_db.sql
db.py tmp_offline_db load ~/offline_db.sql
```

Get uptodate with `dybaux`:

```
mkdir ~/dybaux ; cd ~/dybaux ; svn co http://http://dayabay.ihep.ac.cn/svn/dybaux/catalog
svn up ~/dybaux/catalog
```

Use **rcmpcat** to see changed tables and added SEQNO in the `dybaux` ascii catalog relative to the DB `tmp_offline_db`:

```
db.py tmp_offline_db rcmpcat ~/dybaux/catalog/tmp_offline_db
```

Proceed to **rloadcat** into `tmp_offline_db`:

```
db.py tmp_offline_db rloadcat ~/dybaux/catalog/tmp_offline_db
db.py tmp_offline_db rcmpcat ~/dybaux/catalog/tmp_offline_db    ## should report no updates
```

Repeating **rloadcat** should detect no updates and do nothing. Note that the catalog working copy is changed by the **rloadcat** due to INSERTDATE fastforwarding ([dybsvn:ticket:844](#)), check with:

```
svn status ~/dybaux/catalog/tmp_offline_db    ## will show Vld table changes
```

Following the definitive **rloadcat** into `offline_db` the changed `*Vld.csv` should be committed into `dybaux` with a commit message including `OVERRIDE` (only admin users configured in the pre-commit hook can do this).

## Propagation of multiple commits with `dbaux.py`

When multiple commits need to be propagated the script `dbaux.py` should be used, it takes as arguments commit number ranges and internally invokes the `db.py` script described above.

Usage examples:

```
dbaux.py --help          ## for details on all options
dbaux.py -c -r --dbconf tmp_offline_db rloadcat 5036:5037 --logpath dbaux-rloadcat-5036-5037.log
```

**dbaux -r option resets working copy**

For reliable operation (avoiding svn merge/conflict difficulties) the `-r/--reset` option is used to force catalog working copy to be at clean revisions by deletion of any preexisting directories. A side effect of this is that, the working copy fast forward modifications are lost for all but the last commit propagated.

In order to make complete fastforward commits after using the `dbaux.py -r/--reset` it is necessary to do an `rdumpcat` to get all the fastforward changes first, for example with:

```
db.py offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
svn diff ~/dybaux/catalog/tmp_offline_db      ## INSERTDATE changes for all SEQNO added should be observed
svn ci -m "fastforward updates following offline_db rloadcat of r5036:r5037 OVERRIDE " ~/dybaux/catalog
```

**Post-propagation cross check**

Get uptodate with `dybaux` and `rdumpcat` from `offline_db` ontop of it:

```
svn up ~/dybaux/catalog/tmp_offline_db
db.py offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
svn status ~/dybaux/catalog/tmp_offline_db
```

The `svn status` is expected to return no output, indicating no differences. If differences are observed only in `*Vld.csv` tables `INSERTDATE` then Database managers omitted to commit the fastforwarded catalog.

**Quick Validations**

After getting into environment and directory of `dybgaudi:Database/DbiValidate`, run a collection of tests that traverse over all DBI tables, performing many queries:

```
DBCONF=offline_db nosetests -v

[blyth@cms01 DbiValidate]$ DBCONF=offline_db nosetests -v
test_dbi_tables.test_counts ... ok
test_dbi_tables.test_vld_description('CableMapVld', 'assert_fields') ... ok
...
DbiTimer:CableMap: Query done. 2592rows, 62.2Kb Cpu 0.1 , elapse 1.7
Caching new results: ResultKey: Table:CableMap row:GCableMap. 1 vrec (seqno;versiondate): 213;2011-
DbiTimer:CableMap: Query done. 1728rows, 41.5Kb Cpu 0.1 , elapse 1.7
ok
-----
Ran 159 tests in 685.069s          ## MUCH FASTER WHEN LOCAL TO DB
OK
```

**21.5.6 Exceptional Operating Procedures for Major Changes**

Major changes need to be discussed with Database Managers. As such changes will not pass the SOP validations, a modified **Exceptional operating procedure** is used:

1. Table experts develop **zero** argument scripts (which can internally invoke more flexible scripts and capture arguments used) using their `tmp_offline_db`
2. Table experts communicate update to be made via `dybsvn` revision and path of their scripts
3. DB experts use the script to create tables in their `tmp_offline_db` and perform override commit of new tables into **dybaux**

4. table experts check that the tables in **dybaux** match those from their `tmp_offline_db` (eg via `rdumpcat` onto the working copy)
5. DB experts proceed to load into `offline_db` once table experts have confirmed the change

The steps are mostly the same, but who does what is modified.

### 21.5.7 Hands On Exercise 3 : Copy Offline DB

**Warning:** This exercise requires write permissions into a `tmp_username_offline_db` database

DIY steps:

1. Configure a `tmp_offline_db` section of your config file
2. Use `db.py` to dump and load into your `tmp_username_offline_db` database : which corresponds to section `tmp_offline_db`
3. Use techniques from exercises 1 and 2 to compare row counts in `offline_db` and `tmp_offline_db`

### 21.5.8 Nostests of workflow steps

**Note:** these are tests of the workflow machinery, for other generic tests and more table specific validations see *DB Validation and Testing*

Nostests covering most of the steps of the workflow are available in `dybgaudi:DybPython/tests` in particular:

- `dybgaudi:DybPython/tests/test_dbsvn.py`
- `dybgaudi:DybPython/tests/test_dbops.py`
- `dybgaudi:DybPython/tests/test_write.py`
- `dybgaudi:DybPython/tests/test_write_cascade.py` **WARNINGS ON USAGE STILL APPLY**

To run these tests, get into the directory and environment of `dybgaudi:DybPython` then:

1. Examine what the tests are going to do
2. Review the configuration section names used in the tests (typically `tmp_offline_db` and `offline_db` ). Find these by looking for any `DBCONE=sectname` and first arguments to the `db.py` script
3. Review the corresponding sections of your configuration `~/ .my.cnf` ensuring that **you are talking to the intended DB with identities holding appropriate permissions**
4. You may need to add/rename some sections of your configuration file if they are not present
5. Invoke the tests from the package directory, **not** the tests directory with the below commands

```
nostests -v -s tests/test_dbops.py
nostests -v -s tests/test_dbsvn.py
nostests --help    ## for explanations of the options
```

### 21.5.9 Development History of Workflow

The general approach was first expounded in [doc:5646](#), but has subsequently been improved substantially following feedback from Jijie, Craig and Brett. The changes avoid some painful aspects of the initial suggestion.

1. Remove **local** restriction on the mysql server, enabling your NuWa installation and mysqld server to be on separate machines
2. Eliminate need for DB Managers to keep *dybaux* DBI catalog uptodate, as Table managers now start by copying the actual *offline\_db*

## 21.6 DB Table Writing

### live ipython sessions

The below ipython output was generated when this documentation was built using *live ipython session with ipython directive*. So if you are using a revision close to that of these docs, you can expect to see almost the same output.

- Using DybDbi to write to tmp\_offline\_db
  - Configure Target DB
  - CSV handling
  - Map CSV fieldnames to DBI attributes
  - Create Dbewriter<T> and set ContextRange
  - Convert CSV rows and write to DB
  - Command line and filename Parsing
- Hands On Exercise 4 : Write \$DBWRITERROOT/share/DYB\_MC\_AD1.txt into CalibPmtSpec
- Assigning Applicability of Constants
  - Context Range
  - Choosing TIMEEND
  - Determine run start time from a run number
  - Overlay Versioning Demonstration
- Many more examples of DB writing with DybDbi
- Using DBewriter to write to tmp\_offline\_db

**Warning:** Always check which Database you are connected to

Before doing any DB operations, avoid accidents/confusion by using `status` in mysql shell or `gDbi.Status()` in ipython or checking `DBCONF` settings used in scripts and the corresponding configuration in your configuration file `~/my.cnf`, see *Configuring DB Access* for details.

### 21.6.1 Using DybDbi to write to tmp\_offline\_db

```
from DybDbi import ...
```

DybDbi works by wrapping PyROOT C++ proxy classes with additional functionality, to benefit access classes through `from Dybdbi import..`

DybDbi enables usage of DBI from python in a simple way

## Configure Target DB

**Warning:** do not use easily overridden config such as `os.environ.setdefault`

```
In [27]: import os
```

```
In [28]: os.environ['DBCONF'] = "tmp_offline_db"
```

## CSV handling

`DybDbi.CSV` provides CSV reading/validation facilities, invalid `.csv` files throw exceptions

```
In [1]: from DybDbi import CSV
```

```
In [2]: src = CSV( "$DBWRITERROOT/share/DYB_MC_AD1.txt" )
```

```
In [3]: src.read()
```

```
In [4]: print len(src)
```

```
In [5]: print src[0]    ## first source csv row, note the srcline
```

```
In [5]: print src[-1]  ## last source csv row, note the srcline
```

```
In [6]: print src.fieldnames    ## fields
```

## Map CSV fieldnames to DBI attributes

`DybDbi.Mapper` provides CSV fieldname to DBI attribute name mappings, and type conversions (CSV returns everything as a string)

Using the same CSV fieldnames as DBI attributes may allow auto mapping, otherwise manual mappings must be set.

### Generic Advantage

Each `genDbi/DybDbi` generated class knows the full specification of itself and the corresponding database table, see *DB Table Creation*, thus given the mapping from CSV fieldname to DBI attribute the appropriate type conversions are used.

An incomplete mapping throws exceptions:

```
In [12]: from DybDbi import Mapper, GCalibPmtSpec
```

```
In [13]: mpr = Mapper( GCalibPmtSpec, src.fieldnames )
```

After interactively adding manual mappings, succeed to create the the mapper:

```
In [16]: mpr = Mapper( GCalibPmtSpec, src.fieldnames , afterPulse='AfterPulseProb', sigmaSpe='Sigma
```

```
In [17]: print mpr
```

All elements from a `.csv` are strings. Note the fieldname and type conversion after the `mpr` instance operates on one `src dict` item.

```
In [11]: print src[0]
```

```
In [12]: print mpr(src[0])
```

Apply the `mpr` instance over all items in the `src`:

```
In [13]: dst = map(mpr, src )
```

```
In [14]: len(dst)
```

```
In [16]: print dst[0]
```

### Create `DbiWriter<T>` and set `ContextRange`

```
In [18]: from DybDbi import Site, SimFlag, TimeStamp, ContextRange
```

```
In [19]: wrt = GCalibPmtSpec.Wrt()
```

```
In [20]: cr = ContextRange( Site.kAll, SimFlag.kData|SimFlag.kMC , TimeStamp.GetBOT() ,TimeStamp.Get
```

```
In [21]: wrt.ctx( contextrange=cr, dbno=0, versiondate=TimeStamp(0,0), subsite=0 , task=7, logcomment
```

Notes:

1. `dbno=0`, selects the slot in the DB cascade to write to
2. `logcomment="msg"` are currently ignored, as DBI is not operating in an **Authorising DB** manner with a `GLOBALSEQNO` table, [dybsvn:ticket:803](#) seeks to assess the implications of migrating to **Authorising DB** usage
3. `versiondate=TimeStamp(0,0)` switches on overlay date validity handling

---

### Todo

enforce usage of overlay date in pre-commit hook

---

### Convert CSV rows and write to DB

```
In [23]: for r in map(mpr,src):    ## __call__ method of mpr invoked on all src items
....:     instance = GCalibPmtSpec.Create( *\r )
....:     wrt.Write( instance )
```

Crucial last step that writes the DBI row instances from memory to the DB:

```
In [25]: assert wrt.Close()    ## DB is accessed here
DbiWrt<GCalibPmtSpec>::Close
```

(this step is skipped on building these docs)

### Command line and filename Parsing

Using some simple python techniques for commandline parsing and filename parsing can avoid the anti-pattern of duplicating a writing script and making small changes.

See the examples:

- dybgaudi:Database/DybDbi/examples/GCalibPmtHighGain\_.py
- dybgaudi:Database/DybDbi/examples/cnf.py

A simple regular expression is used to match the name of a .csv file, for example :

```
In [1]: import re

In [2]: ptt = "^(?P<site>All|DayaBay|Far|LingAo|Mid|SAB)_(?P<subsite>AD1|AD2|AD3|AD4|All|IWS|OWS|RPC

In [3]: ptn = re.compile( ptt )

In [4]: match = ptn.match( "SAB_AD2_Data.csv" )

In [5]: print match.groupdict()
{'subsite': 'AD2', 'simflag': 'Data', 'site': 'SAB'}
```

The script then converts these to enum values using the enum FromString functions.

Such an approach has several advantages:

1. standardized file names
2. reduced number of parameters/options on commandline
3. eliminates pointlessly duplicated code

## 21.6.2 Hands On Exercise 4 : Write \$DBWRITERROOT/share/DYB\_MC\_AD1.txt into CalibPmtSpec

**Warning:** This exercise requires write permissions into a tmp\_username\_offline\_db database, and a recent NuWa installation

DIY steps:

1. Use interactive ipython to perform the steps of the previous section
2. Remember to read the API help as you go along eg: CSV? Mapper?
3. Use mysql client to examine your additions to the copied DB

**Note:** Very little added code is required to complete this (hint: manual field name mappings), extra points for using a realistic contextrange

Hint to help with field mapping, genDbi classes know their .spec so ask the class with eg SpecMap() :

```
In [12]: cls.Spec<TAB>
cls.SpecKeys  cls.SpecList  cls.SpecMap

In [12]: cls.SpecMap()
Out[12]: <ROOT.TMap object ("TMap") at 0xb068dc0>

In [13]: cls.SpecMap().asdod()
Out[13]:
{'AfterPulseProb': {'code2db': '',
                    'codetype': 'double',
                    'dbtype': 'float',
                    'description': 'Probability of afterpulsing',
                    'legacy': 'PMTAFTERPULSE',
```

```

        'memb': 'm_afterPulseProb',
        'name': 'AfterPulseProb'},
'DarkRate': {'code2db': '',
             'codetype': 'double',
             'dbtype': 'float',
             'description': 'Dark Rate',
             'legacy': 'PMTDARKRATE',
             'memb': 'm_darkRate',
             'name': 'DarkRate'},
...

```

```

In [14]: cls.SpecKeys().aslist()
Out[14]:
['PmtId',
 'Describ',
 'Status',
 'SpeHigh',
 'SigmaSpeHigh',
 'SpeLow',
 'TimeOffset',
 'TimeSpread',
 'Efficiency',
 'PrePulseProb',
 'AfterPulseProb',
 'DarkRate']

```

### 21.6.3 Assigning Applicability of Constants

The arguments to the writer establish the range of applicability of the constants to be written.

```

from DybDbi import GCalibPmtSpec as cls
from DybDbi import Site, SimFlag, TimeStamp, ContextRange
wrt = cls.Wrt()
cr = ContextRange( Site.kAll, SimFlag.kData|SimFlag.kMC , TimeStamp.GetBOT() , TimeStamp.GetEOT() )
wrt.ctx( contextrange=cr, dbno=0, versiondate=TimeStamp(0,0), subsite=0 , task=0, logcomment="DybDbi

```

The crucial line:

```

wrt.ctx( contextrange=cr, dbno=0, versiondate=TimeStamp(0,0), subsite=0 , task=7, logcomment="DybDbi

```

is python shorthand (via `DbiCtx.__call__()` and setter properties) for defining the attributes of the C++ class `DybDbi.DbiCtx` defined in `dybgaudi:Database/DybDbi/DybDbi/DbiCtx.h`. The choice of attributes determines which underlying `DbiWriter<GTableName>` ctor is invoked.

DbiCtx attribute	notes
contextrange	object described below
dbno	slot in the cascade to write to, usually should be 0
versiondate	<b>always</b> use <code>TimeStamp(0,0)</code> to signify overlay versioning
subsite	<code>Dbi::SubSite</code> enum integer
task	<code>Dbi::Task</code> enum integer
logcomment	description of update, see <a href="#">dybsvn:ticket:803</a>

**Overlay Versioning**

This is a date based versioning scheme that automatically distinguishes validity entries which have the same context range by offsetting of versiondate by minute increments. This scheme allows prior erroneous writes to be overridden. Discussed in [dybsvn:ticket:611](#). Details on this including a demonstration below [Overlay Versioning Demonstration](#)

The `Dbi::enums` are defined in `databaseinterface:Dbi.h`

**Todo**

try changing implementation of enums to make them usable from python

**Context Range**

Example of instantiation from python:

```
from DybDbi import Site, SimFlag, TimeStamp, ContextRange
cr = ContextRange( Site.kAll, SimFlag.kData|SimFlag.kMC , TimeStamp.GetBOT() , TimeStamp.GetEOT() )
```

**Warning:** All times stored in the offline database must be in UTC, this includes validity range times

For the details on these classes see the API docs [DybDbi.ContextRange](#), [DybDbi.TimeStamp](#)

argument	notes
siteMask	An OR of site enum integers <a href="#">conventions:Site.h</a>
simMask	An OR of simflag enum integers <a href="#">conventions:SimFlag.h</a>
tstart	Start of validity, possibly corresponding to start of run time
tend	End of validity, this will very often be <code>TimeStamp::GetEOT()</code> signifying a far future time

**Choosing TIMEEND**

Recommendations :

1. when a definite end time is known use that
2. use `TimeStamp.GetEOT()` when the end time is not known
3. if constants need *decommissioning* this can be done with payload-less writes (in consultation with DB managers)

Do not adopt a policy of blindly using EOT, use the contextrange that best expresses the nature of that set of constants. Note that *decommissioning* allows particular context ranges to yield no constants. This is preferable to inappropriate constants as it is trivial to handle in services.

Things not to do:

1. use random far future times, instead standardize on `TimeStamp.GetEOT()`

**Determine run start time from a run number**

First approach that brings the full table into memory:

```
runNo = 5000
from DybDbi import GDaqRunInfo
rpt = GDaqRunInfo.Rpt()
rpt.ctx( sqlcontext="1=1" , task=-1 , subsite=-1 ) ## wideopen validity query
row = rpt.FirstRowWithIntValueForKey( "RunNo" , runNo )
vrec = rpt.GetValidityRec( row )
print vrec.seqno, vrec.contextrange.timestart, vrec.contextrange.timeend
```

Second approach that brings in only a single row into memory:

```
runNo = 5000
from DybDbi import GDaqRunInfo
rpt = GDaqRunInfo.Rpt()
rpt.ctx( sqlcontext="1=1", datasql="runNo = %s" % runNo , task=-1, subsite=-1 )
assert len(rpt) == 1 , "should only be a single entry for the runNo %s " % runNo
row = rpt[0]
vrec = rpt.GetValidityRec( row )
print vrec.seqno, vrec.contextrange.timestart, vrec.contextrange.timeend
```

A discussion of the relative merits of these approaches is in `dybgaudi:Database/DybDbi/tests/test_find_vrec.py`

### Overlay Versioning Demonstration

The tests in `dybgaudi:Database/DybDbi/tests/test_demo_overlay.py` demonstrate overlay versioning in action:

**test\_write\_ugly** write a mixture of good and bad constants via `GDemo` class into table `Demo`

**test\_read\_ugly** verify read back what was written

**test\_write\_good** an overriding context to correct some the bad constants

**test\_read\_good** verify can read back the overriding constants

**test\_read\_allgood** verify can read back all good constants

By virtue of using overlay versioning, as enabled with `versiondate` in the write context:

```
versiondate=TimeStamp(0,0)
```

Synthetic `VERSIONDATE` are used which coincide with `TIMESTART` unless there is data present already, in which case one minute offsets are made in order to override prior writes. In the validity table, there is a one minute `VERSIONDATE` offset for `SEQNO = 11`:

```
mysql> select * from DemoVld ;
```

SEQNO	TIMESTART	TIMEEND	SITEMASK	SIMMASK	SUBSITE	TASK	AGGREGATE
1	2010-01-01 00:00:00	2010-01-11 00:00:00	127	1	0	0	
2	2010-01-11 00:00:00	2010-01-21 00:00:00	127	1	0	0	
3	2010-01-21 00:00:00	2010-01-31 00:00:00	127	1	0	0	
4	2010-01-31 00:00:00	2010-02-10 00:00:00	127	1	0	0	
5	2010-02-10 00:00:00	2010-02-20 00:00:00	127	1	0	0	
** 6	2010-02-20 00:00:00	2010-03-02 00:00:00	127	1	0	0	
7	2010-03-02 00:00:00	2010-03-12 00:00:00	127	1	0	0	
8	2010-03-12 00:00:00	2010-03-22 00:00:00	127	1	0	0	
9	2010-03-22 00:00:00	2010-04-01 00:00:00	127	1	0	0	
10	2010-04-01 00:00:00	2010-04-11 00:00:00	127	1	0	0	
** 11	2010-02-20 00:00:00	2010-03-02 00:00:00	127	1	0	0	

```
11 rows in set (0.00 sec)
```

Payload table for the bad write and its override:

```
mysql> select * from Demo where seqno in (6,11) ;
+-----+-----+-----+-----+
| SEQNO | ROW_COUNTER | Gain | Id |
+-----+-----+-----+-----+
| 6 | 1 | 5000 | 5 |
| 6 | 2 | 5000 | 5 |
| 6 | 3 | 5000 | 5 |
| 11 | 1 | 500 | 5 |
| 11 | 2 | 500 | 5 |
| 11 | 3 | 500 | 5 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

When not using overlay versioning, by setting `versiondate=TimeStamp()` or any other time than `TimeStamp(0,0)` the consequences are:

1. payload table is the same
2. `test_read_good` and `test_read_allgood` fail
3. validity table has `VERSIONDATE` (in this case aligned with `INSERTDATE`)

```
mysql> select * from DemoVld ;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART | TIMEEND | SITEMASK | SIMMASK | SUBSITE | TASK | AGGR |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 2010-01-01 00:00:00 | 2010-01-11 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 2 | 2010-01-11 00:00:00 | 2010-01-21 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 3 | 2010-01-21 00:00:00 | 2010-01-31 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 4 | 2010-01-31 00:00:00 | 2010-02-10 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 5 | 2010-02-10 00:00:00 | 2010-02-20 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 6 | 2010-02-20 00:00:00 | 2010-03-02 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 7 | 2010-03-02 00:00:00 | 2010-03-12 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 8 | 2010-03-12 00:00:00 | 2010-03-22 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 9 | 2010-03-22 00:00:00 | 2010-04-01 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 10 | 2010-04-01 00:00:00 | 2010-04-11 00:00:00 | 127 | 1 | 0 | 0 | 0 |
| 11 | 2010-02-20 00:00:00 | 2010-03-02 00:00:00 | 127 | 1 | 0 | 0 | 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

Overlay versioning is the default if no `versiondate` is set in the write context.

## 21.6.4 Many more examples of DB writing with DybDbi

Many examples of writing to the DB using `DybDbi` are in `dybgaudi:Database/DybDbiTest/tests/test_07.py`. The full range of DBI functionality is exercised from `DybDbi` by the tests in `dybgaudi:Database/DybDbiTest/tests/`

## 21.6.5 Using DBWriter to write to tmp\_offline\_db

The `dybgaudi:Database/DBWriter` is implemented mostly in C++ and is currently rather inflexible. [dybsvn:ticket:???](#)

## 21.7 DB Table Reading

- DB Reading with `DybDbi`
  - Default Context Reading
  - Examine Default Read Context
  - Change Read Context
- Using `mysql` client
- Hands On Exercise 5 : Read from DB with varying context
- Hands On Exercise 6 : Read run timestart/timeend from `DaqRunInfo` table
  - Default Context Query
  - Modify to use wideopen validity context
- Fixing this page if it breaks

### 21.7.1 DB Reading with `DybDbi`

`DybDbi` exposes most DBI functionality to python. Details in [doc:5642](#). An example of using `DybDbi` to make a DBI query from *ipython*

#### Default Context Reading

A `DbiResultPtr<GCalibPmtSpec>` is constructed under the covers with a default context that is created from a string serialization obtained from the `.spec` file.

```
In [1]: from DybDbi import GCalibPmtSpec, TimeStamp
In [2]: r = GCalibPmtSpec.Rpt()          ## result pointer
```

On requesting the length the DB is queried and `GCalibPmtSpec` instances are created corresponding to the payload rows obtained by the query.

```
In [4]: len(r)                          ## DB is accessed here
Out[4]: 208
```

**Warning:** When zero results are returned it means that the context does not match entries in the DB

Payload instances are accessible by list-like access on the result pointer.

```
In [19]: print(r[0].asdict)
{'Status': 1, 'PmtId': 536936705, 'Describ': 'SABAD1-ring01-column01', 'PrePulseProb': 0.0, 'SigmaSpe
In [6]: r[0]          # r[-1] array/slice access to T* Row objs
In [7]: r[0].spehigh
Out[5]: 20.0
```

#### Examine Default Read Context

```
In [4]: r.ctx        ## representation of default DbiCtx in use
Out[4]:
```

```
{ 'CtorMask': 2097278,
  'DetectorId': 0,
  'Mask': 2097278,
  'SimFlag': 1,
  'Site': 127,
  'SubSite': 0,
  'TableName': 'CalibPmtSpec',
```

```
'Task': 0,
'TimeStamp': Tue, 12 Apr 2011 14:35:49 +0000 (GMT) +564713000 nsec,
'UpdateMask': 0}
```

```
In [5]: GCalibPmtSpec.MetaRctx ## the default DbiCtx supplied in the .spec file
```

```
Out [5]: 'Site.kAll,SimFlag.kData,TimeStamp.kNOW,DetectorId.kUnknown,SubSite.kDefaultSubSite,Task.kDe
```

## Change Read Context

Under the covers changing the read context results in a new `DbiResultPtr<T>` being instantiated, and the old one being cleaned up.

```
In [6]: r.ctx(timestamp=TimeStamp(2010,8,10,18,30,0)) ## anything back then ?
```

```
Out [6]:
```

```
{ 'CtorMask': 2097278,
  'DetectorId': 0,
  'Mask': 2097278,
  'SimFlag': 1,
  'Site': 127,
  'SubSite': 0,
  'TableName': 'CalibPmtSpec',
  'Task': 0,
  'TimeStamp': Tue, 10 Aug 2010 18:30:00 +0000 (GMT) +          0 nsec,
  'UpdateMask': 16}
```

```
In [7]: len(r)
```

```
DbiRpt<GCalibPmtSpec>::Delete
```

```
DbiRpt<GCalibPmtSpec>::MakeResultPtr tablename variant of standard ctor, tablename: CalibPmtSpec
```

```
Caching new results: ResultKey: Table: row: No vrecs
```

```
DbiCtx::RegisterCreation [DbiRpt<GCalibPmtSpec>] mask:2097278 Site,SimFlag,DetectorId,TimeStamp,SubS
```

```
Out [7]: 0 ## nope
```

```
In [8]: r.ctx(timestamp=TimeStamp()) ## default timestamp is now
```

```
Out [8]:
```

```
{ 'CtorMask': 2097278,
  'DetectorId': 0,
  'Mask': 2097278,
  'SimFlag': 1,
  'Site': 127,
  'SubSite': 0,
  'TableName': 'CalibPmtSpec',
  'Task': 0,
  'TimeStamp': Tue, 12 Apr 2011 14:37:29 +0000 (GMT) +443074000 nsec,
  'UpdateMask': 16}
```

```
In [9]: len(r)
```

```
DbiRpt<GCalibPmtSpec>::Delete
```

```
DbiRpt<GCalibPmtSpec>::MakeResultPtr tablename variant of standard ctor, tablename: CalibPmtSpec
```

```
Caching new results: ResultKey: Table:CalibPmtSpec row:GCalibPmtSpec. 1 vrec (seqno;versiondate): 2
```

```
DbiTimer:CalibPmtSpec: Query done. 208rows, 19.1Kb Cpu 0.0 , elapse 0.0
```

```
DbiCtx::RegisterCreation [DbiRpt<GCalibPmtSpec>] mask:2097278 Site,SimFlag,DetectorId,TimeStamp,SubS
```

```
Out [9]: 208
```

## 21.7.2 Using mysql client

**Note:** Interactive examination of the Database is an invaluable first step to validating updates.

---

By virtue of the `client` section, in the configuration `~/ .my.cnf`, which is read by the client, the `mysql` command with no arguments starts an interactive command line interface allowing you to query your configured database (this is not the server, that runs as `mysqld`).

See *Configuring DB Access* for an example of the `client` section, which will typically correspond to the `tmp_offline_db` section.

Example `mysql` client session:

```
[blyth@belle7 DybPython]$ mysql                ## reads from client section
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 32808
Server version: 5.0.77-log Source distribution
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> status                ## verify are connected with expected DB and identity
```

```
-----
mysql Ver 14.12 Distrib 5.0.67, for redhat-linux-gnu (i686) using EditLine wrapper
```

```
Connection id:          32808
Current database:       tmp_offline_db
Current user:           noddy@belle7.nuu.edu.tw
...
```

```
mysql> show tables ;
```

```
+-----+
| Tables_in_tmp_offline_db |
+-----+
| CalibFeeSpec              |
| CalibFeeSpecVld          |
| CalibPmtSpec              |
| CalibPmtSpecVld          |
| DaqRunInfo                |
..
```

```
mysql> select * from CalibPmtSpecVld ;                ## examine changes made
```

SEQNO	TIMESTART	TIMEEND	SITEMASK	SIMMASK	SUBSITE	TASK	AGGREGATE
26	2011-01-22 08:15:17	2020-12-30 16:00:00	127	1	0	0	
18	2010-06-21 07:49:24	2038-01-19 03:14:07	32	1	1	0	
27	2011-01-22 08:15:17	2020-12-30 16:00:00	127	2	0	0	
28	2011-01-22 08:15:17	2038-01-19 03:14:07	1	2	1	0	
29	2011-01-22 08:15:17	2038-01-19 03:14:07	32	1	1	0	
23	2010-09-16 06:31:34	2038-01-19 03:14:07	32	1	1	0	
24	2010-09-21 05:48:57	2038-01-19 03:14:07	32	1	2	0	
25	2010-09-22 04:26:59	2038-01-19 03:14:07	32	1	2	0	
30	2010-09-22 12:26:59	2038-01-19 03:14:07	127	3	0	0	

```
9 rows in set (0.00 sec)
```

```
mysql> select * from CalibPmtSpec where seqno = 30 ;          ## seqno provides the link between pa
+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | ROW_COUNTER | PMTID      | PMTDESCRIB          | PMTSTATUS | PMTSPEHIGH | PMTSIGMASPEHIGH |
+-----+-----+-----+-----+-----+-----+-----+
| 30    | 1           | 536936705 | SABAD1-ring01-column01 | 1         | 41.0905    | 11.356           |
| 30    | 2           | 536936705 | SABAD1-ring01-column01 | 1         | 41.0905    | 11.356           |
| 30    | 3           | 536936705 | SABAD1-ring01-column01 | 1         | 41.0905    | 11.356           |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

### 21.7.3 Hands On Exercise 5 : Read from DB with varying context

**Note:** this can be performed either on a copy tmp\_offline\_db or on offline\_db

Follow the examples of the previous two sections to perform, DIY steps:

1. Use mysql client to query the Vld table, eg `select * from CalibPmtSpecVld ;`
2. Perform queries with varying contexts : with timestamps to distinguish between sets of parameters
3. Contrast row counts obtained with expectations from mysql client selects

Hint, the `vrec.DbiValidityRec` attribute on a `cls.Rpt()` provides access to the `SEQNO` of the query which allows a payload query using a **where clause** to select payload rows corresponding to a particular validity.

```
In [12]: r.vrec
Out[12]:
DbiValidityRec
{  'AggregateNo': -1,
   'ContextRange': |site 0x007f|sim 0x007f
                   2011-01-22 08:15:17.000000000Z
                   2020-12-30 16:00:00.000000000Z,
   'DatabaseLayout': 'NULL',
   'DbNo': 0L,
   'InsertDate': Fri, 25 Feb 2011 08:10:15 +0000 (GMT) +      0 nsec,
   'L2CacheName': '26_2011-01-22_08:15:17',
   'SeqNo': 26L,
   'SubSite': 0,
   'Task': 0,
   'VersionDate': Sat, 22 Jan 2011 08:15:17 +0000 (GMT) +      0 nsec}
```

```
In [13]: r.vrec.seqno
Out[13]: 26L
```

### 21.7.4 Hands On Exercise 6 : Read run timestart/timeend from DaqRunInfo table

**Note:** this can be performed either on a copy tmp\_offline\_db or on offline\_db

#### Default Context Query

### DaqRunInfo has moved

As a scraped table DaqRunInfo does not belong in the db.py default set that gets copied to tmp\_offline\_db Due to this some of the below will no longer work, an adjustment using cacades needs to be tested.

Default context will probably yield no results:

```
In [27]: import os
```

```
In [28]: os.environ['DBCONF'] = "tmp_offline_db"
```

```
In [30]: from DybDbi import GDaqRunInfo
```

```
In [31]: rpt = GDaqRunInfo.Rpt()
```

```
In [32]: len(rpt)
```

```
DbiRpt<GDaqRunInfo>::MakeResultPtr tablename variant of standard ctor, tablename: DaqRunInfo
```

```
Caching new results: ResultKey: Table: row: No vrecs
```

```
DbiCtx::RegisterCreation [DbiRpt<GDaqRunInfo>] mask:2097278 Site,SimFlag,DetectorId,TimeStamp,SubSite
```

```
Out[32]: 0
```

### Modify to use wideopen validity context

Use exceedingly low level technique to access the DaqRunInfo row for a particular run number:

```
In [32]: run = 5647
```

```
In [33]: rpt.ctx( sqlcontext="1=1" , task=-1 , subsite=-1 )
```

```
In [34]: len(rpt)
```

## 21.7.5 Fixing this page if it breaks

On building the docs some of the ipython sessions listed above are actually performed, making live DB queries etc... This leaves the possibility of failure, to debug this just build a single page with eg:

```
sphinx-build -b dirhtml -d _build/doctrees . _build/dirhtml sop/dbread.rst
```

## 21.8 Debugging unexpected parameters

### Context Mismatch

When you do not get what you expect, the overwhelming most likely cause is a query context that does not match the DB entries

The key to debugging is isolation of issues. The recommended first steps to locate where problems are:

- Configuration check
- `ipython DybDbi session`
- `DbiDataSvc test`
- `mysql client session`
- Following the `mysql tail`
- GDB Debugging of Template Laden DBI

### 21.8.1 Configuration check

When operating at the service level first verify that you are using the desired services, ie that you are not using `StaticCalibDataSvc` when you intend to use the DB with `DbiCalibDataSvc`. An example of switching services is provided by `dybsvn:r11843`

Verify that you are connecting to the DB you expect. Avoid confusing config such as having multiple updates of `DBCONF`

```
[blyth@belle7 dbtest]$ grep DBCONF \*.py
runCalib.py:os.environ['DBCONF'] = "offline_db"
testCalibDirectly.py:os.environ.update( DBCONF="tmp_wangzm_offline_db" )
```

For reading use of `os.environ.setdefault("DBCONF", "offline_db")` is recommended, allowing external overriding.

### 21.8.2 ipython DybDbi session

Get into `ipython` either with `DBCONF` set externally or internally. Adjust the default context (for example to correspond to the timestamp of a data file) and do a DBI query:

```
In [1]: import os

In [1]: os.environ['DBCONF'] = "tmp_offline_db"

In [1]: from DybDbi import GCalibPmtSpec, TimeStamp

In [2]: rpt = GCalibPmtSpec.Rpt()

In [3]: from DybDbi import TimeStamp

In [4]: rpt.ctx(timestamp=TimeStamp(2011, 1, 22, 10,0,0 ) )

In [5]: len(rpt)

In [6]: rpt[0]
```

### 21.8.3 DbiDataSvc test

Try running the standard `DbiDataSvc.TestDbiDataSvc` with an appropriate timestamp.

```
DBCONF="tmp_offline_db" nuwa.py -A none --history=off -n 1 -m "DbiDataSvc.TestDbiDataSvc --timeString"
```

---

**Note:** additional context flexibility for this tool would improve it's usefulness as a debugging tool

---

## 21.8.4 mysql client session

Check your [client] section is pointed to the same DB and perform some simple queries:

```
mysql> select * from CalibPmtSpecVld order by TIMESTART ;
```

SEQNO	TIMESTART	TIMEEND	SITEMASK	SIMMASK	SUBSITE	TASK	AGGREGATE
31	1970-01-01 00:00:00	2038-01-19 03:14:07	127	3	0	7	
30	1970-01-01 00:00:00	2038-01-19 03:14:07	127	3	0	7	
18	2010-06-21 07:49:24	2038-01-19 03:14:07	32	1	1	0	
23	2010-09-16 06:31:34	2038-01-19 03:14:07	32	1	1	0	
24	2010-09-21 05:48:57	2038-01-19 03:14:07	32	1	2	0	
25	2010-09-22 04:26:59	2038-01-19 03:14:07	32	1	2	0	
29	2011-01-22 08:15:17	2038-01-19 03:14:07	32	1	1	0	
28	2011-01-22 08:15:17	2038-01-19 03:14:07	1	2	1	0	
27	2011-01-22 08:15:17	2020-12-30 16:00:00	127	2	0	0	
26	2011-01-22 08:15:17	2020-12-30 16:00:00	127	1	0	0	

10 rows in set (0.00 sec)

## 21.8.5 Following the mysql tail

If you have access to the DB server machine and have privileges to access the mysql log file it is exceedingly informative to leave a process tailing the mysql log. For example with:

```
sudo tail -f /var/log/mysql.log
```

This allows observation of the mysql commands performed as you interactively make queries from ipython:

```
[blyth@belle7 ~]$ mysql-tail
64352 Query          SHOW COLUMNS FROM `CalibPmtSpecVld`
64352 Query          SHOW TABLE STATUS LIKE 'CalibPmtSpecVld'
64352 Prepare        [1] select * from CalibPmtSpecVld where      TimeStart <= '2011-02-01 00:00:00'
64352 Execute        [1] select * from CalibPmtSpecVld where      TimeStart <= '2011-02-01 00:00:00'
64352 Prepare        [2] select min(TIMESTART) from CalibPmtSpecVld where TIMESTART > '2011-02-01 00:00:00'
64352 Execute        [2] select min(TIMESTART) from CalibPmtSpecVld where TIMESTART > '2011-02-01 00:00:00'
64352 Prepare        [3] select min(TIMEEND) from CalibPmtSpecVld where TIMEEND > '2011-02-01 00:00:00'
64352 Execute        [3] select min(TIMEEND) from CalibPmtSpecVld where TIMEEND > '2011-02-01 00:00:00'
64352 Prepare        [4] select max(TIMESTART) from CalibPmtSpecVld where TIMESTART < '2011-02-01 00:00:00'
64352 Execute        [4] select max(TIMESTART) from CalibPmtSpecVld where TIMESTART < '2011-02-01 00:00:00'
64352 Prepare        [5] select max(TIMEEND) from CalibPmtSpecVld where TIMEEND < '2011-02-01 00:00:00'
64352 Execute        [5] select max(TIMEEND) from CalibPmtSpecVld where TIMEEND < '2011-02-01 00:00:00'
64352 Prepare        [6] select * from CalibPmtSpec where          SEQNO= 26
64352 Execute        [6] select * from CalibPmtSpec where          SEQNO= 26
64352 Quit
```

The SQL queries from the log can then be copy-and-pasted to a mysql client session for interactive examination.

### Todo

Provide a way for non-administrators to do this style of debugging, perhaps with an extra DBI log file ?

## 21.8.6 GDB Debugging of Template Laden DBI

Isolate issue into small python test, then:

```

gdb $(which python)
(gdb) set args test_dydbdi_write.py
(gdb) b "DbiWriter<GDcsPmtHv>::operator<<(GDcsPmtHv const&)"
Function "DbiWriter<GDcsPmtHv>::operator<<(GDcsPmtHv const&)" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
(gdb) r

```

Determining the symbol names (especially when templated) can be painful.:

```

(gdb) b "DbiWriter<GDcsPmtHv>&
Display all 378426 possibilities? (y or n)
(gdb) b DbiWriter.tpl:100

```

Hunting for symbols inside the `.so` much is faster than attempting to use tab completion, or manually compose the magic symbol names:

```

objdump -t ./i686-slc4-gcc34-dbg/libDybDbiLib.so | c++filt | grep DbiWriter\<GDcsPmtHv\>::operator
0014b0ae w F .text 000002ca DbiWriter<GDcsPmtHv>::operator<<(GDcsPmtHv const&)

```

## 21.9 DB Table Creation

- Workflow Outline for Adding Tables
- Design Tables
- Prepare `.spec` File
- Generate Row Classes from `.spec`
  - Ensuring Consistency When Changing Spec
- Copy `offline_db` to `tmp_offline_db`
- Create New Tables in `tmp_offline_db`
- Populate New Table With Dummy Data
- Verify tables using the `mysql` client

### 21.9.1 Workflow Outline for Adding Tables

The Row classes needed to interact with the database and the Database table descriptions are generated from specification files (`.spec`) stored in `dybgaudi:Database/DybDbi/spec`. The generation is done when the CMT `DybDbi` package is built.

#### Commit early and often

Building `DybDbi` never creates tables or even connects to any DB, so share your `.spec` while you are working on them

1. create the `.spec`
2. generate the code and table descriptions by building `dybgaudi:Database/DybDbi`
3. create the test tables in a copy of `offline_db`
4. populate the table with some dummy data using `DybDbi`
5. make queries against the table using `DybDbi` and services

These last 2 steps in python can then be rearranged into a nosetest.

## 21.9.2 Design Tables

When considering how to divide parameters into tables bear in mind:

- Quantities that are not updated together should not be stored together in the same table
- Joins between tables are not supported by DBI; simplicity is mandatory

Things to avoid in tables:

- duplication, for example integer codes accompanied by a human readable string might seem nice for users but in the long run is a bug magnet
- strings where integer codes are more appropriate, integer columns are easier and more efficient to query against
- varchar when other types can be used, especially in frequently accessed tables

## 21.9.3 Prepare .spec File

Spec files need to be created in `dybgaudi:Database/DybDbi/spec` and named after the table name prefixed with a **G**. An example of a spec file `dybgaudi:Database/DybDbi/spec/GCalibFeeSpec.spec`:

```
"""
docstring
"""
;
table                               | meta                               | legacy                               | Canl
CalibFeeSpec                         | 1                                 | CalibFeeSpec                        | kFAI
;
meta                                 | rctx                               |
2                                    | Site.kAll, SimFlag.kData, TimeStamp.kNOW, DetectorId.kUnknown, SubSite.k
;
meta                                 | wctx                               |
3                                    | SiteMask.kAll, SimMask.kData, TimeStart.kBOT, TimeEnd.kEOT, AggNo.k-1, S
;
name                                 | codetype                           | dbtype                               | legacy                               | memb
ChannelId                           | DayaBay::FeeChannelId             | int(10) unsigned                    | channelId                            | m_channelId
Status                               | int                                | int(10) unsigned                    | status                               | m_status
AdcPedestalHigh                     | double                             | double                              | pedestalHigh                         | m_adcPedestalH
AdcPedestalHighSigma                | double                             | double                              | sigmaPedestalHigh                   | m_adcPedestalH
AdcPedestalLow                      | double                             | double                              | pedestalLow                          | m_adcPedestalL
AdcPedestalLowSigma                 | double                             | double                              | sigmaPedestalLow                    | m_adcPedestalL
AdcThresholdHigh                    | double                             | double                              | thresholdHigh                       | m_adcThreshol
AdcThresholdLow                     | double                             | double                              | thresholdLow                        | m_adcThreshol
```

Spec files are structured into sections divided by semicolons in the first column of otherwise blank lines. The sections comprise:

1. documentation string in triple quotes : which is propagated thru the C++ to the python commandline and used in generated documentation `oum:genDbi/GCalibFeeSpec/`
2. class level quantities (identified by the presence of the meta key)
3. row level quantities (without the meta key)

Within each section a vertical bar delimited format is used that is parsed into python dicts and lists of dicts by `oum:api/dydbipre/`. These objects are made available within the context used to fill the django templates `dybgaudi:Database/DybDbi/templates` for the various derived files : classes, headers, documentation , sql descriptions. Further details are in the API docs linked above.

Specified quantities:

class level qty	notes
table	Name of the table in Database, by convention without the G prefix
meta	Simply used to identify class level properties, values are meaningless
legacy	Name of table again, can be used for migrations but in typical usage use the same string as the table qty
CanL2Cache	leave as kFALSE, enabling L2Cache is not recommended
class	Name of the generated class, use table name prefixed with a G
rctx	Default DBI read context, make sure the TableName.kName is correct
wctx	Default DBI write context, make sure the TableName.kName is correct

### Default DBI Read/Write Contexts

The default context qtys use a comma delimited string representation of DBI context and contextrange based on enum value labels. While these are conveniences that can easily be subsequently changed, it is important to ensure that the NAME in TableName.kNAME corresponds to the name of the database table.

Row level quantities are mostly self explanatory, and are detailed in [oum:api/dydbdipre/](http://oum:api/dydbdipre/).

### Capitalized Attribute Names

To conform to the C++/ROOT convention for Getters/Setters, the column name should be capitalized.

The ones that might be confusing are:

row level qty	notes
name	column name as used in the C++ Getter and Setter methods
legacy	name of the field in the database table
memb	name of the C++ instance variable

When creating new specifications that do not need to conform to existing tables, using the same string for all the above three quantities is recommended.

### Todo

plant internal reference targets to genDbi documentation

## 21.9.4 Generate Row Classes from .spec

On building the CMT package `dybgaudi:Database/DybDbi` the classes corresponding to the `.spec` are generated in the `Database/DybDbi/genDbi` directory. Typically the build will fail with compilation errors in the event of problems.

### Ensuring Consistency When Changing Spec

`DatabaseInterface` and `DybDbi` packages make strong use of templates and generated code. Because of this the 1st thing to try when meeting crashes such as `segv` is to ensure full consistency by cleaning all generated files and rebuilding from scratch.

Deep cleaning can be done by:

```
#DBI
echo rm -rf $CMTCONFIG          ## check
echo rm -rf $CMTCONFIG | sh    ## do

##DybDbi
echo rm -rf genDbi genDict $CMTCONFIG
echo rm -rf genDbi genDict $CMTCONFIG | sh
```

Rebuild *DatabaseInterface* and then *DybDbi*

## 21.9.5 Copy offline\_db to tmp\_offline\_db

Instructions at *Copy offline\_db to tmp\_offline\_db*

## 21.9.6 Create New Tables in tmp\_offline\_db

Configure the DB to connect to with the `DBCONF` envvar, see *Configuring DB Access*

```
[blyth@belle7 DybDbi]$ DBCONF=tmp_offline_db ipython
Python 2.7 (r27:82500, Feb 16 2011, 11:40:18)
IPython 0.9.1 -- An enhanced Interactive Python.
...
In [1]: from DybDbi import gDbi, GPhysAd
In [2]: gDbi.Status()
DybDbi activating DbitableProxyRegistry
Using DBCONF.Export to prime environment with : from DybPython import DBCONF ; DBCONF.Export('tmp_offl
dbconf:export_to_env from $SITEROOT/../../my.cnf:~/my.cnf section tmp_offline_db
Successfully opened connection to: mysql://belle7.nuu.edu.tw/tmp_offline_db
This client, and MySQL server (MySQL 5.0.77-log) does support prepared statements.
DbiCascader Status:-
Status    URL

Closed          0 mysql://belle7.nuu.edu.tw/tmp_offline_db

In [3]: GPhysAd().CreateDatabaseTables(0,"PhysAd")    ## dbno in cascade and tablename without the G p
Out[3]: 1
```

### Notes:

- `DBCONF=tmp_offline_db ipython` sets the configuration for the ipython session
- The call to `gDbi.Status()` is used to verify are talking to the intended Database !

#### Only for new tables

As `CreateDatabaseTables` uses `create table if not exists` a pre-existing table must be manually dropped (**loosing all entries**) before this will work.

## 21.9.7 Populate New Table With Dummy Data

Get into ipython again, with `DBCONF=tmp_offline_db ipython` and add some dummy entries:

```

In [1]: from DybDbi import gDbi, GPhysAd

In [2]: GPhysAd?      ## lookup attribute names

In [3]: r = GPhysAd.Create( AdSerial=1,PhysAdId=10,Describ="red" )
In [4]: g = GPhysAd.Create( AdSerial=2,PhysAdId=20,Describ="green" )
In [5]: b = GPhysAd.Create( AdSerial=3,PhysAdId=30,Describ="blue" )

In [6]: wrt = GPhysAd.Wrt()

In [5]: wrt.Write( r )
DbiWrt<GPhysAd>::MakeWriter standard ctor, contextrange: |site 0x007f|sim 0x007f
      1970-01-01 00:00:00.000000000Z
      2038-01-19 03:14:07.000000000Z
Using DBConf.Export to prime environment with : from DybPython import DBConf ; DBConf.Export('tmp_of
dbconf:export_to_env from $SITEROOT/../../my.cnf:~/my.cnf section tmp_offline_db
Successfully opened connection to: mysql://belle7.nuu.edu.tw/tmp_offline_db
This client, and MySQL server (MySQL 5.0.77-log) does support prepared statements.
DbiCascader Status:-
Status   URL

Closed           0 mysql://belle7.nuu.edu.tw/tmp_offline_db

DbiCtx::RegisterCreation [DbiWrt<GPhysAd>] mask:2128992 SubSite,Task,TimeStart,TimeEnd,SiteMask,SimM
DbiWrt<GPhysAd>::Write

In [5]: wrt.Write( g )
In [6]: wrt.Write( b )

In [7]: wrt.Close()      ## DB is written to here
DbiWrt<GPhysAd>::Close
Out[8]: 1

In [8]: rpt = GPhysAd.Rpt()

In [9]: len(rpt)
DbiRpt<GPhysAd>::MakeResultPtr tablename variant of standard ctor, tablename: PhysAd
Caching new results: ResultKey: Table:PhysAd row:GPhysAd.  1 vrec (seqno;versiondate): 1;1970-01-01
DbiTimer:PhysAd: Query done. 3rows,      0.0Kb Cpu  0.0 , elapse  0.0
DbiCtx::RegisterCreation [DbiRpt<GPhysAd>] mask:2097278 Site,SimFlag,DetectorId,TimeStamp,SubSite,Ta
Out[9]: 3

In [10]: rpt[0].asdict
Out[10]: {'AdSerial': 1, 'Describ': 'red', 'PhysAdId': 10}

```

**Get Real**

More realistic testing would modify the writers context range and readers context from their defaults.

## 21.9.8 Verify tables using the mysql client

After adding tables check them with the `mysql` client. Use the `status` command to check are connected to the expected database, see [Configuring DB Access](#) if not.

Example `mysql` shell session:

```
mysql> status
```

```
mysql> show tables ;
```

```
+-----+
| Tables_in_tmp_offline_db |
+-----+
| CalibFeeSpec              |
| CalibFeeSpecVld          |
| CalibPmtSpec              |
| CalibPmtSpecVld          |
| DaqRunInfo                |
| DaqRunInfoVld            |
| FeeCableMap               |
| FeeCableMapVld           |
| LOCALSEQNO                |
| PhysAd                    |
| PhysAdVld                 |
| SimPmtSpec                |
| SimPmtSpecVld            |
+-----+
13 rows in set (0.00 sec)
```

```
mysql> select * from PhysAd ;
```

```
+-----+-----+-----+-----+-----+
| SEQNO | ROW_COUNTER | ADSERIAL | PHYSADID | DESCRIB |
+-----+-----+-----+-----+-----+
| 1 | 1 | 1 | 10 | red |
| 1 | 2 | 2 | 20 | green |
| 1 | 3 | 3 | 30 | blue |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> select * from PhysAdVld ;
```

```
+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART | TIMEEND | SITEMASK | SIMMASK | SUBSITE | TASK | AGGR |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | 1970-01-01 00:00:00 | 2038-01-19 03:14:07 | 127 | 1 | 0 | 0 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

## 21.10 DB Validation and Testing

- Interactive Checking with ipython
  - DB Checking
  - Ascii Catalog Checking
- Workflow Checks
- Generic Validation
- Specific Parameter Validation

## 21.10.1 Interactive Checking with ipython

### DB Checking

Simple select on LOCALSEQNO table, provides same info as the .seqno methods below:

```
mysql> select * from LOCALSEQNO ;
+-----+-----+
| TABLENAME | LASTUSEDSEQNO |
+-----+-----+
| *          |                |
| CalibFeeSpec |          113 |
| CalibPmtSpec |           50 |
| FeeCableMap |            3 |
| HardwareID  |          372 |
| CableMap    |          460 |
| Reactor     |          372 |
+-----+-----+
7 rows in set (0.07 sec)
```

```
In [1]: from DybPython import DB
```

```
In [2]: db = DB("offline_db")          ## the DBCONF section name
```

```
In [3]: db.seqno          ## obtained from LOCALSEQNO table, providing LASTUSEDSEQNO values
```

```
Out[3]:
{'CableMap': 460,
 'CalibFeeSpec': 113,
 'CalibPmtSpec': 50,
 'FeeCableMap': 3,
 'HardwareID': 372,
 'Reactor': 372}
```

```
In [4]: db.fabseqno      ## fabricated from .allseqno with SEQNO counts
```

```
Out[4]:
{'CableMap': 460,
 'CalibFeeSpec': 111,
 'CalibPmtSpec': 29,
 'FeeCableMap': 3,
 'HardwareID': 372,
 'Reactor': 372}
```

```
## legacy tables : CalibFeeSpec CalibPmtSpec have know SEQNO irregularities
## .. all other tables must be consistent
```

```
In [5]: db.allseqno     ## obtained via SQL queries on validity tables
```

```
Out[5]:
{'CableMap': [1,
              2,
              3,
              4,
              5,
              6,
              7,
              ... too long to show ...
              370,
              371,
              372]}
```

```
In [6]: db.allseqno.keys()
Out[6]:
['Reactor',
 'CalibFeeSpec',
 'HardwareID',
 'CalibPmtSpec',
 'FeeCableMap',
 'CableMap']
```

## Ascii Catalog Checking

Simple cat of LOCALSEQNO.csv table, provides same info as the .seqno methods below:

```
[blyth@belle7 ~]$ svnversion ~/dybaux/catalog/tmp_offline_db
4974
[blyth@belle7 ~]$ cat ~/dybaux/catalog/tmp_offline_db/LOCALSEQNO/LOCALSEQNO.csv
TABLENAME char(64),LASTUSEDSEQNO int(11),PRIMARY KEY (TABLENAME)
"*,0
"CableMap",460
"CalibFeeSpec",113
"CalibPmtSpec",50
"CoordinateAd",1
"CoordinateReactor",1
"FeeCableMap",3
"HardwareID",372
"Reactor",372
```

```
In [8]: from DybPython.asciicat import AsciiCat
```

```
In [9]: cat = AsciiCat("~/dybaux/catalog/tmp_offline_db")      ## reads all entries into memory
```

```
In [10]: cat.seqno
```

```
Out[10]:
{'CableMap': 460,
 'CalibFeeSpec': 113,
 'CalibPmtSpec': 50,
 'CoordinateAd': 1,
 'CoordinateReactor': 1,
 'FeeCableMap': 3,
 'HardwareID': 372,
 'Reactor': 372}
```

```
In [11]: cat.fabseqno
```

```
Out[11]:
{'CableMap': 460,
 'CalibFeeSpec': 111,
 'CalibPmtSpec': 29,
 'CoordinateAd': 1,
 'CoordinateReactor': 1,
 'FeeCableMap': 3,
 'HardwareID': 372,
 'Reactor': 372}
```

## 21.10.2 Workflow Checks

At each step of the workflow the tools, such as `db.py` and `dbaux.py` perform DBI integrity checks, checking things like:

1. SEQNO consistency between LOCALSEQNO table and actual payload and validity tables
2. table existence
3. payload/validity consistency

Furthermore when performing operations such as **reloadcat** or **rempeat** that involve both an ascii catalog and the DB the differences present in the ascii catalog are checked to be valid DBI updates with the expected tables and SEQNO values.

## 21.10.3 Generic Validation

Packages containing tests that focus on DB access/function testing

package	notes on tests
<code>dybgaudi:DybPython</code>	operation of <code>db.py</code> and <code>dbsvn.py</code> , and workflow steps
<code>dybgaudi:Database/DybDbi</code>	simple readonly access to a variety of tables
<code>dybgaudi:Database/DbiTest</code>	DBI supplied C++ tests of most DBI functionality
<code>dybgaudi:Database/DybDbiTest</code>	<code>DybDbi</code> equivalents of all relevant <code>DbiTest</code>
<code>dybgaudi:Database/DbiValidate</code>	generic tests of DBI table structure : integers inside enums, PK etc..

## 21.10.4 Specific Parameter Validation

Packages that focus on validating characteristics of specific tables. For example tests could include constraints on

1. mean/min/max values of parameters
2. values of quantities derived from fits to the parameters
3. differences in parameters between updates

### Generic possibilities

Perhaps some constraints might be applicable generically, for example by extending the `.spec` file to include expectations on allowable mean/min/max and deltas

**Warning:** Table experts need to create tests, inside the `dybsvn` CMT packages used for preparing the parameters

Packages containing table specific tests:

package	notes
<code>dybgaudi:Calibration/CalibParam</code>	<b>NOT YET</b>
...	

## 21.11 DB Administration

- Temporary DB Setup by MySQL Administrators

### 21.11.1 Temporary DB Setup by MySQL Administrators

For non-central temporary databases of a short lived nature it is very convenient to give table experts substantial permissions in temporary databases of specific names. Database names based on SVN user account names (listed at [dybsvn:report:11](#)) are recommended. The names must be prefixed with **tmp\_** as the `db.py` script enforces this as a safeguard for **load** and **loadcat** commands eg:

```
tmp_wangzm_offline_db
tmp_jpochoa_offline_db
tmp_ww_offline_db
tmp_blyth_offline_db
tmp_zhanl_offline_db
```

To grant permissions mysql administrators need to perform something like the below, which give all privileges except *Grant\_Priv*:

```
mysql> grant all on tmp_wangzm_offline_db.* to 'wangzm'@'%' identified by 'realplaintextpassword' ;
```

Administrators can list existing database level permissions with:

```
mysql> select * from mysql.db ;
```

Host	Db	User	Select_priv	Insert_priv	Update_priv
%	offline_db_20101125	dayabay	Y	N	N
%	offline_db_20101124	dayabay	Y	N	N
%	tmp_blyth_offline_db	blyth	Y	Y	Y
...					

## 21.12 DB Services

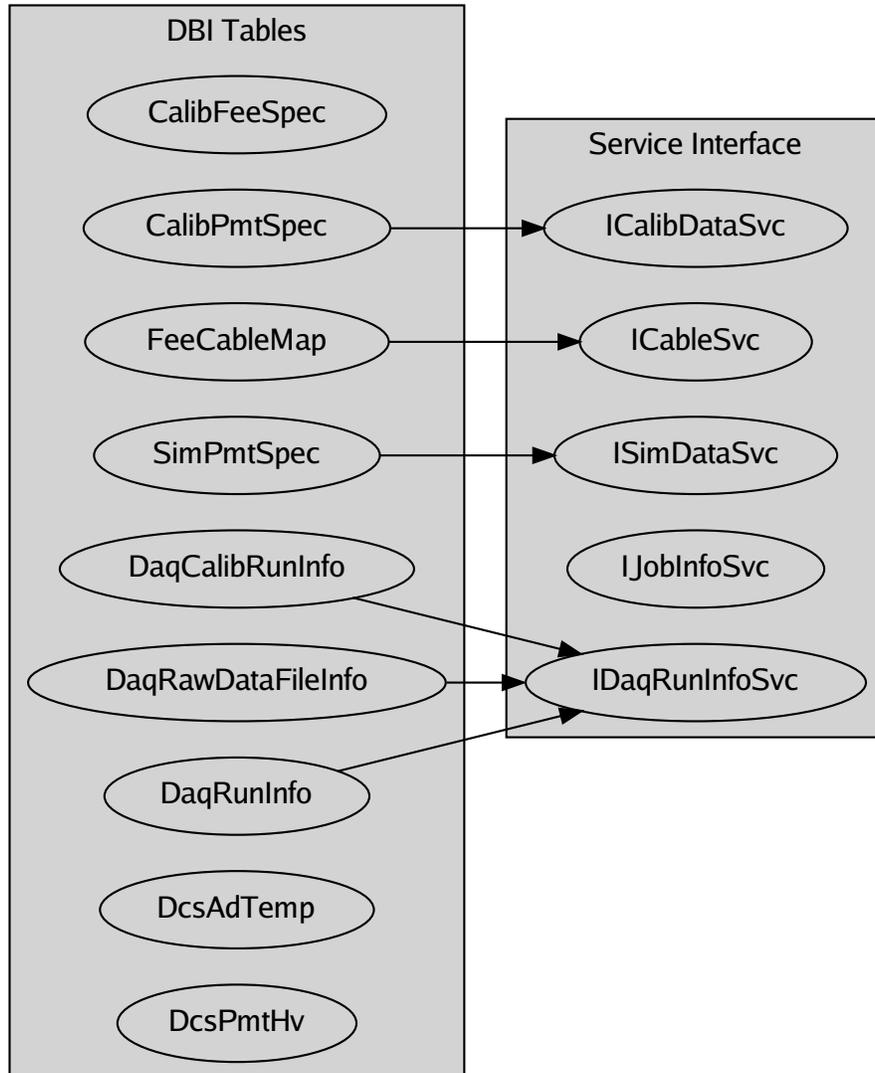
- User Interfaces to DBI Data
- Tables which are Missing something

### 21.12.1 User Interfaces to DBI Data

To a large degree the low level access to DBI tables is shielded from users by the service layer. The intention is to isolate changes in the underlying DBI tables from user analysis code. From the user's perspective, a series of Interfaces are defined:

Interface	Description
ICableSvc	Cable mapping
ICalibDataSvc	Calibration parameters
ISimDataSvc	PMT/Electronics input parameters for simulation
IJobInfoSvc	NuWa Job Information
IDaqRunInfoSvc	DAQ Run information

These interfaces are defined in `dybgaudi:DataModel/DataSvc/DataSvc`



**Please Correct/Update Connections**

Commit updates to `dybgaudi:Documentation/OfflineUserManual/tex/sop/dbserv.rst` in graphviz/dot language

### 21.12.2 Tables which are Missing something

Table	DBI service	DBI Writer
CalibFeeSpec	NO	
SimPmtSpec		NO

### 21.13 DCS tables grouped/ordered by schema

- SAB\_TEMP
- DBNS\_ACU\_HV\_SlotTemp
- DBNS\_Temp
- AD1\_TEMP
- DBNS\_HALL5\_TEMP
- config\_table
- DYBAlarm
- DBNS\_AD1\_HV\_Imon
- DBNS\_AD2\_HV\_Imon
- SAB\_AD1\_HV\_Imon
- SAB\_AD2\_HV\_Imon
- SAB\_AD2\_HV\_SlotTemp
- DBNS\_AD1\_HV\_SlotTemp
- DBNS\_AD2\_HV\_SlotTemp
- SAB\_AD1\_HV\_SlotTemp
- DBNS\_SAB\_TEMP
- site\_table
- DBNS\_MUON\_PMT\_HV\_Imon
- DBNS\_MUON\_PMT\_HV\_SlotTemp
- status\_table
- DBNS\_AD\_HV\_SlotTemp
- dyb\_muoncal
- DBNS\_ACU\_HV\_Pw
- DBNS\_ACU\_HV\_Imon
- DBNS\_ACU\_HV\_Vmon
- EH1\_ENV\_RadonMonitor
- DBNS\_AD1\_LidSensor
- DBNS\_AD2\_LidSensor
- DBNS\_AD1\_VME
- DBNS\_AD2\_VME
- DBNS\_IW\_VME
- DBNS\_Muon\_PMT\_VME
- DBNS\_OW\_VME
- DBNS\_RPC\_VME
- SAB\_AD1\_VME
- DBNS\_AD1\_HV
- DBNS\_AD2\_HV
- SAB\_AD1\_HV\_Vmon
- SAB\_AD2\_HV\_Vmon
- SAB\_AD2\_HV\_Pw
- DBNS\_AD1\_HVPw
- DBNS\_AD2\_HV\_Pw
- SAB\_AD1\_HV\_Pw
- DBNS\_MUON\_PMT\_HV\_Vmon
- DBNS\_MUON\_PMT\_HV\_Pw
- DBNS\_AD\_HV\_Imon
- DBNS\_AD\_HV\_Vmon
- DBNS\_AD\_HV\_Pw

### 21.13.1 SAB\_TEMP

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
SAB_TEMP_PT1	decimal(6,2)	YES		NULL	

### 21.13.2 DBNS\_ACU\_HV\_SlotTemp

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV.Slot11.Temp	decimal(4,2)	YES		NULL	

### 21.13.3 DBNS\_Temp

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_Temp_PT1	decimal(6,2)	YES		NULL	
DBNS_Temp_PT2	decimal(6,2)	YES		NULL	

### 21.13.4 AD1\_TEMP

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
AD1_temp_pt1	decimal(6,2)	YES		NULL	
AD1_temp_pt2	decimal(6,2)	YES		NULL	
AD1_temp_pt3	decimal(6,2)	YES		NULL	
AD1_temp_pt4	decimal(6,2)	YES		NULL	

### 21.13.5 DBNS\_HALL5\_TEMP

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_H5_Temp_PT1	decimal(6,2)	YES		NULL	
DBNS_H5_Temp_PT2	decimal(6,2)	YES		NULL	
DBNS_H5_Temp_PT3	decimal(6,2)	YES		NULL	
DBNS_H5_Temp_PT4	decimal(6,2)	YES		NULL	

### 21.13.6 config\_table

ParaName	varchar(45)	NO	PRI	NULL	
Site	varchar(45)	YES		NULL	
MainSys	varchar(45)	YES		NULL	
SubSys	varchar(45)	YES		NULL	
TableName	varchar(45)	NO	PRI	NULL	
Description	varchar(1023)	YES		NULL	
ReferenceValue	varchar(45)	YES		NULL	

### 21.13.7 DYBAlarm

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
TableName	char(30)	YES		NULL	
Parameter	char(30)	YES		NULL	
Value	char(10)	YES		NULL	
Description	char(50)	YES		NULL	
Status	char(1)	YES		NULL	

### 21.13.8 DBNS\_AD1\_HV\_Imon

### 21.13.9 DBNS\_AD2\_HV\_Imon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV.Slot0.I0	decimal(6,2)	YES		NULL	
DBNS_AD_HV.Slot2.I0	decimal(6,2)	YES		NULL	
DBNS_AD_HV.Slot4.I0	decimal(6,2)	YES		NULL	
DBNS_AD_HV.Slot6.I0	decimal(6,2)	YES		NULL	

### 21.13.10 SAB\_AD1\_HV\_Imon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
SAB_AD1_HV.Slot0.I0	decimal(6,2)	YES		NULL	
SAB_AD1_HV.Slot2.I0	decimal(6,2)	YES		NULL	
SAB_AD1_HV.Slot4.I0	decimal(6,2)	YES		NULL	
SAB_AD1_HV.Slot6.I0	decimal(6,2)	YES		NULL	

### 21.13.11 SAB\_AD2\_HV\_Imon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
SAB_AD2_HV.Slot0.I0	decimal(6,2)	YES		NULL	

SAB_AD2_HV.Slot2.I0	decimal(6,2)	YES		NULL	
SAB_AD2_HV.Slot4.I0	decimal(6,2)	YES		NULL	
SAB_AD2_HV.Slot6.I0	decimal(6,2)	YES		NULL	

### 21.13.12 SAB\_AD2\_HV\_SlotTemp

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
SAB_AD2_HV.Slot0.Temp	decimal(4,2)	YES		NULL	
SAB_AD2_HV.Slot2.Temp	decimal(4,2)	YES		NULL	
SAB_AD2_HV.Slot4.Temp	decimal(4,2)	YES		NULL	
SAB_AD2_HV.Slot6.Temp	decimal(4,2)	YES		NULL	

### 21.13.13 DBNS\_AD1\_HV\_SlotTemp

### 21.13.14 DBNS\_AD2\_HV\_SlotTemp

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV.Slot0.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot2.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot4.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot6.Temp	decimal(4,2)	YES		NULL	

### 21.13.15 SAB\_AD1\_HV\_SlotTemp

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
SAB_AD1_HV.Slot0.Temp	decimal(4,2)	YES		NULL	
SAB_AD1_HV.Slot2.Temp	decimal(4,2)	YES		NULL	
SAB_AD1_HV.Slot4.Temp	decimal(4,2)	YES		NULL	
SAB_AD1_HV.Slot6.Temp	decimal(4,2)	YES		NULL	

### 21.13.16 DBNS\_SAB\_TEMP

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_SAB_Temp_PT1	decimal(6,2)	YES		NULL	
DBNS_SAB_Temp_PT2	decimal(6,2)	YES		NULL	
DBNS_SAB_Temp_PT3	decimal(6,2)	YES		NULL	
DBNS_SAB_Temp_PT4	decimal(6,2)	YES		NULL	
DBNS_SAB_Temp_PT5	decimal(6,2)	YES		NULL	

**21.13.17 site\_table**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS	varchar(20)	NO		NULL	
LANS	varchar(20)	NO		NULL	
FARS	varchar(20)	NO		NULL	
MIDS	varchar(20)	NO		NULL	
LSH	varchar(20)	NO		NULL	
SAB	varchar(20)	NO		NULL	
DCS_GCS	varchar(20)	YES		NULL	

**21.13.18 DBNS\_MUON\_PMT\_HV\_Imon**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
MuonPMTHV.Slot0.I0	decimal(6,2)	YES		NULL	
MuonPMTHV.Slot2.I0	decimal(6,2)	YES		NULL	
MuonPMTHV.Slot4.I0	decimal(6,2)	YES		NULL	
MuonPMTHV.Slot6.I0	decimal(6,2)	YES		NULL	
MuonPMTHV.Slot8.I0	decimal(6,2)	YES		NULL	
MuonPMTHV.Slot10.I0	decimal(6,2)	YES		NULL	

**21.13.19 DBNS\_MUON\_PMT\_HV\_SlotTemp**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
MuonPMTHV.Slot0.Temp	decimal(4,2)	YES		NULL	
MuonPMTHV.Slot2.Temp	decimal(4,2)	YES		NULL	
MuonPMTHV.Slot4.Temp	decimal(4,2)	YES		NULL	
MuonPMTHV.Slot6.Temp	decimal(4,2)	YES		NULL	
MuonPMTHV.Slot8.Temp	decimal(4,2)	YES		NULL	
MuonPMTHV.Slot10.Temp	decimal(4,2)	YES		NULL	

**21.13.20 status\_table**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV	char(4)	NO		NULL	
DBNS_RPC_HV	char(4)	NO		NULL	
FARS	char(4)	NO		NULL	
Safety Interlocking	char(4)	NO		NULL	
GAS	char(4)	NO		NULL	
Background	char(4)	NO		NULL	
DCS_GCS	char(4)	NO		NULL	
DAQ_RUNINFO	char(4)	NO		NULL	

### 21.13.21 DBNS\_AD\_HV\_SlotTemp

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV.Slot0.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot1.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot2.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot3.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot4.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot5.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot6.Temp	decimal(4,2)	YES		NULL	
DBNS_AD_HV.Slot7.Temp	decimal(4,2)	YES		NULL	

### 21.13.22 dyb\_muoncal

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
IOW_CAL_LED_ID	int(5)	YES		NULL	
IOW_CAL_LED_ID_timestamp_begin	datetime	YES		NULL	
IOW_CAL_LED_ID_timestamp_end	datetime	YES		NULL	
IOW_CAL_LED_ID_duration_time	int(11)	YES		NULL	
IOW_CAL_LED_ID_Voltage	float(5,3)	YES		NULL	
IOW_CAL_LED_ID_Frequency	float(4,1)	YES		NULL	
IOW_CAL_Channel_ID	int(11)	YES		NULL	
IOW_CAL_ErrorCode	int(11)	YES		NULL	

### 21.13.23 DBNS\_ACU\_HV\_Pw

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV_Board0_Ch0	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch1	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch2	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch3	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch4	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch5	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch6	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch7	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch8	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch9	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch10	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch11	tinyint(1)	YES		NULL	

**21.13.24 DBNS\_ACU\_HV\_Imon****21.13.25 DBNS\_ACU\_HV\_Vmon**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV_Board0_Ch0	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch1	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch2	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch3	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch4	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch5	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch6	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch7	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch8	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch9	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch10	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch11	decimal(6,2)	YES		NULL	

**21.13.26 EH1\_ENV\_RadonMonitor**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
RunNumber	int(11)	YES		NULL	
CycleNumber	int(11)	YES		NULL	
RunStartTime	int(11)	YES		NULL	
LastUpdateTime	int(11)	YES		NULL	
RunEndTime	int(11)	YES		NULL	
Temperature	int(11)	YES		NULL	
Humidity	int(11)	YES		NULL	
Rn222Conc._Po218	int(11)	YES		NULL	
Rn222Conc._Po218_StatErr	int(11)	YES		NULL	
Rn222Conc._Po214	int(11)	YES		NULL	
Rn222Conc._Po214_StatErr	int(11)	YES		NULL	
LiveTime	int(11)	YES		NULL	
AreaA	int(11)	YES		NULL	
AreaB	int(11)	YES		NULL	
AreaC	int(11)	YES		NULL	
AreaD	int(11)	YES		NULL	

**21.13.27 DBNS\_AD1\_LidSensor****21.13.28 DBNS\_AD2\_LidSensor**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
Ultrasonic_GdLS	decimal(6,2)	YES		NULL	
Ultrasonic_LS	decimal(6,2)	YES		NULL	
Temp_GdLS	decimal(6,2)	YES		NULL	
Temp_LS	decimal(6,2)	YES		NULL	

Tiltx_Sensor1	decimal(6,2)	YES		NULL	
Tilty_Sensor1	decimal(6,2)	YES		NULL	
Tiltx_Sensor2	decimal(6,2)	YES		NULL	
Tilty_Sensor2	decimal(6,2)	YES		NULL	
Tiltx_Sensor3	decimal(6,2)	YES		NULL	
Tilty_Sensor3	decimal(6,2)	YES		NULL	
Capacitance_GdLS	decimal(6,2)	YES		NULL	
Capacitance_Temp_GdLS	decimal(6,2)	YES		NULL	
Capacitance_LS	decimal(6,2)	YES		NULL	
Capacitance_Temp_LS	decimal(6,2)	YES		NULL	
Capacitance_MO	decimal(6,2)	YES		NULL	
Capacitance_Temp_MO	decimal(6,2)	YES		NULL	
PS_Output_V	decimal(6,2)	YES		NULL	
PS_Output_I	decimal(6,2)	YES		NULL	

**21.13.29 DBNS\_AD1\_VME**

**21.13.30 DBNS\_AD2\_VME**

**21.13.31 DBNS\_IW\_VME**

**21.13.32 DBNS\_Muon\_PMT\_VME**

**21.13.33 DBNS\_OW\_VME**

**21.13.34 DBNS\_RPC\_VME**

**21.13.35 SAB\_AD1\_VME**

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
Voltage_5V	decimal(6,2)	YES		NULL	
Current_5V	decimal(6,2)	YES		NULL	
Voltage_N5V2	decimal(6,2)	YES		NULL	
Current_N5V2	decimal(6,2)	YES		NULL	
Voltage_12V	decimal(6,2)	YES		NULL	
Current_12V	decimal(6,2)	YES		NULL	
Voltage_N12V	decimal(6,2)	YES		NULL	
Current_N12V	decimal(6,2)	YES		NULL	
Voltage_3V3	decimal(6,2)	YES		NULL	
Current_3V3	decimal(6,2)	YES		NULL	
Temperature1	decimal(6,2)	YES		NULL	
Temperature2	decimal(6,2)	YES		NULL	
Temperature3	decimal(6,2)	YES		NULL	
Temperature4	decimal(6,2)	YES		NULL	
Temperature5	decimal(6,2)	YES		NULL	
Temperature6	decimal(6,2)	YES		NULL	
Temperature7	decimal(6,2)	YES		NULL	
...					
FanTemperature	decimal(6,2)	YES		NULL	
Fanspeed	decimal(6,2)	YES		NULL	

PowerStatus	tinyint(1)	YES		NULL	
-------------	------------	-----	--	------	--

### 21.13.36 DBNS\_AD1\_HV

### 21.13.37 DBNS\_AD2\_HV

### 21.13.38 SAB\_AD1\_HV\_Vmon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
L8C3R8	decimal(6,2)	YES		NULL	
L8C3R7	decimal(6,2)	YES		NULL	
L8C3R6	decimal(6,2)	YES		NULL	
L8C3R5	decimal(6,2)	YES		NULL	
L8C3R4	decimal(6,2)	YES		NULL	
L8C3R3	decimal(6,2)	YES		NULL	
L8C3R2	decimal(6,2)	YES		NULL	
L8C3R1	decimal(6,2)	YES		NULL	
L8C2R8	decimal(6,2)	YES		NULL	
L8C2R7	decimal(6,2)	YES		NULL	
L8C2R6	decimal(6,2)	YES		NULL	
L8C2R5	decimal(6,2)	YES		NULL	
L8C2R4	decimal(6,2)	YES		NULL	
L8C2R3	decimal(6,2)	YES		NULL	
L8C2R2	decimal(6,2)	YES		NULL	
L8C2R1	decimal(6,2)	YES		NULL	
L8C1R8	decimal(6,2)	YES		NULL	
...					
L1C1R3	decimal(6,2)	YES		NULL	
L1C1R2	decimal(6,2)	YES		NULL	
L1C1R1	decimal(6,2)	YES		NULL	

### 21.13.39 SAB\_AD2\_HV\_Vmon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
L1C1R1	decimal(6,2)	YES		NULL	
L1C1R2	decimal(6,2)	YES		NULL	
L1C1R3	decimal(6,2)	YES		NULL	
L1C1R4	decimal(6,2)	YES		NULL	
L1C1R5	decimal(6,2)	YES		NULL	
L1C1R6	decimal(6,2)	YES		NULL	
L1C1R7	decimal(6,2)	YES		NULL	
L1C1R8	decimal(6,2)	YES		NULL	
L1C2R1	decimal(6,2)	YES		NULL	
L1C2R2	decimal(6,2)	YES		NULL	
L1C2R3	decimal(6,2)	YES		NULL	
L1C2R4	decimal(6,2)	YES		NULL	
L1C2R5	decimal(6,2)	YES		NULL	
L1C2R6	decimal(6,2)	YES		NULL	

L1C2R7	decimal(6,2)	YES		NULL	
L1C2R8	decimal(6,2)	YES		NULL	
L1C3R1	decimal(6,2)	YES		NULL	
...					
L8C3R6	decimal(6,2)	YES		NULL	
L8C3R7	decimal(6,2)	YES		NULL	
L8C3R8	decimal(6,2)	YES		NULL	

### 21.13.40 SAB\_AD2\_HV\_Pw

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
L1C1R1	tinyint(1)	YES		NULL	
L1C1R2	tinyint(1)	YES		NULL	
L1C1R3	tinyint(1)	YES		NULL	
L1C1R4	tinyint(1)	YES		NULL	
L1C1R5	tinyint(1)	YES		NULL	
L1C1R6	tinyint(1)	YES		NULL	
L1C1R7	tinyint(1)	YES		NULL	
L1C1R8	tinyint(1)	YES		NULL	
L1C2R1	tinyint(1)	YES		NULL	
L1C2R2	tinyint(1)	YES		NULL	
L1C2R3	tinyint(1)	YES		NULL	
L1C2R4	tinyint(1)	YES		NULL	
L1C2R5	tinyint(1)	YES		NULL	
L1C2R6	tinyint(1)	YES		NULL	
L1C2R7	tinyint(1)	YES		NULL	
L1C2R8	tinyint(1)	YES		NULL	
L1C3R1	tinyint(1)	YES		NULL	
...					
L8C3R6	tinyint(1)	YES		NULL	
L8C3R7	tinyint(1)	YES		NULL	
L8C3R8	tinyint(1)	YES		NULL	

### 21.13.41 DBNS\_AD1\_HVPw

### 21.13.42 DBNS\_AD2\_HV\_Pw

### 21.13.43 SAB\_AD1\_HV\_Pw

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
L8C3R8	tinyint(1)	YES		NULL	
L8C3R7	tinyint(1)	YES		NULL	
L8C3R6	tinyint(1)	YES		NULL	
L8C3R5	tinyint(1)	YES		NULL	
L8C3R4	tinyint(1)	YES		NULL	
L8C3R3	tinyint(1)	YES		NULL	
L8C3R2	tinyint(1)	YES		NULL	
L8C3R1	tinyint(1)	YES		NULL	

L8C2R8	tinyint(1)	YES		NULL	
L8C2R7	tinyint(1)	YES		NULL	
L8C2R6	tinyint(1)	YES		NULL	
L8C2R5	tinyint(1)	YES		NULL	
L8C2R4	tinyint(1)	YES		NULL	
L8C2R3	tinyint(1)	YES		NULL	
L8C2R2	tinyint(1)	YES		NULL	
L8C2R1	tinyint(1)	YES		NULL	
L8C1R8	tinyint(1)	YES		NULL	
...					
L1C1R3	tinyint(1)	YES		NULL	
L1C1R2	tinyint(1)	YES		NULL	
L1C1R1	tinyint(1)	YES		NULL	

#### 21.13.44 DBNS\_MUON\_PMT\_HV\_Vmon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DCIU3G	decimal(6,2)	YES		NULL	
DCIU3F	decimal(6,2)	YES		NULL	
DCIU3E	decimal(6,2)	YES		NULL	
DCIU3D	decimal(6,2)	YES		NULL	
DCIU3C	decimal(6,2)	YES		NULL	
DCIU3B	decimal(6,2)	YES		NULL	
DCIU3A	decimal(6,2)	YES		NULL	
DCIU39	decimal(6,2)	YES		NULL	
DCIU38	decimal(6,2)	YES		NULL	
DCIU37	decimal(6,2)	YES		NULL	
DCIU36	decimal(6,2)	YES		NULL	
DCIU35	decimal(6,2)	YES		NULL	
DCIU34	decimal(6,2)	YES		NULL	
DCIU33	decimal(6,2)	YES		NULL	
DCIU32	decimal(6,2)	YES		NULL	
DCIU31	decimal(6,2)	YES		NULL	
DCIU24	decimal(6,2)	YES		NULL	
...					
DVIA13	decimal(6,2)	YES		NULL	
DVIA12	decimal(6,2)	YES		NULL	
DVIA11	decimal(6,2)	YES		NULL	

#### 21.13.45 DBNS\_MUON\_PMT\_HV\_Pw

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DCIU3G	tinyint(1)	YES		NULL	
DCIU3F	tinyint(1)	YES		NULL	
DCIU3E	tinyint(1)	YES		NULL	
DCIU3D	tinyint(1)	YES		NULL	
DCIU3C	tinyint(1)	YES		NULL	
DCIU3B	tinyint(1)	YES		NULL	
DCIU3A	tinyint(1)	YES		NULL	

DCIU39	tinyint(1)	YES	NULL	
DCIU38	tinyint(1)	YES	NULL	
DCIU37	tinyint(1)	YES	NULL	
DCIU36	tinyint(1)	YES	NULL	
DCIU35	tinyint(1)	YES	NULL	
DCIU34	tinyint(1)	YES	NULL	
DCIU33	tinyint(1)	YES	NULL	
DCIU32	tinyint(1)	YES	NULL	
DCIU31	tinyint(1)	YES	NULL	
DCIU24	tinyint(1)	YES	NULL	
...				
DVIA13	tinyint(1)	YES	NULL	
DVIA12	tinyint(1)	YES	NULL	
DVIA11	tinyint(1)	YES	NULL	

### 21.13.46 DBNS\_AD\_HV\_Imon

### 21.13.47 DBNS\_AD\_HV\_Vmon

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV_Board0_Ch0	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch1	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch2	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch3	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch4	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch5	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch6	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch7	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch8	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch9	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch10	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch11	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch12	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch13	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch14	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch15	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board0_Ch16	decimal(6,2)	YES		NULL	
...					
DBNS_AD_HV_Board7_Ch45	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board7_Ch46	decimal(6,2)	YES		NULL	
DBNS_AD_HV_Board7_Ch47	decimal(6,2)	YES		NULL	

### 21.13.48 DBNS\_AD\_HV\_Pw

id	int(10) unsigned	NO	PRI	NULL	
date_time	datetime	NO	MUL	NULL	
DBNS_AD_HV_Board0_Ch0	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch1	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch2	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch3	tinyint(1)	YES		NULL	

DBNS_AD_HV_Board0_Ch4	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch5	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch6	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch7	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch8	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch9	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch10	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch11	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch12	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch13	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch14	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch15	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board0_Ch16	tinyint(1)	YES		NULL	
...					
DBNS_AD_HV_Board7_Ch45	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board7_Ch46	tinyint(1)	YES		NULL	
DBNS_AD_HV_Board7_Ch47	tinyint(1)	YES		NULL	
-----	-----	-----	-----	-----	-----

## 21.14 Non DBI access to DBI and other tables

- Summary of Non DBI approaches
  - Python ORMs (Django, SQLAlchemy)
  - ROOT TSQL
  - High Performance Approaches
- SQLAlchemy access to DBI tables with NonDbi

Standard access to the content of `offline_db` (eg for analysis) should be made using DBI, DybDbi or via services that use these. However some usage of the content is better achieved without DBI.

This is not contrary to the rules *Rules for Code that writes to the Database* as although all writing to `offline_db` **must** use DBI, reading from `offline_db` can use whatever approach works best for the application.

**Warning:** Non-DBI access to DBI tables is for **READING ONLY**

Examples:

1. monitoring historical variations, for example of DataQuality paramters or monitored temperatures
2. presenting tables (eg ODM)

Reading from DBI is designed around getting the results for a particular context (at a particular time). When the usage does not fit into this pattern alternative access approaches should be considered.

### DBI Extended Context

DBI Extended Context queries allows full control of the validity portion of the DBI query. As control of the validity query is the central point of DBI, this means that DBI is then not helping much. Thus if your application revolves around using DBI extended context queries you may find that alternate approaches are more efficient and straightforward.

## 21.14.1 Summary of Non DBI approaches

### Python ORMs (Django, SQLAlchemy)

Object relational mappers (ORMs) provide flexible and simple access to Database content, providing row entries as python objects. It is also possible to map to joins between tables with SQLAlchemy.

Note however a limitation of Django, it does not support composite primary keys. As DBI uses composite primary keys (SEQNO, ROW\_COUNTER) for payload tables, these cannot be mapped to Django ORM objects in the general case. However if ROW\_COUNTER only ever takes one value the mapping can be kludged to work.

SQLAlchemy does not have this limitation. The `dybgaudi:Database/NonDbi` package provides some infrastructure that facilitates access to DBI tables with SQLAlchemy. For example:

```
from NonDbi import session_
session = session_("tmp_offline_db")
YReactor = session.dbikls_("Reactor")    ## class mapped to join of payload and validity tables
n = session.query(YReactor).count()
a = session.query(YReactor).filter(YReactor.SEQNO==1).one()    ## both payload and validity attributes
print vars(a)
```

For details examples see `NonDbi`

### ROOT TSQL

Low level access requiring raw SQL, lots of flexibility but is re-inventing the wheel.

### High Performance Approaches

When dealing with many thousands/millions of entries the above approaches are slow.

**An experimental fork (from Simon) of MySQL-python that provides NumPy arrays from MySQL queries.**

- [https://github.com/scb-/mysql\\_numpy](https://github.com/scb-/mysql_numpy)

This rather simple patch to MySQL-python succeeds to integrate the primary python tools for MySQL access and large array manipulation.

- **MySQL-Python** <http://sourceforge.net/projects/mysql-python/> basis of python ORM approaches
- **NumPy** <http://numpy.scipy.org/> high performance array manipulations
- **Matplotlib** <http://matplotlib.sourceforge.net/> plotting library based on NumPy

## 21.14.2 SQLAlchemy access to DBI tables with NonDbi

How can I access the TIMESTART for a particular run ?

```
In [1]: from NonDbi import session_

In [2]: session_??          ## read docstring + code

In [3]: session = session_("offline_db")

In [4]: YDaqRunInfo = session.dbikls_("DaqRunInfo")

In [5]: session.query(YDaqRunInfo).count()
```

```
Out [5]: 11402L
```

```
In [6]: YDaqRunInfo.<TAB>
YDaqRunInfo.AGGREGATENO          YDaqRunInfo.TIMESTART          YDaqRunInfo.__dict__          YD
YDaqRunInfo.INSERTDATE          YDaqRunInfo.VERSIONDATE        YDaqRunInfo.__dictoffset__    YD
YDaqRunInfo.ROW_COUNTER          YDaqRunInfo.__abstractmethods__ YDaqRunInfo.__doc__          YD
YDaqRunInfo.SEQNO                YDaqRunInfo.__base__           YDaqRunInfo.__eq__           YD
YDaqRunInfo.SIMMASK              YDaqRunInfo.__bases__          YDaqRunInfo.__flags__        YD
YDaqRunInfo.SITEMASK             YDaqRunInfo.__basicsize__       YDaqRunInfo.__format__       YD
YDaqRunInfo.SUBSITE              YDaqRunInfo.__call__           YDaqRunInfo.__ge__           YD
YDaqRunInfo.TASK                 YDaqRunInfo.__class__          YDaqRunInfo.__getattr__      YD
YDaqRunInfo.TIMEEND              YDaqRunInfo.__delattr__        YDaqRunInfo.__gt__           YD
```

```
In [6]: q = session.query(YDaqRunInfo)
```

```
In [7]: q
```

```
Out [7]: <sqlalchemy.orm.query.Query object at 0x920058c>
```

```
In [8]: q.count()
```

```
Out [8]: 11408L
```

```
In [9]: q[0]
```

```
Out [9]: <NonDbi.YDaqRunInfo object at 0x9214f8c>
```

```
In [11]: p vars(q[-1])
```

```
...
```

```
In [17]: q.filter_by(runNo=12400).one()
```

```
Out [17]: <NonDbi.YDaqRunInfo object at 0x91fd4ac>
```

```
In [18]: vars(q.filter_by(runNo=12400).one())
```

```
Out [18]:
```

```
{u'AGGREGATENO': -1L,
 u'INSERTDATE': datetime.datetime(2011, 8, 16, 0, 0, 53),
 u'ROW_COUNTER': 1L,
 'SEQNO': 11185L,
 u'SIMMASK': 1,
 u'SITEMASK': 127,
 u'SUBSITE': 0,
 u'TASK': 0,
 u'TIMEEND': datetime.datetime(2011, 8, 15, 23, 57, 19),
 u'TIMESTART': datetime.datetime(2011, 8, 15, 6, 55, 55),
 u'VERSIONDATE': datetime.datetime(2011, 8, 15, 6, 55, 55),
 '_sa_instance_state': <sqlalchemy.orm.state.InstanceState object at 0x91fd4cc>,
 u'baseVersion': 1L,
 u'dataVersion': 813L,
 u'detectorMask': 230L,
 u'partitionName': 'part_eh1',
 u'runNo': 12400L,
 u'runType': 'Physics',
 u'schemaVersion': 17L,
 u'triggerType': 0L}
```

```
In [19]: o = q.filter_by(runNo=12400).one()
```

```
In [21]: o.TIMESTART
```

```
Out [21]: datetime.datetime(2011, 8, 15, 6, 55, 55)
```

Note that this SQLAlchemy access to DBI tables is entirely general. For the common task of run lookups `DybDbi.IRunLookup` has dedicated functionality to allow this.

```
In [23]: import os
```

```
In [24]: os.environ['DBCONF'] = 'offline_db'
```

```
In [25]: from DybDbi import IRunLookup
```

```
In [26]: irl = IRunLookup( 12400, 12681 )
```

```
DbiRpt<GDaqRunInfo>::MakeResultPtr extended query ctor, sqlcontext: 1=1 datasql:runNo in (12400, 12681)
```

```
Using DBConf.Export to prime environment with : from DybPython import DBConf ; DBConf.Export('offline_db')
```

```
dbconf:export_to_env from $SITEROOT/../../my.cnf:~/my.cnf section offline_db
```

```
Successfully opened connection to: mysql://dybdb2.ihep.ac.cn/offline_db
```

```
This client, and MySQL server (MySQL 5.0.45-community) does support prepared statements.
```

```
DbiCascader Status:-
```

```
Status      URL
```

```
Closed      0 mysql://dybdb2.ihep.ac.cn/offline_db
```

```
In table DaqRunInfo row 0 column 4 (TRIGGERTYPE) value "0" of type Long may be truncated before storing
```

```
Caching new results: ResultKey: Table:DaqRunInfo row:GDaqRunInfo. 2 vrecs (seqno min..max;versiondate)
```

```
DbiTimer:DaqRunInfo: Query done. 2rows, 0.1Kb Cpu 0.5 , elapse 2.0
```

```
In [33]: irl[12400].vrec.contextrange
```

```
Out[33]:
```

```
|site 0x007f|sim 0x007f
```

```
2011-08-15 06:55:55.000000000Z
```

```
2011-08-15 23:57:19.000000000Z
```

## 21.15 Scraping source databases into *offline\_db*

In addition to this introductory documentation see also the API reference documentation at [Scraper](#)

- Generic Scraping Introduction
- Scraper Status
- DCS peculiarities
  - Time zones and scraping
- TODO
  - Framework Level
  - Specific Regimes
- Running Scrapers
  - Dybinst Level
  - Package Level
- Implementing Scrapers
  - Outline Steps
  - Create Scraper Module
  - Implementing `changed`
  - Implementing `propagate`
  - Generic Aggregation
  - Error Handling
- Configuring Scrapers
  - Understanding Scraper Operation
  - Catchup and Sleep Auto-Tuning
  - Configuration Mechanics
  - Configuration Tuning
- Testing Scraper Operation
  - Test Scraper With Faker
  - Faker configuration
  - Preparing Target DB for testing
  - Seeding Target Database
  - Scraper Logging
- Continuous running under supervisor
  - Initial Setup
  - Supervisorctl CLI
- Steps to Deployment
- Development Tips
  - Obtain mysqldump of DCS DB to populate fake source DB
  - Single table mysqldump for averager testing
  - Append capable mysqldumps
  - Multi-source table test
  - Start from scratch following schema changes to DCS
  - Interactive SQLAlchemy Querying

### 21.15.1 Generic Scraping Introduction

#### Pragmatic Goals

- eliminate duplication in scraper code
- make it easy to scrape new tables with little additional code to develop/debug
- use DybDbi for writing to *offline\_db*, eliminate problems arising from incomplete DBI spoofing by using real DBI for all writing

#### Assumptions/features of each scraper

- 2 databases : source and target

- *target* is represented by DybDbi generated classes
- *source(s)* are represented by SQLAlchemy mapped classes which can correspond to entries in single source tables or entries from the result of joins to multiple source tables
- one source instance corresponds to 1 or more DybDbi writes under a single DBI writer/contextrange

### 21.15.2 Scraper Status

regime	target table	notes
pmthv	GDcsPmtHv	duplicates old scraper with new framework, needs testing by Liang before deployment
	GDcsAdPmtHv	
adtemp	GDcsAdTemp	duplicates old scraper with new framework, needs testing by Liang before deployment
adlid-sensor	GDcsAdLidSensor	development started end August by David Webber
muon-calib?	GDcsMuonCalib	interest expressed by Deb Mohapatra
wppmt?	GDcsWpPmtHv	?
adgas?	?	Raymond named as responsible by Wei, doc:7005

Existing scraper modules are visible at [dybgaudi:Database/Scraper/python/Scraper](#)

Existing target table specifications [dybgaudi:Database/DybDbi/spec](#)

### 21.15.3 DCS peculiarities

*DCS tables grouped/ordered by schema*

DCS tables have the nasty habit of encoding content (stuff that should be in rows) into table and column names. As a result mapping from source to target in code must interpret these names and sometimes one row of source DCS table will become many rows of destination table.

The task of developing scrapers is much easier when:

- source and target tables are developed with scraping in mind

### Time zones and scraping

#### Local times and Databases

By their very nature of being accessible from any timezone, it is patently obvious that time stamps in Databases should never be in local time. However as this bad practice is rife in the DCS and DAQ it is pragmatically assumed that this bad practice is always followed in the DCS and DAQ DB.

Time zone conventions assumed by the generic scraper:

- All timestamps in *offline\_db* and *tmp\_offline\_db* are in UTC, hardcoded into DBI: **cannot be changed**
- All timestamps in DCS and DAQ DB are in local(Beijing) time

If the 2nd assumption is not true for your tables, you must either change it to follow the local standard of bad practice or request special handling in the scraper framework.

## 21.15.4 TODO

### Framework Level

1. scraper catchup feature needs documenting and further testing
2. DAQ support ? probably no new DAQ tables coming down pipe, but still needs DBI writing
3. confirm assumption : all DCS times local, all DBI times UTC
4. more precise testing, will fully controlled faking/scraping and comparison against expectations (not high priority as this kind of precision is not really expected from a scraper)

### Specific Regimes

1. in old scraper code : table names do not match current *offline\_db* : DcsPmtHv
2. in old scraper code : apparently no timezone handling ?

## 21.15.5 Running Scrapers

**Warning:** scraper config include source and target DBCONF, thus ensure that the corresponding entries in `~/my.cnf` are pointing at the intended Databases before running scrapers or fakers

- [Dybinst Level](#)
- [Package Level](#)

### Dybinst Level

To allow use of scrapers and fakers from a pristine environment, such as when invoked under supervisor control, a *dybinst* level interface is provided:

```
./dybinst trunk scrape adtemp_scraper
./dybinst trunk scrape pmtHV_scraper
```

The last argument specifies a named section in `$SCRAPERROOT/python/Scraper/.scraper.cfg` When testing fake entries can be written to a fake source DB using a **faker** config section, with for example:

```
./dybinst trunk scrape adtemp_faker
./dybinst trunk scrape pmtHV_faker
```

### Package Level

The *dybinst* interface has the advantage of operating from an empty environment but is not convenient for debugging/testing. When within the environment of `dybgaudi:Database/Scraper` package (included in standard DybRelease environment) it is preferable to directly use:

```
scr.py --help          ## uses a default section
scr.py -s adtemp_scraper
scr.py -s adtemp_faker
```

Examining the help is useful for seeing the config defaults for each config section:

```
scr.py -s adtemp_faker --help
scr.py -s adtemp_scraper --help
```

## 21.15.6 Implementing Scrapers

The generic scraper framework enables the addition of new scrapers with only code that is specific to the source and target tables concerned. The essential tasks are to judge sufficient change to warrant propagation and to translate from source instances to one or more target DBI table instances. Note that the source instances can be joins between multiple source tables.

- [Outline Steps](#)
- [Create Scraper Module](#)
- [Implementing changed](#)
- [Implementing propagate](#)
- [Generic Aggregation](#)
- [Error Handling](#)

### Outline Steps

1. Create *offline\_db* target table by committing a `.spec` file and building *DybDbi*, *DB Table Creation*
2. Create scraper module, implementing only the table specifics: *Create Scraper Module*
3. Test scraper operation into a copy of *offline\_db*, *Copy offline\_db to tmp\_offline\_db*

### Create Scraper Module

Scraper modules live in `dybgaudi:Database/Scraper/python/Scraper`. To eliminate duplication they only handle the specifics of transitioning source DCS/DAQ table(s) columns into target *offline\_db* table columns as specified in your `.spec`

Compare and contrast the example scraper modules:

- `dybgaudi:Database/Scraper/python/Scraper/pmthv.py` *Scraper.pmthv*
- `dybgaudi:Database/Scraper/python/Scraper/adtemp.py` *Scraper.adtemp*

Note the structure of classes, using *PmtHv* as an example:

1. *PmtHv(Regime)* umbrella sub-class
2. *PmtHvSource(list)* list of source tables (or joins of tables)
3. *PmtHvScraper(Scraper)* sub-class that implements two methods, both taking single `SourceVector sv` argument
  - (a) *changed(self,sv)* returns `True/False` indicating if there is sufficient change to justify calling the *propagate* method
  - (b) *propagate(self,sv)* converts source vector into one or more yielded target *dicts* with keys corresponding to `.spec` file attribute names
4. *PmtHvFaker(Faker)* sub-class used to Fake entries in the source DB table(s) to allow fully controlled testing

Further implementation details are documented in the API docs [Scraper](#)

## Implementing changed

The simplest changed implementation:

```
def changed(self, sv ):
    return False
```

The source vector `sv` holds 2 source instances, accessible with `sv[0]` and `sv[-1]` corresponding to the last propagated instance and the latest one. Even with a changed implementation that always returns `False` the propagate will still be called when the age differences between `sv[0]` and `sv[-1]` exceed the `maxage` configuration setting.

---

**Note:** `changed()` is not intended for age checking, instead just use config setting such as `maxage=1h` for that

---

If *Generic Aggregation* can be used it is easier and more efficient to do so. However if the required aggregation can not be performed with MySQL aggregation functions then the `changed()` method could be used to collect samples as shown in the below example. Note that `self.state` is entirely created/updated/used within the `changed` and `propagate` methods. This is just an example of how to maintain state, entirely separately from the underlying framework:

```
def changed(self, sv):

    if not hasattr(self, 'state'):          ## only on 1st call when no state
        kls = self.target.kls              ## the genDbi target class
        keys = kls.SpecKeys().aslist()
        state = dict(zip(keys, map(lambda _:0, keys)))    ## state dict with all values 0
        self.state = state

    ## work of associating source to target attributes
    for k in self.state:
        sk = ..some_fn..( k )              ## source key from target key
        ## do running aggregates min/max/avg
        self.state[k] += sv[-1][sk]

    return False    ## True if sufficient changes to warrant non-age based propagation
```

## Implementing propagate

The main work of `changed` and `propagate` is translating between the source instances eg `sv[-1]` and the target dict ready to be written using target `genDbi` class. The ease with which this can be done depends on the design of source and target.

Example implementation, when do accumulation at each changed sampling:

```
def propagate(self, sv ):
    yield self.state
```

Alternatively if do not need to accumulate over samples and want to write just based on the last values can see examples:

1. `Scraper.pmthv.PmtHvScraper`
2. `Scraper.adtemp.AdTempScraper`

## Generic Aggregation

Aggregation is configured via config keys beginning with **aggregate**. Presence of a non-empty aggregate key switches on an extra aggregation query, performed at every sample immediately after the normal entry query. The aggregate key must specify a comma delimited list naming [MySQL aggregate/group-by functions](#):

```
aggregate = avg,min,max,std
aggregate_count = count
aggregate_skips = id,date_time
aggregate_filter = Quality != 0
```

Meanings of the settings:

setting	notes
aggregate	comma delimited list of MySQL aggregation functions
aggregate_count	name of attribute that holds the sample count, default <b>count</b>
aggregate_skips	comma delimited attributes to skip aggregating, default <b>None</b>
aggregate_filter	SQL where clause applied in addition to time filtering, default <b>None</b>

**Note:** Most MySQL group\_by functions do not work well with times, if that is a major problem workarounds could be developed

The functions are called for every source attribute within a single query that is performed on the source DB after the simple row query. The results are provided in the **aggd** dict with keys such as DBNS\_SAB\_Temp\_PT1\_avg, DBNS\_SAB\_Temp\_PT1\_min etc..

The aggregation query is over all source DB entries within a timerange that excludes the time of the last instance:

```
sv[0].date_time <= t < sv[-1].date_time
```

The **aggd** dict are available from the `sv[0].aggd` and `sv[-1].aggd` within the `changed` and `propagate` methods, but existence of `sv[0].aggd` should be checked as will not be available at startup:

```
aggz = sv[0].aggd
if aggz:
    for k,v in aggz.items():
        print k,v
else:
    print "no aggz at startup"

aggd = sv[-1].aggd
assert aggd, "missing aggd - this should always be present"
for k,v in aggd.items():
    print k,v
```

When using the docs virtual python *Build Instructions for Sphinx based documentation* the aggregate can be dumped `print str(aggd)` as an rst table like:

att [2]	avg	max	min	std
DBNS_SAB_Temp_PT1	48.500000	49.00	48.00	0.5
DBNS_SAB_Temp_PT2	28.500000	29.00	28.00	0.5
DBNS_SAB_Temp_PT3	38.500000	39.00	38.00	0.5
DBNS_SAB_Temp_PT4	48.500000	49.00	48.00	0.5
DBNS_SAB_Temp_PT5	58.500000	59.00	58.00	0.5
date_time	2.01102010008e+13	2011-02-01 00:08:10	2011-02-01 00:08:00	5.0
id	48.5000	49	48	0.5

## Error Handling

Possible error cases that must be handled:

- aggregation query may yield zero samples, resulting in the configured **aggregate\_count** value coming back as zero and all aggregates being **None**
  - occurs when the configured **aggregate\_filter** (typically a source quality requirement) results in no entries
  - most likely to occur on the first sample after a propagation
  - having a very short **interval** compared to the source **heartbeat** will make this more likely to occur
  - if not trapped the scraper will crash when attempting to coerce **None** into the float/int attributes of the DybDbi instance, eg:

```
File "/home/dwebber/NuWa/NuWa-trunk/dybgaudi/InstallArea/python/DybDbi/wrap.py", line 98, in
    Set( instance, v )
TypeError: void GDcsAdLidSensor::SetTemp_LS_avg(double Temp_LS_avg) => could not convert arg
```

- options to handle this is under consideration
  - replace the None with an **aggregate\_none** configured value

## 21.15.7 Configuring Scrapers

- Understanding Scraper Operation
- Catchup and Sleep Auto-Tuning
- Configuration Mechanics
- Configuration Tuning

### Understanding Scraper Operation

#### heartbeat parameter

The source DB updating period is not under the control of the scraper, however scraper configuration should include this approximate **heartbeat** setting in order to inform the scraper to allow appropriate sleep tuning.

Scrapers distinguish between the notions:

1. **actual time** ticking by, at which actual DB queries are made
2. **DB time** populated by *date\_time* stamps on DB entries

This allows the scraper to catch up, on being restarted after a hiatus and not substantially impact the resulting scraped target table.

Ascii art, each lines corresponding to a sample:

```
tc0      tc1
|        |
|1       |
|1 2     |
|1 . 3   |   propagation can be triggerd for any
|1 . . 4 |   of these if sufficient change in value or date_time
|1 . . . 5|
```

```
|          |6  
|          |6 7  
|          |6 . 8  
|          |6 . . 9  
|          |6 . . . a  
|          |6 . . . . b  
|          |6 . . . . . c
```

The scrapers region of interest in DB time is controlled by:

- the time cursor `tcursor`
- `date_time` of last collected instance in the source vector

The first entry beyond the `tcursor` and any already collected entries is read. In this way the scraper is always looking for the next entry. Following a propagation the `tcursor` is moved ahead to the time of the last propagated entry plus the **interval**.

---

**Note:** to avoid excessive querying scraper parameters must be tuned, *Configuration Tuning*

---

Sampling activity in actual time is controlled by:

#### offset mechanics and interplay with aggregation

Using an `offset = N` where  $N > 0$  effectively means the scraper only sees every  $N$ th entry in the source database. This does not effect the source DB samples that contribute to the aggregation, all source samples that pass the `aggregate_filter` contribute to the aggregation. The `offset` however directly reduces the frequency with which aggregate (and normal) sampling is performed.

- **sleep** config setting, initial value for sleep that is subsequently tuned depending on lag
- **heartbeat** config setting, guidance to scraper regards source updating period : used to constrain other parameters in sleep tuning
- **offset** config setting, allows skipping of source entries : default of zero reads all entries, set to 10 to only read every 10th

Propagation is controlled by:

- value changes between source vector instances, optionally parameterized by the **threshold** config setting
- the **maxage** config setting compared to the difference in `date_time` between source vector instances

Features of this approach:

1. reproducible re-running of the scraper (target entries made should depend on the source not details of scraper running)
2. allows the scraper to catch up with missed entries after a hiatus
3. realistic testing

The **heartbeat** setting should correspond approximately to the actual source table updating period. (**sleep** setting should be the same as the **heartbeat** , it is subsequently tuned depending on detected lags behind the source).

See the below *Scraper Logging* section to see how this works in practice.

---

**Note:** some of the config parameters can probably be merged/eliminated, however while development of scrapers is ongoing retaining flexibility is useful

---

## Catchup and Sleep Auto-Tuning

Relevant config parameters:

parameter	notes
tunesleep-mod	how often to tune the sleep time, default of 1 tunes after every propagation, 10 for every 10th etc..
interval	quantum of DB time, controls <code>tcursor</code> step after propagation
offset	integer sampling tone down, default of 0 samples almost all source entries. 10 would sample only every 10th entry.
heartbeat	guidance regarding the raw source tables updating period (without regard for any <b>offset</b> ) used to control sleep tuning
timefloor	time prior to which the scraper has no interest

A restarted scrapers first actions include checking the DBI validity table in the target DB to determine the **target last validity**, a DBI Validity Record which allows the **tcursor** from the prior run of the scraper to be recovered. Hence the scraper determines where it is up to and resumes from that point.

Following some propagations the scraper queries to determine `date_time` of the last entry in the source table. Comparing this with the **tcursor** yields a lag for each source `Scraper.base.sourcevector.SourceVector.lag()`, the maximum lag over all sources `Scraper.base.Scraper.maxlag()` is obtained.

The extent of this maximum lag time is translated into units of the *effective heartbeat* (namely `heartbeat*(offset+1)`). This number of effective heartbeats lag is used within `Scraper.base.Scraper.tunesleep()` to adjust the sleep time. This algorithm is currently very primitive; it may need to be informed by real world operational experience.

## Configuration Mechanics

All scrapers are configured from a single config file, which is arranged into sections for each scraper/faker. The path of the config file can be controlled by `SCRAPER_CFG`, the default value:

```
echo $$SCRAPER_CFG      ## path of default config file
--> $SITEROOT/dybgaudi/Database/Scraper/python/Scraper/.scraper.cfg
--> $SCRAPERROOT/python/Scraper/.scraper.cfg
```

Generality of scraper frontends is achieved by including a specification of the *Regime* subclass with the configuration, for example an extract from:

```
[adtemp_scraper]

regime = Scraper.adtemp:AdTemp
kls = GDcsAdTemp
mode = scraper
source = fake_dcs
target = tmp_offline_db

interval = 10s
sleep = 3s
heartbeat = 3s
offset = 0
maxage = 10m
threshold = 1.0
maxiter = 0

dbi_loglevel = INFO
```

Settings defining **what and where**:

<b>regime</b>	python dotted import path and Regime subclass name
<b>cls</b>	target DybDbi class name
<b>mode</b>	must be <i>scraper</i> , can be <i>faker</i> for a Faker
<b>source</b>	name of dbconf section in <code>~/ .my .cnf</code> , pointing to origin DB typically <code>fake_dcs</code>
<b>target</b>	name of dbconf section in <code>~/ .my .cnf</code> , pointing to DBI database typically <code>tmp_offline_db</code> while testing

Settings impacting **how and when**:

<b>interval</b>	DB time quantum, minimum sampling step size (DB time)
<b>heartbeat</b>	approximate source table updating period, used by sleep tuning.
<b>offset</b>	Integer specifying source table offsets. The default of 0 reads all source entries, 1 for every other, 10 for every 10th, etc.. <b>This is the best setting to increase to reduce excessive sampling.</b>
<b>sleep</b>	Initial period to sleep in the scraper loop (is subsequently auto-tuned depending on lag to the source)
<b>maxage</b>	maximum period after which entries are propagated even if unchanged (DB time)
<b>threshold</b>	optional parameter accessed within scrapers via <code>self.threshold</code> , typically used within <code>def changed()</code> method
<b>maxiter</b>	number of iterations to scrape, usually 0 for no limit

Time durations such as **interval**, **sleep** and **maxage** are all expressed with strings such as `10s`, `1m` or `1h` representing periods in seconds, minutes or hours.

Other configuration settings for scrapers:

```
## time before which the scraper is not interested, used to limit expense of lastvld query at start
timefloor = 2010-01-01 00:00:00

## see below section on seeding the target, seeding is not allowed when targeting offline_db
seed_target_tables = True
seed_timestart = 2011-02-01 00:00:00
seed_timeend = 2011-02-01 00:01:00
```

See `Scraper.base.main()` for further details on configuration.

### Configuration Tuning

Consider scraping wildly varying source quantities that always leads to a propagation, the 1st entry beyond the `tcursor` would immediately be propagated and the `tcursor` moved ahead to the time of the last propagated entry plus the `interval` leading to another propagation of the 1st entry beyond the new `tcursor`.

In this situation:

- **offset** could be increased to avoid sampling all source entries
- **interval** must be tuned to achieve desired propagation frequency/value tracking

Alternatively consider almost constant source quantities, that never cause a `def changed()` to return `True`. In this case samples are dutifully made of entries beyond the `tcursor` until the time difference between the first and last exceeded the **maxage** and a propagation results and `tcursor` is moved forwards to the time of the last propagated entry plus the **interval**.

In this situation:

- **maxage** dominates what is scraped
- **offset** should be increased to avoid pointless unchanged sampling within the `maxage` period

**Note:** setting **offset** only impacts the raw querying, it does not influence the aggregate query which aggregates over all source entries within the time range defined by the raw queries.

## 21.15.8 Testing Scraper Operation

- Test Scraper With Faker
- Faker configuration
- Preparing Target DB for testing
- Seeding Target Database
- Scraper Logging

### Test Scraper With Faker

Fakers exist in order allow testing of Scrapers via fully controlled population of a fake source DB, typically `fake_dcs`. At each faker iteration an instance for each source class (an SQLAlchemy dynamic class) is created and passed to the fakers `fake` method, for example:

```
class AdTempFaker(Faker):
    def fake(self, inst, id, dt):
        """
        Invoked from base class, sets source instance attributes to form a fake

        :param inst: source instance
        :param id: id to assign to the instance
        """
        if dt==None:
            dt = datetime.now()
        for k,v in inst.asdict.items():
            if k == 'id':
                setattr( inst, k, id )
            elif k == 'date_time':
                setattr( inst, k, dt )
            else:
                setattr( inst, k, float(id%10))    ## silly example of setting attribute values based
```

This allows the attributes of the fake instance to be set as desired. It is necessary to set the `id` and `date_time` attributes as shown to mimic expect source DB behaviour.

### Faker configuration

Fakers are configured similarly to scrapers. An example configuration:

```
[adtemp_faker]

regime = Scraper.adtemp:AdTemp
mode = faker

source = fake_dcs
faker_dropsrc = True

timeformat = %Y-%m-%d %H:%M:%S
faker_timestart = 2011-02-01 00:00:00
profile = modulo_ramp
interval = 10s
sleep = 3s
maxiter = 0
```

**Warning:** When running in mode = faker the faker\_dropsrc = True wipes the DB pointed to by source = fake\_dcs

The faker\_dropsrc=True key causes the fake source DB to be dropped and then recreated from a mysql dump file ~/fake\_dcs.sql that must reside with \$HOME. This dropping and reloading is done at each start of the **faker**.

### Preparing Target DB for testing

The database specified in the *target* config parameter of scrapers must be existing and accessible to the scraper identity, as defined in the ~/.my.cnf. Create the target DB and grant permissions with:

```
mysql> create database offline_db_dummy
mysql> grant select,insert,create,drop,lock tables,delete on offline_db_dummy.* to 'dayabay'@'%' iden
```

Privileges are needed for DBI operations used by the Scraper:

priv	first fail without it
lock tables	locks around updating LOCALSEQNO
insert	inserting ('*', 0) into LOCALSEQNO
delete	LASTUSEDSEQNO updating deletes then inserts

### Seeding Target Database

Scraping requires an entry in the target DB table in order to discern where in time the scraping is up to. When testing into empty DB/Tables a seed entry needs to be planted using DybDbi for each source table. This can be done using the config settings like:

```
seed_target_tables = True
seed_timestart = 2011-02-01 00:00:00
seed_timeend = 2011-02-01 00:01:00
```

Together with implementing the def seed(src) : method in the scraper to return a dict of attributes appropriate to the genDbi target class. If the target has many attributes, a programmatic approach can be used, eg starting from:

```
In [1]: from DybDbi import GDcsAdLidSensor as kls

In [2]: kls.SpecKeys().aslist()
Out[2]:
['PhysAdId',
 'Ultrasonic_GdLS',
 'Ultrasonic_GdLS_SD',
```

```
'Ultrasonic_GdLS_min',
'Ultrasonic_GdLS_max',
'Ultrasonic_LS',
'Ultrasonic_LS_SD',
'Ultrasonic_LS_min',
'Ultrasonic_LS_max',
...
```

## Scraper Logging

The bulk of the output comes from the `smry` method of `Scraper.base.sourcevector` which displays the *id* and *date\_time* of the source instances held by the `SourceVector` as well as the **time cursor** of the source vector which corresponds to the time of last propagation. An extract from a scraper log, showing the startup:

```
INFO:Scraper.base.scraper:timecursor(local) {'subsite': 1, 'sitemask': 32} Tue Feb 1 00:01:00 2011
INFO:Scraper.base.sourcevector:SV 1 (6,) 2011-02-01 00:01:00 partial notfull (00:01:00 +
INFO:Scraper.base.sourcevector:SV 2 (6, 7) 2011-02-01 00:01:00 full unchanged (00:01:00 00
INFO:Scraper.base.sourcevector:SV 3 (6, 8) 2011-02-01 00:01:00 full unchanged (00:01:00 00
INFO:Scraper.base.sourcevector:SV 4 (6, 9) 2011-02-01 00:01:00 full unchanged (00:01:00 00
INFO:Scraper.base.sourcevector:SV 5 (6, 10) 2011-02-01 00:01:00 full unchanged (00:01:00 00
INFO:Scraper.base.sourcevector:SV 6 (6, 11) 2011-02-01 00:01:00 full unchanged (00:01:00 00
INFO:Scraper.base.sourcevector:SV 7 (6, 12) 2011-02-01 00:01:00 full unchanged (00:01:00 00
INFO:Scraper.base.sourcevector:SV 8 (6, 13) 2011-02-01 00:01:00 full overage (00:01:00 00
Warning in <TClass::TClass>: no dictionary for class DbiWriter<GDcsPmtHv> is available
Proceeding despite Non-unique versionDate: 2011-01-31 16:01:00 collides with that of SEQNO: 2 for tak
INFO:Scraper.base.scraper: 0 tune detects maxlag 9 minutes behind namely 59 intervals ... sleep 0:00
INFO:Scraper.base.sourcevector:SV 9 (13, 14) 2011-02-01 00:02:20 full unchanged (00:02:10 00
INFO:Scraper.base.sourcevector:SV 10 (13, 15) 2011-02-01 00:02:20 full unchanged (00:02:10 00
INFO:Scraper.base.sourcevector:SV 11 (13, 16) 2011-02-01 00:02:20 full unchanged (00:02:10 00
INFO:Scraper.base.sourcevector:SV 12 (13, 17) 2011-02-01 00:02:20 full unchanged (00:02:10 00
INFO:Scraper.base.sourcevector:SV 13 (13, 18) 2011-02-01 00:02:20 full unchanged (00:02:10 00
INFO:Scraper.base.sourcevector:SV 14 (13, 19) 2011-02-01 00:02:20 full unchanged (00:02:10 00
INFO:Scraper.base.sourcevector:SV 15 (13, 20) 2011-02-01 00:02:20 full overage (00:02:10 00
INFO:Scraper.base.scraper: 1 tune detects maxlag 8 minutes behind namely 52 intervals ... sleep 0:00
INFO:Scraper.base.sourcevector:SV 16 (20, 21) 2011-02-01 00:03:30 full unchanged (00:03:20 00
INFO:Scraper.base.sourcevector:SV 17 (20, 22) 2011-02-01 00:03:30 full unchanged (00:03:20 00
INFO:Scraper.base.sourcevector:SV 18 (20, 23) 2011-02-01 00:03:30 full unchanged (00:03:20 00
INFO:Scraper.base.sourcevector:SV 19 (20, 24) 2011-02-01 00:03:30 full unchanged (00:03:20 00
INFO:Scraper.base.sourcevector:SV 20 (20, 25) 2011-02-01 00:03:30 full unchanged (00:03:20 00
INFO:Scraper.base.sourcevector:SV 21 (20, 26) 2011-02-01 00:03:30 full unchanged (00:03:20 00
INFO:Scraper.base.sourcevector:SV 22 (20, 27) 2011-02-01 00:03:30 full overage (00:03:20 00
INFO:Scraper.base.scraper: 2 tune detects maxlag 7 minutes behind namely 45 intervals ... sleep 0:00
INFO:Scraper.base.sourcevector:SV 23 (27, 28) 2011-02-01 00:04:40 full unchanged (00:04:30 00
INFO:Scraper.base.sourcevector:SV 24 (27, 29) 2011-02-01 00:04:40 full unchanged (00:04:30 00
```

This is with config settings:

```
interval = 10s
sleep = 3s
maxage = 1m
threshold = 1.0
maxiter = 0
task = 0
```

---

**Note:** while testing it is convenient to sample/propagate far faster than would be appropriate in production

---

Points to note:

1. initially the source vector contains only one sample and is marked `partial/notfull`, there is no possibility of propagation
2. at the 2nd sample (10s later in DB time, **not necessarily the same in real time**) the source vector becomes `full/unchanged` and the source `id` held are (6,7) at times (00:01:00 00:01:10)
3. for the 3rd to 7th samples the `sv[0]` stays the same but `sv[-1]` gets replaced by new sampled instances
4. at the 8th sample a sufficient change between `sv[0]` and `sv[-1]` is detected (in this example due to `maxage = 1m` being exceeded) leading to a **PROCEED** which indicates a propagation into the target DB
5. at the 9th sample, the `sv[0]` is replaced by the former `sv[-1]` which led to the propagation, correspondingly note the change in `id` held to (13,14) and times (00:02:10 00:02:20)

In this case propagation as marked by the **PROCEED** is occurring due to **overage** arising from config. If aggregation were to be configured in this example the aggregation would have been performed:

1. at 2nd sample for all entries between (00:01:00 00:01:10)
2. for 3rd to 7th samples for all entries between (00:01:00 00:01:20) and so on
3. at the 8th sample the aggregation is between (00:01:00 00:02:10) which would have then been propagated
4. at the 9th sample the aggregation is between (00:02:10 00:02:20) with starting point corresponding to the former endpoint

## 21.15.9 Continuous running under supervisor

- Initial Setup
- Supervisorctl CLI

### Initial Setup

Prepare example supervisor config file using `-S` option:

```
./dybinst -S /tmp/adtemp_scraper.ini trunk scrape adtemp_scraper
sudo cp /tmp/adtemp_scraper.ini /etc/conf/
```

Prepare the configs for all named section of the file using special cased **ALL** argument:

```
mkdir /tmp/scr
./dybinst -S /tmp/scr trunk scrape ALL
sv- ; sudo cp /tmp/scr/*.ini $(sv-confdir)          ## when using option sv- bash functions
```

NB the location to place supervisor `.ini` depends on details of the supervisor installation and in particular settings in `supervisord.conf`, for background see <http://supervisord.org/configuration.html> The config simply specifies details of how to run the command, and can define the expected exit return codes that allow auto-restarting. For example:

```
[program:adtemp_scraper]
environment=HOME='/home/scrapper',SCRAPER_CFG='/home/scrapper/adtemp_scraper_production.cfg'
directory=/data1/env/local/dyb
command=/data1/env/local/dyb/dybinst trunk scrape adtemp_scraper
redirect_stderr=true
```

```

redirect_stdout=true
autostart=true
autorestart=true
priority=999
user=blyth

```

**Note:**

1. program name `adtemp_scraper`, which is used in `supervisorctl` commands to control and examine the process.
2. environment setting pointing the production scraper to read config from a separate file:

```
``environment=HOME='/home/scraper',SCRAPER_CFG='/home/scraper/adtemp_scraper_production.cfg'``
```

**NB the single quotes** which is a workaround for `svenvparsebug` needed in some supervisor versions.

**Supervisorctl CLI**

Start the process from `supervisorctl` command line as shown:

```

[blyth@belle7 conf]$ sv                                     ## OR supervisorctl if not using sv- bash functions
dybslv             RUNNING   pid 2990, uptime 5:39:59
hgweb              RUNNING   pid 2992, uptime 5:39:59
mysql              RUNNING   pid 2993, uptime 5:39:59
nginx              RUNNING   pid 2991, uptime 5:39:59

```

N> help

default commands (type help <topic>):

=====

```

add      clear  fg      open  quit  remove  restart  start  stop  update
avail   exit   maintail  pid  reload  reread  shutdown  status  tail  version

```

N> reread

```

adtemp_faker: available
adtemp_scraper: available
pmthv_faker: available
pmthv_scraper: available

```

N> avail

```

adtemp_faker          avail      auto      999:999
adtemp_scraper        avail      auto      999:999
dybslv                in use    auto      999:999
hgweb                 in use    auto      999:999
mysql                 in use    auto      999:999
nginx                 in use    auto      999:999
pmthv_faker           avail      auto      999:999
pmthv_scraper         avail      auto      999:999

```

N> add adtemp\_faker

```
adtemp_faker: added process group
```

N> status

```

adtemp_faker          STARTING
dybslv                RUNNING   pid 2990, uptime 5:41:46
hgweb                 RUNNING   pid 2992, uptime 5:41:46

```

```
mysql          RUNNING    pid 2993, uptime 5:41:46
nginx          RUNNING    pid 2991, uptime 5:41:46
```

```
N> status
adtemp_faker   RUNNING    pid 22822, uptime 0:00:01
dybslv        RUNNING    pid 2990, uptime 5:41:50
hgweb         RUNNING    pid 2992, uptime 5:41:50
mysql         RUNNING    pid 2993, uptime 5:41:50
nginx         RUNNING    pid 2991, uptime 5:41:50
```

Subsequently can start/stop/restart/tail in normal manner. Following changes to supervisord configuration, such as environment changes, using just **start** for stopped process does not pick up the changed config. Ensure changes are picked up by using **remove**, **reread** and **add** which typically also starts the child process.

### 21.15.10 Steps to Deployment

#### Separate Testing and Production Config

Convenient testing requires far more rapid scraping that is needed in production, thus avoid having to change config by separating config for testing and production. The scraper can be instructed to read a different config file via `SCRAPER_CFG`, as described above *Configuration Mechanics*. This envvar can be set within the **supervisord** control file as described above *Continuous running under supervisord*.

Recommended steps towards scraper deployment:

1. setup a **faker** to write into `fake_dcs` with one process while the corresponding scraper is run in another process `fake_dcs -> tmp_offline_db`, as described above *Testing Scraper Operation*, this allows testing:
  - (a) live running
  - (b) catchup : by stop/start of the scraper
  - (c) scraper parameter tuning
2. test from real `dcs -> tmp_offline_db`
  - (a) make sure you have readonly permissions in the DBCONF “dcs” source section first!
  - (b) get supervisord setup *Continuous running under supervisord* to allow long term running over multiple days
  - (c) check the scraper can run continuously,
    - i. look for sustainability (eg avoid dumping huge logs)
    - ii. check responses to expected problems (eg network outtages), possibly **supervisord** config can be adjusted to auto-restart scrapers

### 21.15.11 Development Tips

#### Obtain mysqldump of DCS DB to populate fake source DB

Dumping just the table creation commands from the replicated DCS DB into file `~/fake_dcs.sql` (password read from a file):

```
mysqldump --no-defaults --no-data --lock-tables=false --host=202.122.37.89 --user=dayabay --password=
```

**Note:**

1. `--no-data` option must be used, to avoid creation of unusably large dump files
2. `--lock-tables=false` is typically needed to avoid permission failures

**Single table mysqldump for averager testing**

Averager testing requires a large dataset, so rather than add batch capability to the faker to generate this it is simpler and more realistic to just dump real tables from the replicated DCS DB. For example:

```
time mysqldump --no-defaults --lock-tables=false --host=202.122.37.89 --user=dayabay --password=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 32 | head -n 1 | tr -d '\n')
## 27 min yielded 207MB of truncated dump up to 1420760,'2011-02-19 10:19:10'

time mysqldump --no-defaults --lock-tables=false --host=202.122.37.89 --user=dayabay --password=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 32 | head -n 1 | tr -d '\n')
## cut the dump down to size with where clause : 10 seconds, 2.1M, full range

time mysqldump --no-defaults --lock-tables=false --host=202.122.37.89 --user=dayabay --password=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 32 | head -n 1 | tr -d '\n')
## 84 seconds, 21M, full range

time mysqldump --no-defaults --lock-tables=false --host=202.122.37.89 --user=dayabay --password=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 32 | head -n 1 | tr -d '\n')
## 462 seconds, 203M, full range
```

Check progress of the dump with:

```
tail --bytes=200 ~/AD1_LidSensor_10.sql ## use bytes option as very few newlines in mysqldumps
```

Replace any pre-existing `fake_dcs.AD1_LidSensor` table with:

```
cat ~/AD1_LidSensor_10.sql | mysql fake_dcs
cat ~/AD1_LidSensor_100.sql | mysql fake_dcs
cat ~/AD1_LidSensor_1000.sql | mysql fake_dcs
```

Check ranges in the table with group by year query:

```
echo "select count(*),min(id),max(id),min(date_time),max(date_time) from AD1_LidSensor group by year"

count(*)      min(id) max(id)      min(date_time)      max(date_time)
13697         1      3685338 0000-00-00 00:00:00      0000-00-00 00:00:00
151           9941588 13544749 1970-01-01 08:00:00      1970-01-01 08:00:00
11032689     43      11046429 2011-01-10 10:34:28      2011-12-31 23:59:58
2508947     11046430 13555485 2012-01-01 00:00:00      2012-02-29 15:19:43
```

If seeding is used, the range of averaging will be artificially truncated. For rerunnable test averages over full range:

```
time ./scr.py -s adlid_averager --ALLOW_DROP_CREATE_TABLE --DROP_TARGET_TABLES
## full average of modulo 10 single AD1_LidSensor table : ~6m
## full average of modulo 100 single AD1_LidSensor table : ~4m35s
```

**Append capable mysqldumps**

The dumps created as described above have structure:

```
DROP TABLE IF EXISTS `AD1_LidSensor`;
..
CREATE TABLE `AD1_LidSensor` (
```

```

'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
'date_time' datetime NOT NULL,
...
);
LOCK TABLES `AD1_LidSensor` WRITE;
INSERT INTO `AD1_LidSensor` VALUES
(10,'0000-00-00 00:00:00',237,301,'18.77','18.95','-0.77','0.24','0.01','-0.44','-0.57','1.12',5,'19
(20,'0000-00-00 00:00:00',237,302,'18.77','18.90','-0.77','0.24','0.02','-0.44','-0.57','1.12',5,'19
...
(13558330,'2012-02-29 16:54:33',2277,2103,'22.30','22.42','-1.01','0.27','-0.28','-0.42','-0.75','1.1
UNLOCK TABLES;

```

Skip the DROP+CREATE with `--no-create-info`, restrict to new id and pipe the dump directly into dev DB to bring uptodate (modulo 100):

```

maxid=$(echo "select max(id) from AD1_LidSensor" | mysql --skip-column-names fake_dcs ) ; echo $maxid
time mysqldump --no-defaults --no-create-info --lock-tables=false --host=202.122.37.89 --user=dayaba

```

Test append running of averager:

```

time ./scr.py -s adlid_averager

```

Catches up with 2 bins:

```

INFO:Scraper.base.datetimebin: [0 ] ['Wed Feb 29 15:00:00 2012', 'Thu Mar  1 00:00:00 2012'] 9:00:00
INFO:Scraper.base.datetimebin: [1 ] ['Thu Mar  1 00:00:00 2012', 'Thu Mar  1 11:00:00 2012'] 11:00:00
INFO:Scraper.base.averager:looping over 2 territory bins performing grouped aggregate queries in each
INFO:Scraper.base.sourcevector:SV 1 (0, 1) 2012-02-29 15:00:00=>00:00:00 full
INFO:Scraper.base.sourcevector:SV 2 (0, 1) 2012-03-01 00:00:00=>11:00:00 full

```

Checking target, shows no seams:

```

echo "select * from DcsAdLidSensorVld where Timestart > DATE_SUB(UTC_TIMESTAMP(),INTERVAL 36 HOUR)"

6515 2012-02-29 02:00:13 2012-02-29 02:56:53 1 1 1 0 -1 2012-
6516 2012-02-29 03:00:13 2012-02-29 03:56:53 1 1 1 0 -1 2012-
6517 2012-02-29 04:00:13 2012-02-29 04:56:53 1 1 1 0 -1 2012-
6518 2012-02-29 05:00:13 2012-02-29 05:56:53 1 1 1 0 -1 2012-
6519 2012-02-29 06:00:13 2012-02-29 06:56:53 1 1 1 0 -1 2012-
6520 2012-02-29 07:00:13 2012-02-29 07:56:53 1 1 1 0 -1 2012-
6521 2012-02-29 08:00:13 2012-02-29 08:56:53 1 1 1 0 -1 2012-
6522 2012-02-29 09:00:13 2012-02-29 09:56:57 1 1 1 0 -1 2012-
6523 2012-02-29 10:00:17 2012-02-29 10:56:57 1 1 1 0 -1 2012-
6524 2012-02-29 11:00:17 2012-02-29 11:56:57 1 1 1 0 -1 2012-
6525 2012-02-29 12:00:17 2012-02-29 12:56:57 1 1 1 0 -1 2012-

```

## Multi-source table test

### Start from scratch following schema changes to DCS

Drop pre-existing `fake_dcs` DB and recreate from the nodata mysqldump:

```

mysql> status ## verify connected to local development server
mysql> drop database if exists fake_dcs ;
mysql> create database fake_dcs ;
mysql> use fake_dcs
mysql> source ~/fake_dcs.sql ## use nodata dump to duplicate table definitions
mysql> show tables

```

**Warning:** only use the below approach on local development server when confident of mysql config

Quick (and **DANGEROUS**) way of doing the above which works as mysqldump defaults to including `DROP TABLE IF EXISTS` prior to `CREATE TABLE` allowing emptying data from all tables without having to drop/recreate the DB. **CAUTION:** this assumes that the **client** section of `~/ .my.cnf` is on the same server as the DB called **fake\_dcs**

```
cat ~/fake_dcs.sql | mysql fake_dcs
```

## Interactive SQLAlchemy Querying

Use `NonDbi` to pull up a session and dynamic SQLAlchemy class to query with ipython:

```
[blyth@belle7 Scraper]$ ipython
Python 2.7 (r27:82500, Feb 16 2011, 11:40:18)
Type "copyright", "credits" or "license" for more information.

IPython 0.9.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

In [1]: from NonDbi import session_

In [2]: session = session_("fake_dcs")    ## dbconf

In [3]: kls = session.kls_("DBNS_AD1_HV")  ## table name

In [4]: q = session.query(kls).order_by(kls.date_time)  ## does not touch DB yet

In [5]: q.count()                          ## hits DB now
Out[5]: 74L

In [6]: q.first()                          ## LIMIT 0, 1    same as q[0:1][0]    (maybe different errors if empty though)
Out[6]: <NonDbi.YDBNS_AD1_HV object at 0xa7a404c>

In [7]: q[70:74]                            ## LIMIT 70,4
Out[7]:
[<NonDbi.YDBNS_AD1_HV object at 0xa7bea4c>,
 <NonDbi.YDBNS_AD1_HV object at 0xa7beaac>,
 <NonDbi.YDBNS_AD1_HV object at 0xa7bea8c>,
 <NonDbi.YDBNS_AD1_HV object at 0xa7beb2c>]

In [8]: q[70:75]                            ## LIMIT 70,5
Out[8]:
[<NonDbi.YDBNS_AD1_HV object at 0xa7bea4c>,
 <NonDbi.YDBNS_AD1_HV object at 0xa7beaac>,
 <NonDbi.YDBNS_AD1_HV object at 0xa7bea8c>,
 <NonDbi.YDBNS_AD1_HV object at 0xa7beb2c>]

In [9]: q[74:75]                            ## LIMIT 74,1
Out[9]: []

In [10]: q[73:74]                           ## LIMIT 73,1
Out[10]: [<NonDbi.YDBNS_AD1_HV object at 0xa7beb2c>]
```

## 21.16 DBI Internals

- Overlay versioning implementation
- Overlay overriding problem
- Fix attempt A
  - Possible Workaround
  - Possible Solution
  - Looking for preexisting manifestations
  - Problem with this fix A
- Write single entry into empty table
- Write two entries at same validity range ts:EOT into empty table
- Write 3 entries for different runs into empty table
- Delving into overlay detection and DbValidityRecBuilder
- fGap : special vrec holding trim results
- Trimming in Builder ctor
  - AndTimeWindow : overlap range
  - FindTimeBoundaries
  - Bracketed Trimming : effective range reduced to overlap other
  - Non bracketed trim : effective range reduced to exclude the other
  - Double Overlay Example
  - Trim In full

Bug hunting inside DBI, not for users.

### 21.16.1 Overlay versioning implementation

Driven by the writer Close dybgaudi:Database/DatabaseInterface/DatabaseInterface/DbiWriter.tpl

```
template<class T>
Bool_t DbiWriter<T>::Close(const char* fileSpec)

... snipped ...

// Use overlay version date if required.
    if ( fUseOverlayVersionDate && fValidRec )
        fPacket->SetVersionDate(fTableProxy->QueryOverlayVersionDate(fValidRec, fDbNo));

// Set SEQNO and perform I/O.
    fPacket->SetSeqNo(seqNo);
    ... snip ...
    ok = fPacket->Store(fDbNo);
```

From the various Open:

```
fUseOverlayVersionDate = vrec.GetVersionDate() == TimeStamp(0,0);
```

Quoting comments from QueryOverlayVersionDate of dybgaudi:Database/DatabaseInterface/src/DbiTableProxy.cxx:

```
TimeStamp DbiTableProxy::QueryOverlayVersionDate(const DbValidityRec& vrec,
                                                UInt_t dbNo)
{
//
// Purpose: Determine a suitable Version Date so that this validity
```

```

//          record, if written to the selected DB, will overlay
//          correctly.
//
// Specification:-
// =====
//
// o Determine optimal Version Date to overlay new data.  See Program Notes.

// Program Notes:-
// =====

// It is normal practice, particularly for calibration data, to have
// overlapping the validity records.  Each time a new set of runs are
// processed the start time of the validity is set to the start time of
// the first run and the end time is set beyond the start time by an
// interval that characterises the stability of the constants.  So long
// as a new set of constants is created before the end time is reached
// there will be no gap.  Where there is an overlap the Version Date is
// used to select the later constants on the basis that later is better.
// However, if reprocessing old data it is also normal practice to
// process recent data first and in this case the constants for earlier
// data get later version dates and overlay works the wrong way.  To
// solve this, the version date is faked as follows:-
//
//
// 1.  For new data i.e. data that does not overlay any existing data,
//     the version date is set to the validity start time.
//
// 2.  For replacement data i.e. data that does overlay existing data,
//     the version date is set to be one minute greater than the Version
//     Date on the current best data.
//
// This scheme ensures that new data will overlay existing data at the
// start of its validity but will be itself overlaid by data that has
// a later start time (assuming validity record start times are more
// than a few minutes apart)

// Create a context that corresponds to the start time of the validity
// range.  Note that it is O.K. to use SimFlag and Site masks
// even though this could make the context ambiguous because the
// context is only to be used to query the database and the SimFlag and
// Site values will be ORed against existing data so will match
// all possible data that this validity range could overlay which is
// just what we want.

const ContextRange& vr(vrec.GetContextRange());
Context vc((Site::Site_t) vr.GetSiteMask(),
           (SimFlag::SimFlag_t) vr.GetSimMask(),
           vr.GetTimeStart());

DbiConnectionMaintainer cm(fCascader); //Stack object to hold connections

// Build a complete set of effective validity records from the
// selected database.
DbiValidityRecBuilder builder(fDBProxy,vc,vrec.GetSubSite(),vrec.GetTask(),dbNo);

```

```
// Pick up the validity record for the current aggregate.
const DbValidityRec& vrecOvlay(builder.GetValidityRecFromAggNo(vrec.GetAggregateNo()));

// If its a gap i.e. nothing is overlaid, return the start time, otherwise
// return its Version Date plus one minute.
TimeStamp ovlayTS(vr.GetTimeStart());
if ( ! vrecOvlay.IsGap() ) {
    time_t overlaySecs = vrecOvlay.GetVersionDate().GetSec();
    ovlayTS = TimeStamp(overlaySecs + 60,0);
}

LOG(dbi,Logging::kDebug1) << "Looking for overlay version date for: "
    << vrec << "found it would overlap: "
    << vrecOvlay << " so overlay version date set to "
    << ovlayTS.AsString("s") << std::endl;

return ovlayTS;
```

### 21.16.2 Overlay overriding problem

Consider overlay usage in a run-by-run to EOT regime:

```

                                                    EOT
100 -----
101      -----
102          -----
103              -----
```

Other than for the first entry (run 100) in the table there will always be pre-existing data as each subsequent run record gets written. Thus the VERSIONDATE will always get incremented off the TIMESTART of the last entry. This will cause problems as in the case of overriding overrides there will be VERSIONDATE clashes.

Clearly the solution is to somehow distinguish between an *intended* overlay:

```

                                                    EOT
100 -----
101      -----
102          -----
102              -----    <<< real overlay in need of VERSIONDATE = ts102+1min
103                  -----
```

As opposed to a *technical* overlay:

```

                                                    EOT
100 -----
101      -----
102          -----
102              -----
103                  -----
104                      -----    <<<< new entry that needs VERSIONDATE = ts104
                                   rather than ts103 + 1min
                                   aka ts102 + 1min + 1min
                                   aka ts101 + 1min + 1min + 1min
                                   aka ts100 + 1min + 1min + 1min + 1min
```

### 21.16.3 Fix attempt A

#### Possible Workaround

Do not use overlay versioning on the first pass... instead force the versiondate to be the timestart versiondate = cr.timestart

#### Possible Solution

Modify the feeler query to make the distinction, maybe as simple as adding clause and VERSIONDATE >= ts  
Simulate this solution by applying an SqlCondition during the writer close.

```

if fixcondition:
    condition = "VERSIONDATE >= '%s'" % cr.timestart.AsString("s")
    log.debug( "write_ fixcondition %s during writer close " % condition )
    gDbi.registry.SetSqlCondition(condition)    ## CAUTION THIS IS A GLOBAL CONDITION

assert wrt.Close()

if fixcondition:
    log.debug( "write_ fixcondition clear after writer close " )
    gDbi.registry.SetSqlCondition("")

```

This succeeds without requiring special treatment on the first pass.

#### Looking for preexisting manifestations

Find duplicate versiondates:

```
SELECT SEQNO,VERSIONDATE,COUNT(VERSIONDATE) AS dupe FROM CalibFeeSpecVld GROUP BY VERSIONDATE HAVING
```

Added a db.py command to do this over all validity tables, usage:

```

db.py tmp_offline_db vdupe
db.py offline_db      vdupe      ## there are many

```

#### Problem with this fix A

Approach A will usually delay manifestation of the problem, but it does not fix it... as apparently the logic that is finding overlays is throwing up VERSIONDATES that are duplicated within the same context.

### 21.16.4 Write single entry into empty table

Using starttime of run 11717 and EOT and forced timegate of 60s (tg):

```

mysql> select p.runNo, v.TIMESTART, v.TIMEEND, v.VERSIONDATE, v.INSERTDATE from DaqRunInfo as p, Daq
+-----+-----+-----+-----+-----+
| runNo | TIMESTART          | TIMEEND          | VERSIONDATE      | INSERTDATE      |
+-----+-----+-----+-----+-----+
...
| 11717 | 2011-08-04 05:54:47 | 2011-08-04 05:59:51 | 2011-08-04 05:54:47 | 2011-08-04 06:09:15 |
...

```

The pre-write query on empty table is:



```
select max(TIMEEND) from DemoVld where
    TIMEEND < '2011-08-04 05:53:47'          ##      timeend < ts - t
    and VERSIONDATE >= '1970-01-01 00:00:00' ## and versiondate >= (
    and SiteMask & 127 and SimMask & 1 and SubSite = 0 and Task = 0
```

Source is FindTimeBoundaries from dybgaudi:Database/DatabaseInterface/src/DbiDBProxy.cxx which is driven from DbiValidityRecBuilder ctor dybgaudi:Database/DatabaseInterface/src/DbiValidityRecBuilder.cxx and is controllable by argument **findFullTimeWindow**

Resulting insert goes in with VERSIONDATE == TIMESTART:

```
INSERT INTO DemoVld VALUES      ## TIMESTART          TIMEEND          VERSIONDATE
(31,'2011-08-04 05:54:47','2038-01-19 03:14:07',127,1,0,0,-1,'2011-08-04 05:54:47')
INSERT INTO Demo VALUES
(31,1,10,11717)
```

### 21.16.5 Write two entries at same validity range ts:EOT into empty table

1st entry proceeds precisely as above. The feeler query of 2nd entry is the same but this time it yields the 1st entry:

SEQNO	TIMESTART	TIMEEND	SITEMASK	SIMMASK	SUBSITE	TASK	AGGREGATE
31	2011-08-04 05:54:47	2038-01-19 03:14:07	127	1	0	0	

The min-maxing proceeds similarly but this time with VERSIONDATE >= '2011-08-04 05:54:47'

Resulting insert goes in with VERSIONDATE == TIMESTART + 1min:

```
INSERT INTO DemoVld VALUES      ##
(32,'2011-08-04 05:54:47','2038-01-19 03:14:07',127,1,0,0,-1,'2011-08-04 06:05:47')
INSERT INTO Demo VALUES
(32,1,11,11717)
```

### 21.16.6 Write 3 entries for different runs into empty table

Result is coupled VERSIONDATE:

```
mysql> select * from DemoVld ;
+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART          | TIMEEND          | SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATE |
+-----+-----+-----+-----+-----+-----+-----+
| 33    | 2011-08-04 05:54:47 | 2038-01-19 03:14:07 | 127      | 1       | 0       | 0    |           |
| 34    | 2011-08-04 06:15:46 | 2038-01-19 03:14:07 | 127      | 1       | 0       | 0    |           |
| 35    | 2011-08-04 07:02:51 | 2038-01-19 03:14:07 | 127      | 1       | 0       | 0    |           |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select * from Demo ;
+-----+-----+-----+-----+
| SEQNO | ROW_COUNTER | Gain | Id   |
+-----+-----+-----+-----+
| 33    | 1           | 10  | 11717 |
| 34    | 1           | 10  | 11718 |
| 35    | 1           | 10  | 11719 |
+-----+-----+-----+-----+
```



quite lightweight with just entries that start off as Gaps for each aggregate in the vector (contrary for first impressions and very different extended context behaviour) and are trimmed by the Vld query entries (which are not stored).

### 21.16.8 fGap : special vrec holding trim results

Created by `DbiValidityRecBuilder::MakeGapRec(const Context& vc, const string& tableName, Bool_t findFullTimeWindow)` essentially:

```
ContextRange gapVR(vc.GetSite(), vc.GetSimFlag(), startGate, endGate);    ##
fGap = DbiValidityRec(gapVR, fSubSite, fTask, -2, 0, 0, kTRUE);
        ## range subsite task aggNo seqNo dbNo isGap
```

Gate is BOT:EOT when findFullTimeWindow=True otherwise tis ts-tg:ts+tg, `Dbi::GetTimeGate(tableName)` defaults are big ~10days

### 21.16.9 Trimming in Builder ctor

Prior to vld row loop

```
const Timestamp curVTS = vc.GetTimeStamp();
Timestamp earliestCreate(0);    // Set earliest version date to infinite past - the right value
```

Within the vld row loop

```
const DbiValidityRec* vr = dynamic_cast<const DbiValidityRec*>(result.GetTableRow(row));

// Trim the validity record for the current aggregate number by this record and see if we have
DbiValidityRec& curRec = fVRecs[index];    // curRec summarizes all the validities within an aggregate

curRec.Trim( curVTS, \*vr );

#####
##### only while curRec is still a gap does Trim do anything
#####     ... it becomes non gap when bracketing validity is hit
#####     ... the ordering is VERSIONDATE desc, so that means the highest VERSIONDATE with val
#####     ... becomes non-gap first, there-after no more trimming is done
#####
#####
##### if curVTS is within \*vr range (ie \*vr brackets curVTS)
#####     curRec becomes \*vr                                     (that includes the VERSIONDATE)
#####     range is trimmed to the overlap with the other
##### otherwise
#####     range is trimmed to exclude the other
#####
#####

if ( ! curRec.IsGap() ) { foundData = kTRUE; curRec.SetDbNo(dbNo); }

// Find the earliest non-gap version date that is used
if ( curRec.GetSeqNo() == vr->GetSeqNo() && ( earliestCreate > vr->GetVersionDate() || earliestCreat

##### no non-gap restriction ?
#####     ... implicitly done as while curRec is a gap it has SEQNO 0
##### WHY SEQNO EQUALITY ?
#####     WILL ONLY FIND ONE ENTRY IN ENTIRE VECTOR
```



```

      .      |      .
      start  |      end

1   min(ts) where ts > eg
2   min(te) where te > eg
3   max(ts) where ts < sg
4   max(te) where te < sg

```

But with restriction: VERSIONDATE >= earliestCreate

```

void DbiDBProxy::FindTimeBoundaries(const Context& vc,
                                     const Dbi::SubSite& subsite,
                                     const Dbi::Task& task,
                                     UInt_t dbNo,
                                     TimeStamp earliestCreate,
                                     TimeStamp& start,
                                     TimeStamp& end) const {
//
//
// Purpose: Find next time boundaries beyond standard time gate.
//
// Arguments:
//   vc           in   The Validity Context for the query.
//   subsite      in   The subsite of the query.
//   task         in   The task of the query.
//   dbNo        in   Database number in cascade (starting at 0).
//   earliestCreate in Earliest version date of data in the time gate
//   start        out  Lower time boundary or TimeStamp(0,0) if none
//   end          out  Upper time boundary or TimeStamp(0x7FFFFFFF,0) if none
//
// Specification:-
// =====
//
// o Find the next time boundary (either TIMESTART or TIMEEND)
//   outside the current time gate with a version date >= earliestCreate.

LOG(dbi,Logging::kMonitor) << "FindTimeBoundaries for table " << fTableName
                             << " context " << vc
                             << " subsite " << subsite
                             << " task " << task
                             << " Earliest version date " << earliestCreate
                             << " database " << dbNo << std::endl;

// Set the limits wide open
start = TimeStamp(0,0);
end   = TimeStamp(0x7FFFFFFF,0);

// Construct a Time Gate on the current date.

const TimeStamp curVTS = vc.GetTimeStamp();
Int_t timeGate = Dbi::GetTimeGate(this->GetTableName());
time_t vcSec = curVTS.GetSec() - timeGate;

TimeStamp startGate(vcSec,0);
vcSec += 2*timeGate;
TimeStamp endGate(vcSec,0);

```

```

string earliestCreateString(Dbi::MakeDateTimeString(earliestCreate));
string startGateString(Dbi::MakeDateTimeString(startGate));
string endGateString(Dbi::MakeDateTimeString(endGate));

// Extract information for Context.

Site::Site_t      detType(vc.GetSite());
SimFlag::SimFlag_t      simFlg(vc.GetSimFlag());

// Use an auto_ptr to manage ownership of DbStatement and TSQLStatement
std::auto_ptr<DbStatement> stmtDb(fCascader.CreateStatement(dbNo));

for (int i_limit =1; i_limit <= 4; ++i_limit ) {

    DbString sql("select ");
    if ( i_limit == 1 ) sql << "min(TIMESTART) from " << fTableName << "Vld where TIMESTART > ' " <<
    if ( i_limit == 2 ) sql << "min(TIMEEND)   from " << fTableName << "Vld where TIMEEND > ' " <<
    if ( i_limit == 3 ) sql << "max(TIMESTART) from " << fTableName << "Vld where TIMESTART < ' " <<
    if ( i_limit == 4 ) sql << "max(TIMEEND)   from " << fTableName << "Vld where TIMEEND < ' " << st

    sql << " and SiteMask & " << static_cast<unsigned int>(detType) << " and SimMask & " << static_c
    << " and VERSIONDATE >= ' " << earliestCreateString << "' "
    << " and SubSite = " << subsite
    << " and Task = " << task;

    LOG(dbi,Logging::kMonitor) << " FindTimeBoundaries query no. " << i_limit << " SQL:" <<sql.c_str()

    std::auto_ptr<TSQLStatement> stmt(stmtDb->ExecuteQuery(sql.c_str()));
    stmtDb->PrintExceptions(Logging::kDebug1);

// If the query returns data, convert to a time stamp and trim the limits
TString date;
if ( ! stmt.get() || ! stmt->NextResultRow() || stmt->IsNull(0) ) continue;
date = stmt->GetString(0);

if ( date.IsNull() ) continue;
TimeStamp ts(Dbi::MakeTimeStamp(date.Data()));

LOG(dbi,Logging::kMonitor) << " FindTimeBoundaries query result: " << ts << std::endl;
if ( i_limit <= 2 && ts < end   ) end   = ts;
if ( i_limit >= 3 && ts > start ) start = ts;

}

LOG(dbi,Logging::kMonitor) << "FindTimeBoundaries for table " << fTableName
    << " found " << start << " .. " << end << std::endl;

}

```

## Bracketed Trimming : effective range reduced to overlap other

Both ranges have validity, but **Caution**

1. **other becomes this** with range chopped by the initial **this**
  - (a) “‘VERSIONDATE gets transferred from other to this!!’”

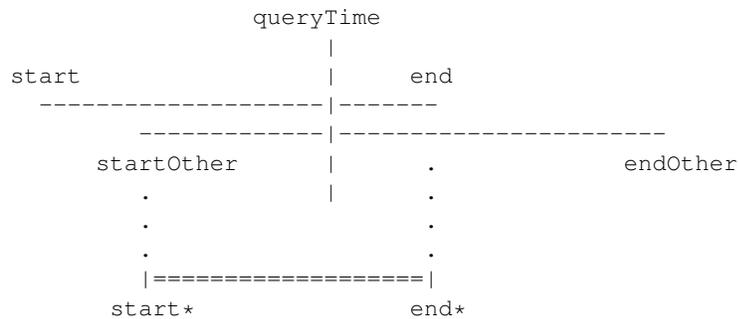
```
// If this record is not a gap then the other record can be ignore
// as it is of lower priority.

if ( ! IsGap() ) return;

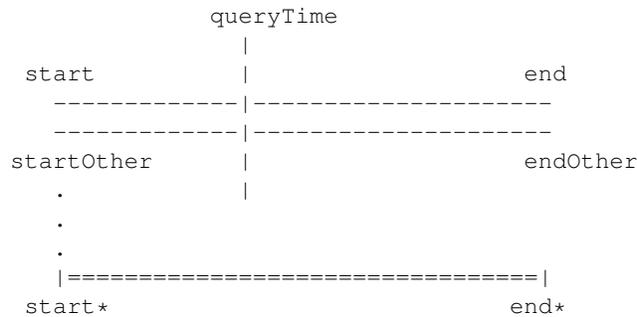
// If entry brackets query date, then use it but with a validity that
// is trimmed by the current record.

if ( startOther <= queryTime && endOther > queryTime ) {
  if ( start < startOther ) start = startOther;
  if ( end > endOther ) end = endOther;
  \*this = other;
  SetTimeWindow(start,end);
}
```

Pictorially:



Consider the equal range trim, tis bracketing so VERSIONDATE will adopt the others:



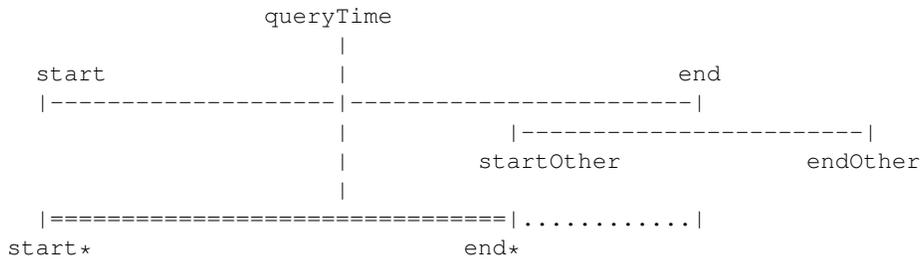
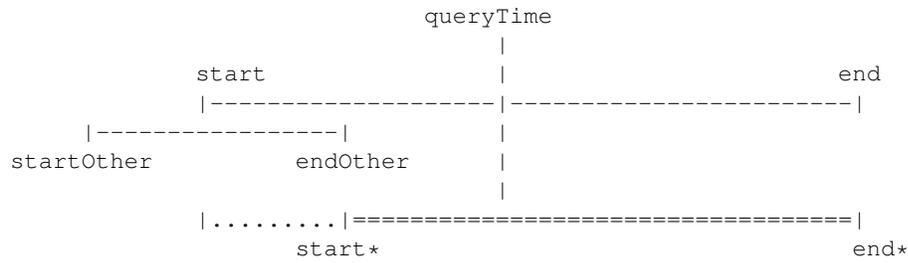
**Non bracketed trim : effective range reduced to exclude the other**

Other range is not valid for the queryTime but the current validity range is impinged by the other. Before and after overlap trims with no identity/VERSIONDATE change.

```
// It doesn't bracket, so use it to trim the window

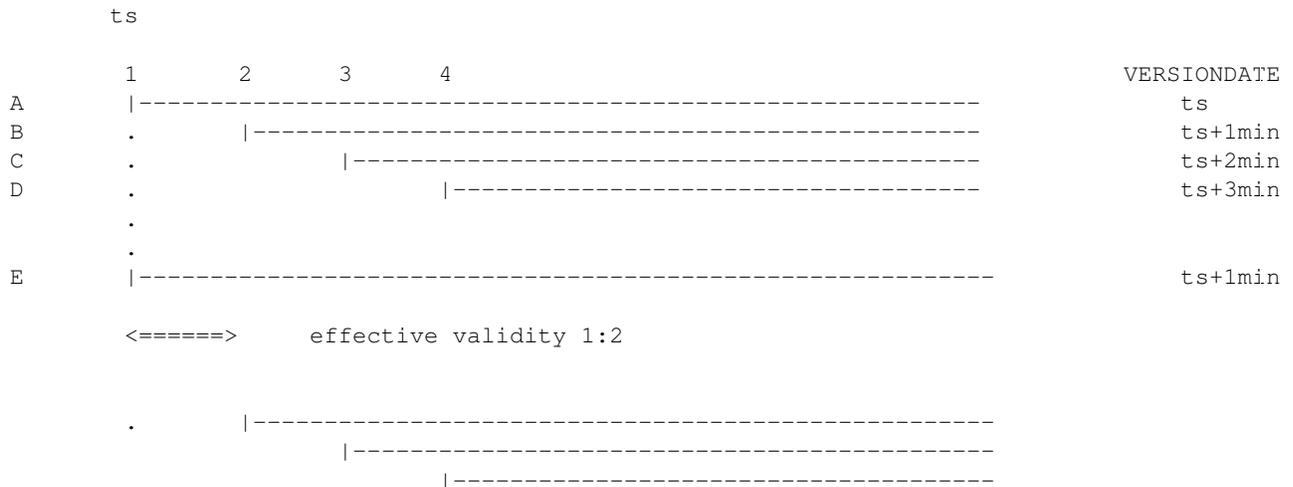
if ( endOther <= queryTime ) {
  if ( start < endOther ) SetTimeWindow(endOther,end);
}
else if ( startOther > queryTime ) {
  if ( end > startOther ) SetTimeWindow(start, startOther);
}
```

Other before (will never occur with endOther at EOT) and other after:



### Double Overlay Example

Consider writing 4 runs with timeend to EOT and then going back an overlaying on top:



When checking for overlay prior to 5th write a VERSIONDATE desc loop causes gap trimming until strike validity at A.

D	1970-01-01 00:00:00 .. 2010-01-01 04:00:00
C	1970-01-01 00:00:00 .. 2010-01-01 03:00:00
B	1970-01-01 00:00:00 .. 2010-01-01 02:00:00
A	2010-01-01 01:00:00 .. 2010-01-01 02:00:00

Debugging:

```
[12 /79 ] check_write      ('b', (11717,)), 'a')      ==>
DVRB rowvr  row:0 seqNo:4 ts:2010-01-01 04:00:00.000000000Z vd:2010-01-01 01:03:00.000000000Z
DVRB rowvr  row:1 seqNo:3 ts:2010-01-01 03:00:00.000000000Z vd:2010-01-01 01:02:00.000000000Z
```

```

DVRB rowvr row:2 seqNo:2 ts:2010-01-01 02:00:00.000000000Z vd:2010-01-01 01:01:00.000000000Z
DVRB rowvr row:3 seqNo:1 ts:2010-01-01 01:00:00.000000000Z vd:2010-01-01 01:00:00.000000000Z
DVRB curRec SeqNo: 0 AggNo: -1 DbNo: 0 (gap) ContextRange: |0x07f|0x 1| 1970-01-01 00:00:00 .. 20
DVRB curRec SeqNo: 0 AggNo: -1 DbNo: 0 (gap) ContextRange: |0x07f|0x 1| 1970-01-01 00:00:00 .. 20
DVRB curRec SeqNo: 0 AggNo: -1 DbNo: 0 (gap) ContextRange: |0x07f|0x 1| 1970-01-01 00:00:00 .. 20
DVRB curRec SeqNo: 1 AggNo: -1 DbNo: 0 ContextRange: |0x07f|0x 1| 2010-01-01 01:00:00 .. 20
Traceback (most recent call last):
  File "test_overlay_versioning.py", line 382, in <module>
    ret = fn(*args)
  File "test_overlay_versioning.py", line 293, in check_write
    dwrite = write_( cr_(g['Id']) , *g )
  File "test_overlay_versioning.py", line 195, in write_
    assert lvd not in lvds.values(), "lvd %s is already present %r " %(lvd,lvds)
AssertionError: lvd 2010-01-01 01:01:00 is already present {1L: '2010-01-01 01:00:00', 2L: '2010-01-01

```

## Trim In full

```

void DbValidityRec::Trim(const TimeStamp& queryTime, const DbValidityRec& other) {
//
//
// Purpose: Trim this validity record so that represents
//          best validity record for query.
//
// Arguments:
//   queryTime    in    Time of query
//   other         in    DbValidity record satisfying query
//
// Return:      None.
//
// Contact:     N. Tagg          Original Author: N. West, Oxford
//
// Specification:-
// =====
//
// o Update this validity record so that it remains the best
//   validity record taking into account the supplied record.
//
// Program Notes:-
// =====
//
// This is the function that deal with validity management.
// It takes into account that several validity records may
// overlap and that the best one is the one with the latest
// version date that brackets the query date. Other entries
// with later version dates may trim start or end times.
//
// Assumptions:-
// =====
//
// That entries are submitted in strict descending priority i.e.:-
//
// 1) Entries for a higher priority database precede those from a
//    lower priority one.
//
// 2) Within a database entries are in descending version date order.

```

```
// Ignore other records that are either gaps or have wrong
// aggregate number.

    if ( fAggregateNo != other.fAggregateNo || other.IsGap() ) return;

// If this record is not a gap then the other record can be ignore
// as it is of lower priority.

    if ( ! IsGap() ) return;

    TimeStamp start      = fContextRange.GetTimeStart();
    TimeStamp end        = fContextRange.GetTimeEnd();
    TimeStamp startOther = other.GetContextRange().GetTimeStart();
    TimeStamp endOther   = other.GetContextRange().GetTimeEnd();

// If entry brackets query date, then use it but with a validity that
// is trimmed by the current record.

    if ( startOther <= queryTime && endOther > queryTime ) {
        if ( start < startOther ) start = startOther;
        if ( end > endOther ) end = endOther;
        *this = other;
        SetTimeWindow(start,end);
    }

// It doesn't bracket, so use it to trim the window

    else {

        if ( endOther <= queryTime ) {
            if ( start < endOther ) SetTimeWindow(endOther,end);
        }
        else if ( startOther > queryTime ) {
            if ( end > startOther ) SetTimeWindow(start, startOther);
        }
    }
}
```

## 21.17 DBI Overlay Versioning Bug

- Background
  - Root Cause of Issue
  - Ordering of validity queries
  - Validity Look Up Tables
  - DBI Scanning
- Current Intended code/SOP changes
  - Planned rollout of DBI modifications
    - \* Extra Ordering Fix
    - \* Timestart Flooring and Collision Avoidance
- Intended Migration of Existing DB entries
  - Rebuild Approach
    - \* Table Rebuilding and Insertdates
    - \* Table Summary
      - CableMap/HardwareID
      - CalibPmtSpec
      - CalibPmtHighGain
    - \* Test Rebuilding
  - Fixup Validity Approach
    - \* Current Transfixion Approach
- Vlut Comparisons
  - Summary Examination
    - \* tmp\_offline\_db (copy of offline\_db)
    - \* fix\_offline\_db (with VERSIONDATE TIMESTART flooring)
    - \* tmp\_offline\_db\_cf\_fix\_offline\_db
  - Observations
  - Sampling Vlut extracts
    - \* tmp\_offline\_db\_cf\_fix\_offline\_db    vlut    orderingSEQNOdesc    insertdates:19  
timestarts:18 ndif:19
- Alternative to Recreat Rather Than Fix Approach
  - CableMap/HardwareID
    - \* Pumping dybaux history with auxlog.py
    - \* Re-creation Discrepancy in FEC loading : resolved with takebogus option
    - \* Relevant Tickets
    - \* Bogus Logic
    - \* Whole table groupby INSERTDATE for overview
- DBI Validity Ordering Change
- Payload Digest Rather than SEQNO comparison
  - CableMap
  - HardwareID

See also dybsvn:ticket:948

## 21.17.1 Background

### Root Cause of Issue

When assigning versiondates DBI considers prior validity corresponding only to the TIMESTART of the new entry. In anything but the simplest of overlay histories this leads to a VERSIONDATE that collide with those from subsequent TIMESTART, this does **not** cause an issue at the initial TIMESTART but it does at later ones.

Validity queries at latter times see multiple validities tied in VERSIONDATE. Which wins is kinda undefined.

Earlier validity can leak forwards in time.

## Ordering of validity queries

The ordering of DBI validity queries crucially determines which of overlaid validities wins. Problems with the ordering such as caused by duplicated VERSIONDATE lead to **insidious** DBI behavior of silently returning wrong values in some regions of (INSERTDATE,TIME).

Validity Query Ordering	
a VERSIONDATE desc	historical DBI default : OK without VERSIONDATE duplications
b VERSIONDATE desc, SEQNO desc	makes the higher SEQNO of degenerate sets win ( <b>intended fix</b> )
c VERSIONDATE desc, SEQNO asc	makes the lower SEQNO of degenerate sets win (canary to see problems)

MySQL has implicit *SEQNO asc* as *SEQNO* is the PK. But observed differences between *VERSIONDATE desc* and *VERSIONDATE desc, SEQNO asc* indicates this does not follow thru to multi-column orderings.

Comparing VLUT created with different validity query ordering allows ambiguities to be located by varying the way that VERSIONDATE degeneracy is broken. Interpreting differences:

Difference	Interpretation
a-b	smoking-gun for affliction
b-c / a-c	skating on thin-ice

## Validity Look Up Tables

DBI Validity Look Up Tables (VLUTs) express all possible DBI validity results(SEQNO) determined by performing DBI queries at all TIMESTARTs with rollbacks to all INSERTDATES. They are presented as SEQNO values within tables with INSERTDATE vertically and TIMESTART horizontally.

Comparisons between such VLUTs enable problem periods to be identified, such comparisons lists the SEQNO values in the cell when differences are found.

## DBI Scanning

Scanning scripts to create VLUTs for all contexts in all tables `DybDbi.vld.vlut` (`dybgaudi:Database/DybDbi/python/DybDbi/vld/vlut.py`) The results are accessible beneath <http://belle7.nuu.edu.tw/dbiscan/> DBI scanning is an expensive operation that should not be done to production DB, instead copy tables from *offline\_db* into local *tmp\_offline\_db*.

Within each context variations to default DBI validity ordering are made, and the resulting VLUTs compared for 2 DBCONF:

### 21.17.2 Current Intended code/SOP changes

1. extra ordering to break validity degeneracy from VERSIONDATE collisions is mandatory, how to do that fairly clear:
  - (a) SEQNO desc (**best** approach as follows in spirit of VERSIONDATE, and makes future degeneracy impossible)
  - (b) SEQNO asc (lower SEQNO wins in VERSIONDATE collisions, good canary )
  - (c) TIMESTART asc (future possibilities of degeneracy are not eliminated when using TIMESTART, must use SEQNO for that)

2. adopt timestart floored VERSIONDATE
  - (a) reduces occurrence of degeneracy, and makes VERSIONDATEs more understandable
3. enforce no VERSIONDATE collisions
  - (a) DBI will refuse to write entries with collisions in the written context
  - (b) table experts will have to manually set VERSIONDATEs to achieve the desired overlaying, this cannot be automated as DBI cannot read the mind of the expert as to the intended overlaying
  - (c) there is still possibility of collisions when reading from wider contexts than written, thus must still pin down the extra ordering

### Planned rollout of DBI modifications

As multiple people need to check tables as migration is done it is not practical to change all tables at once.

### Extra Ordering Fix

The extra ordering fix is a fundamental change to DBI:

1. it touches almost all DBI operations, including reading and writing
2. it changes the results of validity queries and thus DBI results in many rollback/time regions
3. almost by definition changes are restricted to afflicted tables `CableMap`, `HardwareID`, `CalibPmtSpec`
4. implemented within DBI, in the `DbiDBProxy.cpp` ctor
5. `DybDbi` spec key `CanFixOrdering = [kTRUE|kFALSE]` allows per-table testing/rollout

---

**Note:** to minimise behaviour transitions the change in standard table `.spec` it is preferable to be done in concert with a DB rebuild

---

The ordering can be overridden (for test purposes only) with:

```
kls.GetTableProxy().GetDBProxy().SetExtraOrdering("SEQNO desc")
```

### Timestart Flooring and Collision Avoidance

These changes effect writing only and can be controlled table-by-table either:

1. via the spec writer default `wctx` strings:
  - (a) `RequireUniqueVersionDate.kTRUE`
  - (b) `TimeStartFlooredVersionDate.kTRUE`
2. dynamically on configuring the writer, `wrt.ctx( requireuniqueversiondate=True )`

`DybDbi` propagates these `ctx` settings in `void DbiWrt<T>::MakeWriter()`:

```
m_wrt->SetTimeStartFlooredVersionDate(m_ctx.GetTimeStartFlooredVersionDate());
m_wrt->SetRequireUniqueVersionDate(m_ctx.GetRequireUniqueVersionDate());
m_wrt->SetUniqueVersionDateSiteMask(m_ctx.GetUniqueVersionDateSiteMask());
m_wrt->SetUniqueVersionDateSimMask(m_ctx.GetUniqueVersionDateSimMask());
```

## 21.17.3 Intended Migration of Existing DB entries

### Rebuild Approach

Where possible it is much preferable to correct DBI and rerun over source loadfiles rather than attempting to **fix** things up and risk inconsistencies/confusion/doubts.

Tables that have been recreated from source reloading, listed with dybaux revision numbers:

CableMap	31	[4898, 4913, 4914, 4915, 4916, 4917, 4918, 4919, 4920, 4921, 4922, 4923, 4924, 4925, 4926, 4927, 4928, 4929, 4930, 4931, 4932, 4933, 4934, 4935, 4936, 4937, 4938, 4939, 4940, 4941, 4942, 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951, 4952, 4953, 4954, 4955, 4956, 4957, 4958, 4959, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969, 4970, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986, 4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998, 4999, 5000]
CalibPmtSpec	71	[4942, 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951, 4952, 4953, 4954, 4955, 4956, 4957, 4958, 4959, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969, 4970, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986, 4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998, 4999, 5000]
HardwareID	22	[4898, 4913, 4914, 4917, 4919, 4920, 4921, 4922, 4923, 4924, 4925, 4926, 4927, 4928, 4929, 4930, 4931, 4932, 4933, 4934, 4935, 4936, 4937, 4938, 4939, 4940, 4941, 4942, 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951, 4952, 4953, 4954, 4955, 4956, 4957, 4958, 4959, 4960, 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969, 4970, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986, 4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998, 4999, 5000]

Other tables that need taming:

CalibPmtHighGain	4	[5019, 5042, 5048, 5063]	## rollinggain entries may be prob
CalibPmtPedBias	1	[5034]	
CoordinateAd	1	[4974]	
CoordinateReactor	1	[4974]	
Reactor	1	[5065]	

### Table Rebuilding and Insertdates

Implications of slated approach for table rebuilding *Exceptional Operating Procedures for Major Changes*

1. changes INSERTDATE to the times of the re-insertions
2. simplest approach would even clump everything under one INSERTDATE **MUST AVOID THIS**

What to do with insertdates ?

1. staying honest with INSERTDATE, is important part of DBI *contract* do not want to kludge this
2. some tables **CableMap/HardwareID** hold little information in INSERTDATE
3. avoid everything going in under a single INSERTDATE
  - (a) when employing SOP loading with `dbaux.py` this requires separate **dybaux** commits (eg for each loaded file)
  - (b) a long sequence of **OVERRIDE** commits ? actually only the start scratching commit needs to be **OVER-RIDE**
4. other tables with significant information in INSERTDATE ?
  - (a) compromise and forgo rebuilding these : just code changes and future write protections

### Table Summary

Table	Recreation?	Notes
Ca-bleMap/HardwareID	OK	Many duplicated loads and little information in INSERTDATEs (as they recreate prior static history) <b>MOST IN NEED OF REBUILD</b>
CalibPmt-Spec	OK(?) Sept 30th load not verified	2 ctx with issues, but fairly localized
Cal-ibPmtHigh-Gain	not-OK rollinggain entries not easily recreatable	

**CableMap/HardwareID** Done:

1. developed blind duplication script/driver file, which succeeds to precisely recreate tables (warts and all)
2. developed simple all in one (de-duped) recreation script/driver file that uses the *G\*Fix* classes in *dyb-gaudi/Database/TableTests/TestCableMap/python/TestCableMap* to load de-duped driver in fixed mode
  - (a) see no collisions, ie no need to set manual VERSIONDATEs **db.py sssta checks confirm this**
3. payload digest comparisons (at last INSERTDATE) between de-duped *tmp\_offline\_db* and current *tmp\_copy\_db*
  - (a) no digest differences for *CableMap* and *HardwareID*, thus no difference between payloads returned (at last INSERTDATE)

## Todo:

1. chopped **dybaux-rebuild** commit script (one commit per input loadfile)
  - (a) **chopping is essential to avoid all under one INSERTDATE**
  - (b) test into an NTU repository

## Testing auxcommitted updating:

```

catdir=~/ntudybaux/catalog/tmp_offline_db
svn st $catdir

## just drops and recreates empty tables from dir with exports
cd t
../share/load_static.py -r 0:0 -l INFO --DROP ../recreate/driver_fix.txt

## rdumpcat empties into catalog, as clobbering must use OVERRIDE
db.py tmp_offline_db rdumpcat $catdir --OVERRIDE

svn st $catdir    ## check only expected changed to CableMap/HardwareID/LOCALSEQNO

## as find PhysAd diffs due to local PhysAd testing not in catalog, recreate tmp_offline_db and try a
db.py offline_db dump ~/offline_db.sql
db.py tmp_offline_db load ~/offline_db.sql

## 1st check that current offline_db and the catalog are in sync
db.py tmp_offline_db rdumpcat $catdir
##    ==> find not in sync due to prior dump with local PhysAd tests

## checkout fresh catalog wc
rm -rf $catdir/*      ## leaves .svn to allowing "svn up"
svn up $catdir/..

## OR from scratch approach
( rm -rf $catdir ; cd $(dirname $catdir) ; svn co http://dayabay.phys.ntu.edu.tw/svn/dybaux/catalog

## huh, svn checkout is hanging half way through

```

maybe corruption in recovered dybaux repo, as lsof is pointing to getting stuck on a single rev:

```

[root@cms02 log]# lsof | grep dybaux
httpd      22803  nobody   16r      REG          3,2   3056650   2667632 /var/scm/svn/dybaux/db/revs/4

```

from Trac <http://dayabay.phys.ntu.edu.tw/tracs/dybaux/changeset/4970> that is a full catalog recreation revision check-out eventually fails:

```

A    tmp_offline_db/SimPmtSpec/SimPmtSpecVld.csv
svn: REPORT of '/svn/dybaux/!svn/vcc/default': Could not read response body: Connection reset by peer

```

repeating, reveals that /var/scm/svn/dybaux/db/revs/4970 again gets stuck in  
craw \* <http://subversion.apache.org/faq.html#stuck-bdb-repos> \* <http://svnbook.red-bean.com/en/1.6/svn.reposadmin.maint.html#svn.reposadmin.maint.tk>

Note **crucial** points:

1. prevent other access to repo while using svnadmin, by `sv stop apache`
2. use the appropriate apache user, to avoid subsequent permission issues

Verification goes thru every revision, taking ~second for each:

```
sudo -u nobody svnadmin verify /var/scm/svn/dybaux
```

Tarball appears normal:

```
tar ztvf /data/var/scm/backup/dayabay/svn/dybaux/last/dybaux-5069.tar.gz
```

**CalibPmtSpec** The 2 problem **CalibPmtSpec** contexts

1. [http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1\\_simflag1\\_site32\\_subsite1\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNO](http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1_simflag1_site32_subsite1_task0/tmp_offline_db/vlut_cf_orderingSEQNO)
  - (a) Broad ambiguity (many TIMESTARTs) between (29L, 39L), but only for 2 INSERTDATES:

```
2011-06-27 13:34:26
2011-06-27 13:35:00
```

2. [http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1\\_simflag1\\_site1\\_subsite1\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNO](http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1_simflag1_site1_subsite1_task0/tmp_offline_db/vlut_cf_orderingSEQNO)
  1. Narrow ambiguity for TIMESTARTs:

```
2011-07-07 03:17:21
2011-07-08 06:23:13
2011-07-09 05:58:01
```

for 3 INSERTDATES:

```
(85L, 89L)      2011-07-26 07:22:30
(86L, 90L)      2011-07-26 07:22:56
(87L, 91L)      2011-07-26 07:23:23
```

**CalibPmtHighGain** Live(without rollback) ambiguity for **CalibPmtHighGain**, with 4 smoking guns from the  
INSERTDATE 2011-09-30 01:12:27 mixing SEQNO (66L, 390L)

- [http://belle7.nuu.edu.tw/dbiscan/CalibPmtHighGain/aggno-1\\_simflag1\\_site1\\_subsite1\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNO](http://belle7.nuu.edu.tw/dbiscan/CalibPmtHighGain/aggno-1_simflag1_site1_subsite1_task0/tmp_offline_db/vlut_cf_orderingSEQNO)
- [http://belle7.nuu.edu.tw/dbiscan/CalibPmtHighGain/aggno-1\\_simflag1\\_site1\\_subsite1\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNO](http://belle7.nuu.edu.tw/dbiscan/CalibPmtHighGain/aggno-1_simflag1_site1_subsite1_task0/tmp_offline_db/vlut_cf_orderingSEQNO)

For TIMESTARTs:

```
2011-07-31 18:59:05
2011-07-31 19:58:03
2011-07-31 19:59:05
2011-07-31 20:58:03
```

## Test Rebuilding

Test rebuilds using non-standard switched on .spec to determine:

1. how many collisions occur ?

2. can they be reduced by re-ordering loads ? eg TIMESTART ordering (backdating is know to increase degeneracy liklihood)
3. how to supply manual VERSIONDATEs ?

### Fixup Validity Approach

Appling a correction to all VERSIONDATEs to become TIMESTART floored reducing degeneracy etc.. is easy to do. It is also possible to devise *clever* schemes that attempt to replicate what DBI would have done with a changed policy.

While easy to go behind DBIs back and diddle with the VERSIONDATEs, it is not easy to know that the resulting changes in DBI results are OK. There are too many changes for it be feasible to confirm that the changes match the requirements of the table experts.

The extent and location of changes are visible from scans and summaries thereof.

### Current Trefixion Approach

Done by `DybDbi.vld.versiondate` (dybgaudi:Database/DybDbi/python/DybDbi/vld/versiondate.py )

1. copies all DBI tables from `tmp_offline_db` into `fix_offline_db` with `VERSIONDATE` changed to timestart floored scheme.
  - (a) uses `kls.GetTableProxy().QueryOverlayVersionDate` DBI call (with timestart floored option) to arrive at the `VERSIONDATE`
  - (b) this DBI call is done in the `fix_DB` with a `SEQNO asc` growing validity table

## 21.17.4 VLUT Comparisons

Summary tables created from the full DBI scan by `DybDbi.vld.vsmry` (dybgaudi:Database/DybDbi/python/DybDbi/vld/vsmry.py)

### Summary Examination

The dynamically derived version of this context summary is at \* <http://belle7.nuu.edu.tw/dbiscan/Summary/ctxsmry/>

An intermediate presentation listing contexts with differences, and including TIME and INSERTDATE ranges afflicted by differences is at \* <http://belle7.nuu.edu.tw/dbiscan/Summary/difctx/>

These summaries reference the full VLUT tables, such as \* [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggnol\\_simflag2\\_site2\\_subsite2\\_task0/tmp\\_offline\\_db/vlutorderingSEQNOasc\\_cf\\_orderingSEQNOdesc/](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggnol_simflag2_site2_subsite2_task0/tmp_offline_db/vlutorderingSEQNOasc_cf_orderingSEQNOdesc/)

### tmp\_offline\_db (copy of offline\_db)

Cells show number of ctxs with differences over total number of ctxs.

tn	vlut_cf_orderingSEQNOasc	vlut_cf_orderingSEQNOasc	vlut_cf_orderingSEQNOdesc
CableMap	16/35	19/35	19/35
Cal- ibFeeSpec	0/1	0/1	0/1
Cal- ibPmtHigh- Gain	0/6	0/6	0/6
Cal- ibPmtPed- Bias	0/1	0/1	0/1
CalibPmt- Spec	2/9	2/9	2/9
Coordi- nateAd	0/1	0/1	0/1
Coordi- nateReactor	0/1	0/1	0/1
Demo	1/1	1/1	1/1
FeeCa- bleMap	0/3	0/3	0/3
HardwareID	14/33	19/33	19/33
Reactor	0/6	0/6	0/6
SimPmtSpec	0/1	0/1	0/1
alltn	33/98	41/98	41/98

1. Impact of changing from default to controlled 2ndary ordering is apparent
2. issue is restricted to tables with significant overlaying : CableMap, CalibPmtSpec, HardwareID

fix\_offline\_db (with VERSIONDATE TIMESTART flooring)

tn	vlut_cf_orderingSEQNOasc	vlut_cf_orderingSEQNOdesc	vlut_cf_orderingSEQNOdesc.rst
CableMap	0/35	0/35	0/35
Cal-ibFeeSpec	1/1	1/1	1/1
Cal-ibPmtHigh-Gain	0/6	0/6	0/6
Cal-ibPmtPed-Bias	0/1	0/1	0/1
CalibPmtSpec	0/9	0/9	0/9
CoordinateAd	0/1	0/1	0/1
CoordinateReactor	0/1	0/1	0/1
Demo	0/1	0/1	0/1
FeeCableMap	0/3	0/3	0/3
HardwareID	0/33	0/33	0/33
Reactor	0/6	0/6	0/6
SimPmtSpec	0/1	0/1	0/1
alltn	1/98	1/98	1/98

1. very little ordering dependency as almost all degeneracy has been eliminated
2. a single pathological context, with a single pair of SEQNO causing issue.

- <http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/>
- [http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1\\_simflag1\\_site32\\_subsite1\\_task0/fix\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdesc.rst:2 \(97,99\)](http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1_simflag1_site32_subsite1_task0/fix_offline_db/vlut_cf_orderingSEQNOdesc.rst:2 (97,99))
- [http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1\\_simflag1\\_site32\\_subsite1\\_task0/fix\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdesc.rst:16 \(99,97\)](http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1_simflag1_site32_subsite1_task0/fix_offline_db/vlut_cf_orderingSEQNOdesc.rst:16 (99,97))

Two timestarts with only 40s between em:

```
mysql> select * from tmp_offline_db.CalibFeeSpecVld where SEQNO in (97,99) ;
+-----+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART          | TIMEEND          | SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATION |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 97    | 2010-01-07 06:45:28 | 2038-01-19 03:14:07 | 32      | 1      | 1      | 0    |             |
| 99    | 2010-01-07 06:44:12 | 2038-01-19 03:14:07 | 32      | 1      | 1      | 0    |             |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

```
mysql> select * from fix_offline_db.CalibFeeSpecVld where SEQNO in (97,99) ;
+-----+-----+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART          | TIMEEND          | SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATION |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 97    | 2010-01-07 06:45:28 | 2038-01-19 03:14:07 | 32      | 1      | 1      | 0    |             |
| 99    | 2010-01-07 06:44:12 | 2038-01-19 03:14:07 | 32      | 1      | 1      | 0    |             |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

1. check this corresponds to transfixion error

Check this corresponds to the error during transfixion of CalibFeeSpec, TODO: avoid this:

```
INFO:__main__:transfix_tab CalibFeeSpec
WARNING:__main__:transfixion of 89 sees collidingSeqno 83
WARNING:__main__:transfixion of 90 sees collidingSeqno 84
WARNING:__main__:transfixion of 91 sees collidingSeqno 77
WARNING:__main__:transfixion of 92 sees collidingSeqno 85
WARNING:__main__:transfixion of 94 sees collidingSeqno 78
WARNING:__main__:transfixion of 95 sees collidingSeqno 86
WARNING:__main__:transfixion of 96 sees collidingSeqno 87
WARNING:__main__:transfixion of 98 sees collidingSeqno 88
WARNING:__main__:transfixion of 99 sees collidingSeqno 97
WARNING:__main__:transfixion of 100 sees collidingSeqno 79
WARNING:__main__:transfixion of 101 sees collidingSeqno 80
```

**tmp\_offline\_db\_cf\_fix\_offline\_db**

It is straightforward to devise a fix, that has this better order change behavior (by removing degeneracy) ... but this is changing the results of DBI validity queries in some regions of (INSERTDATE, TIME).

Counting ctx with difference/totals when comparing *tmp\_offline\_db* with *fix\_offline\_db*, the corresponding orderings are used. Although that is fairly mute for *fix\_offline\_db* as it has very little extra ordering dependency, it is very relevant for *tmp\_offline\_db*

tn	vlut.rst	vlutorderingSEQNOasc.rst	vlutorderingSEQNOdesc.rst
CableMap	16/35	1/35	19/35
CalibFeeSpec	1/1	1/1	1/1
CalibPmtHighGain	0/6	0/6	0/6
CalibPmtPedBias	0/1	0/1	0/1
CalibPmtSpec	3/9	1/9	3/9
CoordinateAd	0/1	0/1	0/1
CoordinateReactor	0/1	0/1	0/1
Demo	1/1	1/1	1/1
FeeCableMap	0/3	0/3	0/3
HardwareID	14/33	0/33	19/33
Reactor	0/6	0/6	0/6
SimPmtSpec	0/1	0/1	0/1
alltn	35/98	4/98	43/98

1. vlutorderingSEQNOasc favors lower SEQNO in degenerate collisions (in tmp\_)
  - (a) this almost matches between tmp\_ and fix\_ **WHY?**
  - (b) because in fix\_ degeneracies are almost eliminated, the effect is that lower SEQNO results are peeking out that formerly were improperly overlaid
2. vlutorderingSEQNOdesc favors higher SEQNO in degenerate collisions #. I initially expected vlutorderingSEQNOdesc.rst would be most matched... #. But on further consideration, this is due to the breaking apart of degeneracy done by the fix
  - (a) timestart flooring used to create fix\_ almost eliminates degenerates (so in cases where
  - (b) SEQNOasc plucks lower SEQNO from degenerate tmp\_offline\_db ... which corresponds to the un-degenerated fix\_offline\_db
  - (a) all academic, the important one is vlut.rst as this comparing current DBI with intended future (modulo fix ordering, but that should not matter)

- (b) `tmp_offline_db/vlutorderingSEQNOdesc.rst` is kinda current *offline\_db* with degenerates fixed in place
3. Three/Four red herrings ?
    - (a) [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1\\_simflag2\\_site32\\_subsite1\\_task0/tmp\\_offline\\_db\\_cf\\_fix\\_offline\\_db/vlutdif:23](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site32_subsite1_task0/tmp_offline_db_cf_fix_offline_db/vlutdif:23)
    - (b) [http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1\\_simflag1\\_site32\\_subsite1\\_task0/tmp\\_offline\\_db\\_cf\\_fix\\_offline\\_db/vlutdif:94](http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1_simflag1_site32_subsite1_task0/tmp_offline_db_cf_fix_offline_db/vlutdif:94)
    - (c) [http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1\\_simflag1\\_site32\\_subsite2\\_task0/tmp\\_offline\\_db\\_cf\\_fix\\_offline\\_db/vlutdif:305](http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1_simflag1_site32_subsite2_task0/tmp_offline_db_cf_fix_offline_db/vlutdif:305) !!!
      - i. overlapping with very close TIMESTARTs is suspected to be implicated
  4. better to do the cross comparisons
    - (a) `tmp_offline_db/vlut cf fix_offline_db/SEQNOdesc`
    - (b) `tmp_offline_db/vlut cf fix_offline_db/SEQNOasc`
  5. BUT these are expected to match the first column however...
    - (a) need to present 35/98 ctxs with differences palatably ... ( need mismatch fractions within each )

## Observations

1. in `tmp_cf fix_` comparisons tis notable that `fix_` usually comes up with lower SEQNO : check generality of this

Sampling VLUT extracts

tmp\_offline\_db\_cf\_fix\_offline\_db vlut orderingSEQNOdesc insertdates:19 timestarts:18 ndif:19

in- sert- date	2009-03-16 11:27:11	2009-06-03 13:36:21	2010-12-07 14:16:49	2011-02-08 15:49:51	2011-02-22 17:38:17	2011-02-22 17:08:58	2011-02-22 17:07:46	2011-02-23 16:49:16	2011-03-25 16:31:19	2011-04-01 19:29:03	2011-04-18 23:42:20	2011-04-19 20:56:00	2011-05-03 09:35:09	2011-05-05 09:42:08	2011-05-23 08:22:19	2011-05-23 09:09:40	2011-06-01 00:00:00
2011-06-24 05:02:54	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29
2011-06-24 05:03:04	51	51	51	51	51	51	51	51	51	51	51	51	51	51	51	51	51
2011-06-24 05:03:39	51	59	73	73	73	73	73	73	73	123	139	155	171	187	187	187	187
2011-06-24 05:04:21	51	209	(209L,73L)	123	139	155	171	187	187	187	187						
2011-06-24 05:04:44	51	209	223	223	223	223	223	223	223	(223L,123L)	139	155	171	187	187	187	187
2011-06-24 05:05:08	51	209	223	223	223	223	223	223	223	(223L,123L)	139	155	171	187	187	187	187

The first pair:

```
mysql> select * from tmp_offline_db.HardwareIDVld where SEQNO in (209,73) ;
```

SEQNO	TIMESTART	TIMEEND	SITEMASK	SIMMASK	SUBSITE	TASK	AGGREGATION
73	2011-02-08 15:49:51	2038-01-19 03:14:07	1	2	2	0	
209	2010-12-07 19:14:20	2038-01-19 03:14:07	1	2	2	0	

2 rows in set (0.00 sec)

Observe:

1. a later insert is doing a backdated(earlier TIMESTART) override, this is prone to degeneracy issues
2. looks like the fix might be scrubbing an intended backdated override in this case ?
3. Earlier validity can leak forwards in time, but that is sometimes the desired.

## 21.17.5 Alternative to Recreat Rather Than Fix Approach

### CableMap/HardwareID

CableMap + HardwareID can be recreated (after quite a bit of detective work dealing with duplications and code changes impacting results: **takebogus** ) with:

```
../share/recreate_from_scratch.sh
```

1. <http://dayabay.ihep.ac.cn/tracs/dybsvn/ticket/880>
2. <http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/TableTests/TestCableMap/share/dlfcrs.sh?rev=12754>
3. <http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/Database/TableTests/TestCableMap/doc/notes.rst?rev=12373>
4. <http://dayabay.ihep.ac.cn/tracs/dybsvn/log/dybgaudi/trunk/DataModel/DataSvc/share/feeCableMap.txt> log of **feeCableMap.txt**

### Pumping dybaux history with `auxlog.py`

#. Most INSERTDATE groupings correspond to single TIMESTARTs, except h3#14, c2#2, c5#16 Notes:

1. the hN and cN correlate these auxlog commits with the INSERTDATE groupings below, indicating consistent SEQNO
  - (a) multi-timestart
  - (b) INSERTDATE correspondence only when both CableMap and HardwareID updated together
  - (c) need to make association between load files and commits
  - (d) INSERTDATE groupings works well for `tmp_copy_db` (a copy of `offline_db` ) due to artificial ff alignment, not so clear for local recreation in `tmp_offline_db`
  - (e) Most INSERTDATE groupings correspond to single TIMESTARTs, except h3:14, c2:2, c5:16

### Re-creation Discrepancy in FEC loading : resolved with `takebogus` option

are in lock step until hit the **fec**:

```
CableMap    : left TIMESTART 2011-05-23 08:22:19 [18][6 ][r12183:fecCableMap_fake_old.txt:viktor:6] ha
HardwareID  : left TIMESTART 2011-05-23 08:22:19 [18][6 ][r12183:fecCableMap_fake_old.txt:viktor:6] ha
```

1. confirmed that RPC bogosity code changes are changing selection of entries ?

### Relevant Tickets

1. `dybsvn:ticket:892` fixing `feeCableMap.txt` using `Database/TableTests/TestCableMap/share/fix_static_feeCableMap.py`. The fix is in `dybsvn:r12820`
2. `dybsvn:ticket:928` tracing warning
3. `dybsvn:ticket:937` bogus reporting due to not yet committed to DB
4. `dybsvn:ticket:940` splitting a dybaux commit

## Bogus Logic

Change in bogus logic confirmed to explain the differences

- <http://dayabay.ihep.ac.cn/tracs/dybsvn/log/dybgaudi/trunk/DataModel/Conventions/src/Detectors.cc>
- <http://dayabay.ihep.ac.cn/tracs/dybsvn/log/dybgaudi/trunk/DataModel/Conventions/src/Electronics.cc>
- <http://dayabay.ihep.ac.cn/tracs/dybsvn/changeset/13643/dybgaudi/trunk/DataModel/Conventions/src/Electronics.cc>
- <http://dayabay.ihep.ac.cn/tracs/dybsvn/changeset/13258/dybgaudi/trunk/DataModel/Conventions/src/Electronics.cc> returning undefined bool !
- <http://dayabay.ihep.ac.cn/tracs/dybsvn/changeset/13213/dybgaudi/trunk/DataModel/Conventions/src/Electronics.cc> RPC specific bogosity check

Relevant insertdate 2011-06-24 05:03:39 is before RPC bogosity developments circa end of July

## Whole table groupby INSERTDATE for overview

Grouping by INSERTDATE is informative for tmp\_copy\_db but not for the locally recreated, due to the fastforward clumping under a single INSERTDATE done by SOP.

Groupings:

```
mysql> select min(SEQNO),max(SEQNO),min(TIMESTART),max(TIMESTART),count(distinct(TIMESTART)) as dTS,
+-----+-----+-----+-----+-----+-----+
| min(SEQNO) | max(SEQNO) | min(TIMESTART) | max(TIMESTART) | dTS | INSERTDATE |
+-----+-----+-----+-----+-----+-----+
|          1 |          42 | 2009-03-16 11:27:43 | 2009-03-16 11:27:43 |    1 | 2011-06-24 05:02:54 |
|          43 |          58 | 2009-06-03 21:36:27 | 2009-06-03 21:36:27 |    1 | 2011-06-24 05:03:04 |
|          59 |         208 | 2010-12-07 19:14:20 | 2011-05-23 13:09:43 |   14 | 2011-06-24 05:03:39 |
|         209 |         222 | 2010-12-07 19:14:20 | 2010-12-07 19:14:20 |    1 | 2011-06-24 05:04:21 |
|         223 |         236 | 2011-02-08 15:49:51 | 2011-02-08 15:49:51 |    1 | 2011-06-24 05:04:44 |
|         237 |         248 | 2011-02-22 12:38:11 | 2011-02-22 12:38:11 |    1 | 2011-06-24 05:05:08 |
|         249 |         254 | 2011-02-22 17:08:51 | 2011-02-22 17:08:51 |    1 | 2011-06-24 05:05:34 |
|         255 |         260 | 2011-02-22 18:07:45 | 2011-02-22 18:07:45 |    1 | 2011-06-24 05:05:58 |
|         261 |         266 | 2011-02-23 10:49:36 | 2011-02-23 10:49:36 |    1 | 2011-06-24 05:06:24 |
|         267 |         272 | 2011-03-25 19:31:49 | 2011-03-25 19:31:49 |    1 | 2011-06-24 05:06:52 |
|         273 |         288 | 2011-04-01 17:29:23 | 2011-04-01 17:29:23 |    1 | 2011-06-24 05:07:18 |
|         289 |         304 | 2011-04-18 03:42:40 | 2011-04-18 03:42:40 |    1 | 2011-06-24 05:07:47 |
|         305 |         320 | 2011-04-19 23:56:10 | 2011-04-19 23:56:10 |    1 | 2011-06-24 05:08:15 |
|         321 |         336 | 2011-05-03 02:35:09 | 2011-05-03 02:35:09 |    1 | 2011-06-24 05:08:46 |
|         337 |         352 | 2011-05-05 17:42:22 | 2011-05-05 17:42:22 |    1 | 2011-06-24 05:09:17 |
|         353 |         355 | 2011-05-23 08:22:19 | 2011-05-23 08:22:19 |    1 | 2011-06-24 05:09:47 |
|         356 |         358 | 2011-05-23 13:09:43 | 2011-05-23 13:09:43 |    1 | 2011-06-24 05:10:23 |
|         359 |         372 | 2010-12-07 19:14:20 | 2010-12-07 19:14:20 |    1 | 2011-06-28 02:26:02 |
|         373 |         386 | 2011-06-01 00:00:00 | 2011-06-01 00:00:00 |    1 | 2011-08-09 06:35:49 |
+-----+-----+-----+-----+-----+-----+
19 rows in set (0.00 sec)
```

```
mysql> select min(SEQNO),max(SEQNO),min(TIMESTART),max(TIMESTART),count(distinct(TIMESTART)) as dTS,
+-----+-----+-----+-----+-----+-----+
| min(SEQNO) | max(SEQNO) | min(TIMESTART) | max(TIMESTART) | dTS | INSERTDATE |
+-----+-----+-----+-----+-----+-----+
|          1 |          42 | 2009-03-16 11:27:43 | 2009-03-16 11:27:43 |    1 | 2011-06-24 05:02:54 |
|          43 |          59 | 2009-06-03 21:36:27 | 2009-12-27 23:52:51 |    2 | 2011-06-24 05:03:04 |
|          60 |          60 | 2009-12-27 23:52:51 | 2009-12-27 23:52:51 |    1 | 2011-06-24 05:03:14 |
|          61 |          61 | 2010-03-02 11:34:36 | 2010-03-02 11:34:36 |    1 | 2011-06-24 05:03:24 |
```



Listing ctx and VLUT regions of payload change between legacy and extra ordering fixed **SEQNO desc**

1. [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1\\_simflag2\\_site1\\_subsite7\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site1_subsite7_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
  - (a) INSERTDATE 2011-06-24 05:09:47 3 ambi-cells (250L, 437L) from TIMES: 2011-05-23 13:09:43 2011-06-01 00:00:00 2011-06-22 03:02:52
2. [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1\\_simflag2\\_site2\\_subsite7\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site2_subsite7_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
  - (a) INSERTDATE 2011-06-24 05:09:47 3 ambi-cells (248L, 435L) from TIMES: 2011-05-23 13:09:43 2011-06-01 00:00:00 2011-06-22 03:02:52
3. [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1\\_simflag2\\_site4\\_subsite7\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site4_subsite7_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
  - (a) INSERTDATE 2011-06-24 05:09:47 3 ambi-cells (249L, 436L) TIMES 2011-05-23 13:09:43 2011-06-01 00:00:00 2011-06-22 03:02:52
4. [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1\\_simflag2\\_site2\\_subsite5\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site2_subsite5_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
  - (a) INSERTDATE 2011-06-24 05:04:21 1 ambi-cell (92L,268L) 2011-02-08 15:49:51
5. [http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1\\_simflag2\\_site1\\_subsite5\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site1_subsite5_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
  - (a) INSERTDATE 2011-06-24 05:04:21 1 ambi-cell 87L, 263L) 2011-02-08 15:49:51

## HardwareID

- <http://belle7.nuu.edu.tw/dbiscan/HardwareID/>
1. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site1\\_subsite6\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site1_subsite6_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    - (a) I 2011-06-24 05:04:21 T 2011-02-08 15:49:51 (80L, 217L)
  2. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site1\\_subsite7\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site1_subsite7_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    - (a) I 2011-06-24 05:09:47 T 2011-05-23 13:09:43 2011-06-01 00:00:00 (208L, 355L)
  3. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site1\\_subsite5\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site1_subsite5_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    - (a) I 2011-06-24 05:04:21 T 2011-02-08 15:49:51 (79L, 216L)
  4. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site2\\_subsite5\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site2_subsite5_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    1. I 2011-06-24 05:04:21 T 2011-02-08 15:49:51 (84L, 221L)
  1. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site2\\_subsite6\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site2_subsite6_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    1. I 2011-06-24 05:04:21 T 2011-02-08 15:49:51 (74L, 210L)
  1. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site2\\_subsite7\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site2_subsite7_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    1. I 2011-06-24 05:09:47 T 2011-05-23 13:09:43 2011-06-01 00:00:00 (206L, 353L)
  1. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site4\\_subsite5\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site4_subsite5_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    1. I 2011-06-24 05:04:21 T 2011-02-08 15:49:51 (76L, 212L)
  1. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site4\\_subsite6\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site4_subsite6_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    1. I 2011-06-24 05:04:21 T 2011-02-08 15:49:51 (75L, 211L)
  1. [http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1\\_simflag2\\_site4\\_subsite7\\_task0/tmp\\_offline\\_db/vlut\\_cf\\_orderingSEQNOdes](http://belle7.nuu.edu.tw/dbiscan/HardwareID/aggno-1_simflag2_site4_subsite7_task0/tmp_offline_db/vlut_cf_orderingSEQNOdes)
    1. I 2011-06-24 05:09:47 T 2011-05-23 13:09:43 2011-06-01 00:00:00 (207L, 354L)

## About the SOP

The SOP is sourced from reStructuredText in `dybgaudi:Documentation/OfflineUserManual/tex/sop`, and html and pdf versions are derived as part of the automated Offline User Manual build. For help with building see *Build Instructions for Sphinx based documentation*



# ADMIN OPERATING PROCEDURES FOR SVN/TRAC/MYSQL

**Release** 17726

**Date** August 03, 2012

This documentation attempts to provide the practical knowledge needed to perform admin operations.

1. *Env Repository : Admin Infrastructure Sources*
2. *Source Code Management*
3. *SSH Setup For Automated Rsync transfers*

Detailed table of contents:

## 22.1 Env Repository : Admin Infrastructure Sources

- Installing Env
- Heirarchy of Bash Functions
- Node Characterisation

Most of the development history and documentation (commits, tickets, wikipages) of the admin infrastructure is managed in the **env** SVN/Trac instance at NTU [env:wiki:WikiStart](#) . This includes:

1. scripts for backup/recovery of the MySQL DB `offline_db`
2. scripts for backup/recovery of the SVN repository and Trac instance
3. patches against Trac and various Trac plugins such as *bitten*

Using the Trac tags is the best way to locate this documentation, for example:

- `env:tag:SCM` provides a list of Source Code Management related pages

### 22.1.1 Installing Env

**bash shell is mandatory**

The Admin sources are composed primarily of a large number of bash functions. Use with any other shell is **condemned**, and will not be supported in any way.

Follow the instructions on the front page [env:wiki:WikiStart](http://env.wiki:WikiStart) to install **env**, starting from:

```
cd $HOME ; svn checkout http://dayabay.phys.ntu.edu.tw/repos/env/trunk/ env
cd $HOME/env ; svn update
```

Hook up **env** to the bash shell of the user by adding the below to the `.bash_profile`:

```
export ENV_HOME=$HOME/env
env-() { . $ENV_HOME/env.bash && env-env $* ; }
env-    ## precursor function
```

## 22.1.2 Heirarchy of Bash Functions

The *env-precursor* function defines other precursor functions such as

1. scm-
2. scm-backup-
3. db-

These precursor functions are very simple, all following the form of defining a set of related functions and setting up any environment with eg `scm-env`:

```
scm-() { . $(env-home)/scm/scm.bash && scm-env $* ; }
scm-backup-() { . $(env-home)/scm/scm-backup.bash && scm-backup-env $* ; }
```

Certain conventions are followed:

1. precursor function names end with a hyphen: -
2. functions defined by precursor functions like `scm-` are named to extend that name, ie `scm-create` and `scm-wipe`
3. certain standard functions are included for all eg `scm-vi`, `scm-backup-vi`, `db-vi` all open the functions source in the **vi** editor
4. other standard endings include `*-usage`, `*-source` etc

Conventional naming structure provides convenient tab completion:

```
[blyth@belle7 ~]$ scm-
[blyth@belle7 ~]$ scm-<TAB>
scm-                scm-create          scm-env            scm-postcommit-   scm-rename
scm-backup-        scm-eggcache       scm-postcommit    scm-postcommit-test  scm-source
```

**interactively examine unfamiliar functions before usage**

The bash functions themselves are always the best and most up-to-date documentation. It is **very** important to understand what functions are going to do before usage. For example `scm-wipe <name>` deletes the svn repository and Trac instance called **name**.

The `env-` function defines some aliases such as `t` which provides interactive access to function definitions from the commandline:

```
[blyth@belle7 ~]$ t t
t is aliased to `type'
[blyth@belle7 ~]$ t scm-create
scm-create is a function
scm-create ()
{
    local msg===" $FUNCNAME :";
    local name=$1;
    shift;
    [ -z "$name" ] && echo $msg an instance name must be provided && return 1;
    svn-;
    svn-create $name $*;
    trac-;
    trac-create $name
}
[blyth@belle7 ~]$ t scm-vi
scm-vi is a function
scm-vi ()
{
    vim $(scm-source)
}
[blyth@belle7 ~]$ scm-source
/data1/env/local/env/home/scm/scm.bash
[blyth@belle7 ~]$ scm-vi
```

As far as possible the functions seek to abstract away node specific details, such as directory paths and basis application layout (eg different versions of apache have files in different places). The functions should shield the user from these node specifics.

**Warning:** the functions are used on many different nodes, this requires care to avoid breaking things for other nodes by ignoring the node agnostic approach

### 22.1.3 Node Characterisation

Node abstraction is achieved by node detection and the setting of standard envvars such as `NODE_TAG` by the `elocal-` precursor:

```
[blyth@belle7 home]$ t elocal-
elocal- is a function
elocal- ()
{
    . $(env-home)/base/local.bash && local-env $*
}
[blyth@belle7 home]$ t local-env
local-env is a function
local-env ()
{
    local dbg=${1:-0};
    local msg===" $FUNCNAME :";
    [ "$dbg" == "1" ] && echo $msg;
    export SOURCE_NODE="g4pb";
    export SOURCE_TAG="G";
    export LOCAL_ARCH=$(uname);
```

```
export LOCAL_NODE=$(local-node);
export NODE_TAG=$(local-nodetag);
export BACKUP_TAG=$(local-backup-tag);
export SUDO=$(local-sudo);
export SYSTEM_BASE=$(local-system-base);
export LOCAL_BASE=$(local-base);
export ENV_PREFIX=$(local-prefix);
export VAR_BASE=$(local-var-base);
export SCM_FOLD=$(local-scm-fold);
export VAR_BASE_BACKUP=$(local-var-base $BACKUP_TAG);
export USER_BASE=$(local-user-base);
export OUTPUT_BASE=$(local-output-base);
local-userprefs
}
```

---

**Note:** simple usage of echo from bash functions to return values to other functions, requires care regards extraneous output

---

The **NODE\_TAG** is very widely used for branching on node specifics:

```
[blyth@belle7 home]$ local-nodetag
N
```

For example the `local-scm-fold` emits the path used as the base for backups:

```
[blyth@belle7 home]$ local-scm-fold
/var/scm

[blyth@belle7 home]$ t local-scm-fold
local-scm-fold is a function
local-scm-fold ()
{
    case ${1:-$NODE_TAG} in
        WW)
            echo /home/scm
            ;;
        *)
            echo $(local-var-base $*)/scm
            ;;
    esac
}
[blyth@belle7 home]$ t local-var-base
local-var-base is a function
local-var-base ()
{
    local t=${1:-$NODE_TAG};
    case $t in
        U)
            echo /var
            ;;
        P)
            echo /disk/d3/var
            ;;
        G1)
            echo /disk/d3/var
            ;;
    ...etc...
```

A `case` function is used on the `NODE_TAG` to locate the different places on

## 22.2 Source Code Management

- Backups with `scm-backup-all`
- Offbox Transfers with `scm-backup-rsync`
- Recovery using `scm-recover-all`
- Adding a new target node for backups
  - Node characterization
  - Placement of SSH keys
  - Add target tag to `BACKUP_TAG` of source node

Backups/transfers and recovery of Trac/SVN instances are implemented in bash functions `scm-backup-*` `env:source:trunk/scm/scm-backup.bash` to interactively examine these functions use the normal `env` discovery approach *Heirarchy of Bash Functions*, for example:

```
env-
scm-backup-
scm-backup-<TAB>
type scm-backup-all
t scm-backup-trac      ## env- defines alias "t" for type
scm-backup-vi
```

### 22.2.1 Backups with `scm-backup-all`

Performs `scm-backup-trac` and `scm-backup-repo` for all instances/repositories under `$SCM_FOLD/{repos,svn,tracs}`. The `SCM_FOLD` is node dependent: `/home/scm` on the `dybsvn` server, `/var/scm` on the `env` server. Such node dependent details are defined in `local-*` bash functions.

The backups are performed using *hotcopy* techniques/scripts provided by the Trac and Subversion projects, and result in tarballs in dated folders beneath `$SCM_FOLD/backup/$LOCAL_NODE` where `LOCAL_NODE` is eg `dayabay` or `cms01` : the node on which the instances reside.

Additional tasks are performed by `scm-backup-all`:

1. `$(svn-setupdir)` which contains config details such as users lists are backed up by `scm-backup-folder` into a separate tarball
2. a digest of each tarball is made and the resulting 32 char hex code is stored in `.dna` sidecar files
3. tarballs are purged by `scm-backup-purge` to retain a configured number
4. locks are planted and cleared during backups

### 22.2.2 Offbox Transfers with `scm-backup-rsync`

**SSH Node tags**

Node tags are short aliases for SSH connected nodes such as **C** that are listed in the `~/.ssh/config` file of form:

```
host C
    user blyth
    hostname 140.112.101.190
    protocol 2
```

First the SSH agent on the source node is checked with `ssh--agent-check`, then for each target node tag listed in `$BACKUP_TAG`, an `rsync` command is composed and run. The target directory for each node is provided by an echoing bash function `scm-backup-dir`:

```
[blyth@cms02 ~]$ scm-backup-dir          ## defaults to current node
/var/scm/backup
[blyth@cms02 ~]$ scm-backup-dir C        ## knows about other nodes
/data/var/scm/backup
[blyth@cms02 ~]$ scm-backup-dir N
/var/scm/backup
```

Locks are planted and cleared during transfers in order to avoid usage of incomplete tarballs.

When the target account has the **env** functions installed additional DNA checks are performed following this transfer. This recalculate the tarball digests on the target machines and compares values with those written in the sidecar `.dna` files.

The most problematic part of adding new nodes as backup targets, is usually configuring the SSH connections that allows passwordless `rsync` transfers to be performed using SSH keys *SSH Setup For Automated Rsync transfers*.

### 22.2.3 Recovery using `scm-recover-all`

Requires a **fromnode** argument, recovers all Trac/SVN tarballs with `scm-recover-repo` and users with `scm-recover-users`, performs apache required ownership changes and synchronises the trac instances with corresponding svn repositories `scm-backup-synctrac`.

### 22.2.4 Adding a new target node for backups

The administrator of the source node will need to:

1. create a new node tag in `~/.ssh/config` with the nodename and user identity of the new target, an unused tag must be chosen: check with `local-vi` to see tags that have been used already

#### Node characterization

The target node administrator will need to update the **env** node characterisation of the new node, using the `local-vi` function and commit changes into the **env** repository. The changes required are mostly just additional lines in case statements, providing for example:

1. `local-scm-fold`
2. `local-var-base` used by `local-scm-fold`

## Placement of SSH keys

### If policy or lack of trust prevents such intimacy

If the **target** node administrator is not willing to afford such trust in the **source** node administrator, alternatives are possible using a special **scponly** [env:wiki:RestrictedShell](#) but this is not straightforward to setup.

Source account public keys `~/.ssh/id_dsa.pub` or `~/.ssh/id_rsa.pub` need to be **appended** to the target account `~/.ssh/authorized_keys2` on the target node. This affords access from the source account to the target account allowing the `scm-backup-rsync` to automatically perform its transfers.

### Add target tag to `BACKUP_TAG` of source node

Once the **env** working copy is updated on the source node to pick up the new target node characterization the new backup node for the source node is configured by modifying the `case` statement in the `local-backup-tag` function.

## 22.3 SSH Setup For Automated Rsync transfers

The basics of setting up passwordless SSH are described in [env:wiki>PasswordLessSSH](#)

### About the AOP

The AOP is sourced from reStructuredText in [dybgaudi:Documentation/OfflineUserManual/tex/aop](#), and html and pdf versions are derived as part of the automated Offline User Manual build. For help with building see *Build Instructions for Sphinx based documentation*



# NUWA PYTHON API

**Release** 17726

**Date** August 03, 2012

See *Autodoc : pulling reStructuredText from docstrings* for a description of how this python API documentation was extracted from source docstrings.

## 23.1 DB

### 23.1.1 DybPython.db

\$Id: db.py 16813 2012-04-27 08:01:24Z blyth \$

DB operations performed via MySQLdb:

```
./db.py [options] <dbconf> <cmd>
```

Each invocation of this script talks to a single database only. A successful connection to “sectname” requires the config file (default `~/my.cnf`) named section to provide the below keys, eg:

```
[offline_db]
host = dybdb1.ihep.ac.cn
user = dayabay
password = youknowit
database = offline_db
```

```
[tmp_username_offline_db]
...
```

For a wider view of how `db.py` is used see *DB Table Updating Workflow*

### TODO

1. dry run option to report commands that would have been used without doing them
2. better logging and streamlined output

### Required Arguments

**dbconf** the name of the section in `~/my.cnf` that specifies the host/database/user/password to use in making connection to the mysql server

**cmd** perform command on the database specified in the prior argument. NB some commands can only be performed locally, that is on the same node that the MySQL server is running on.

### command summary

Com-mand	Action	Note
dump	performs mysqldump, works remotely	special LOCALSEQNO handling
load	loads mysqldump, works remotely	very slow when done remotely, insert statement for every row
rdump-cat	dumps ascii catalog, works remotely	duplicates dumpcat output using low level _mysql uses LOCALSEQNO merging
rloadcat	loads ascii catalog, works remotely	mysqlimport implementation,
rcmpcat	compare ascii catalog with DB	readonly command

### former commands

Command	Action	Note
dumpcat	dumps ascii catalog, LOCAL ONLY	SELECT ... INTO OUTFILE
loadcat	loads ascii catalog, LOCAL ONLY	LOAD DATA LOCAL INFILE ... INTO TABLE

Former **loadcat** and **dumpcat** can be mimicked with `--local` option of **rdumpcat** and **rloadcat**. These are for expert usage only into self administered database servers.

### using db.py in standalone manner (ie without NuWa)

This script is usable with any recent python which has the mysql-python (1.2.2 or 1.2.3) package installed.

Check your python and mysql-python with:

```
which python
python -V
python -c "import MySQLdb as _ ; print __version__ "
```

Checkout `DybPython/python/DybPython` in order to access `db.py`, `dbcmd.py` and `dbconf.py`, for example with

```
cd
svn co http://dayabay.ihep.ac.cn/svn/dybsvn/dybgaudi/trunk/DybPython/python/DybPython
chmod u+x DybPython/db.py
```

Use as normal:

```
~/DybPython/db.py --help
~/DybPython/db.py offline_db count
```

### checkout offline\_db catalog from dybaux

Example, checkout OR update the catalog:

```
mkdir ~/dybaux
cd ~/dybaux
svn co http://dayabay.ihep.ac.cn/svn/dybaux/catalog
```

OR

```
cd ~/dybaux/catalog
svn up
```

rdumpcat tmp\_offline\_db into dybaux working copy:

```
db.py tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
```

### Test usage of serialized ascii DB

Get into environment and directory of pkg `dybgaudi:Database/DybDbi` Modify the config to use ascii DB, for an example see `dybgaudi:Database/DybDbi/tests/test_calibpmtspec.py`

### rloadcat testing, DB time machine

**Warning:** `forced_rloadcat` is for testing only, it skips checks and ploughs ahead with the load, also `--DROP` option drops and recreates tables

Fabricate a former state of the DB using `forced_rloadcat` and an earlier revision from dybaux, with:

```
## get to a clean revision of catalog (blowing away prior avoids conflicts when doing that)
rm -rf ~/dybaux/catalog/tmp_offline_db ; svn up -r 4963 ~/dybaux/catalog/tmp_offline_db

## forcefully propagate that state into the tmp_offline_db
./db.py tmp_offline_db forced_rloadcat ~/dybaux/catalog/tmp_offline_db --DROP

## compare DB and catalog .. no updates should be found
./db.py tmp_offline_db rcmpcat ~/dybaux/catalog/tmp_offline_db

## wind up the revision
rm -rf ~/dybaux/catalog/tmp_offline_db ; svn up -r 4964 ~/dybaux/catalog/tmp_offline_db

## compare DB and catalog again ... updates expected, check timeline diffs
./db.py tmp_offline_db rcmpcat ~/dybaux/catalog/tmp_offline_db

## test rloadcat operation and check diffs afterwards
./db.py tmp_offline_db rloadcat ~/dybaux/catalog/tmp_offline_db
./db.py tmp_offline_db rcmpcat ~/dybaux/catalog/tmp_offline_db
```

### 23.1.2 DybPython.db.DB

```
class DybPython.db.DB (sect=None, opts={}, **kwa)
    Bases: object
```

Initialize config dict corresponding to section of config file

**Parameters** `sect` – section in config file

**allseqno**

Provides a table name keyed dict containing lists of all SEQNO in each Vld table. The tables included correspond to the read DBI tables (namely those in LOCALSEQNO)

**check\_** (\*args, \*\*kwa)

check connection to DB by issuing a SELECT of info functions such as DATABASE() and CURRENT\_USER() command

**check\_allseqno** ()**check\_seqno** ()

Compares the LASTUSEDSEQNO entries read into `self._seqno` with the `max(SEQNO)` results of selects on the DB payload and validity tables.

**count\_** (\*args, \*\*kwa)

List table counts of all tables in database, usage example:

```
db.py offline_db count
```

`offline_db` is `~/ .my.cnf` section name specifying host/database/user/password

**desc** (tab)

Header line with table definition in .csv files shift the pk definition to the end

**describe** (tab)**classmethod docs** ()

collect the docstrings on command methods identified by naming convention of ending with `_` (and not starting with `_`)

**dump\_** (\*args, \*\*kwa)

Dumps tables from any accessible database into a mysqldump file. Usage:

```
db.py                               offline_db dump /tmp/offline_db.sql  ## without -t a default li
db.py -t CableMap,HardwareID offline_db dump /tmp/offline_db.sql
tail -25 /tmp/offline_db.sql          ## checking tail, look for
```

Use the `-t/--tselect` option with a comma delimited list of to select payload tables. Corresponding validity tables and the `LOCALSEQNO` table are included automatically.

The now default `-d/--decoupled` option means that the `LOCALSEQNO` table is dumped separately and only contains entries corresponding to the selected tables. The decoupled dump can be loaded into `tmp_offline_db` without any special options, as the table selection is reflected within the dump:

```
db.py tmp_offline_db load /tmp/offline_db.sql
```

Partial dumping is implemented using:

```
mysqldump ... --where="TABLENAME IN ('*', 'CableMap', 'HardwareID')" LOCALSEQNO
```

**fabseqno**

Summarizes `db.allseqno`, by fabricating a dict keyed by table name containing the number of Vld SEQNO (from length of values in `db.allseqno`)

This dict can be compared with `db.seqno`, which is obtained from the `LASTUSEDSEQNO` entries in the `LOCALSEQNO` table:: Assuming kosher DBI handling of tables this fabricated dict `db.fabseqno` should match `db.seqno`, meaning that SEQNO start from 1 and have no gaps.

```
In [1]: from DybPython import DB
```

```
In [2]: db = DB("tmp_fake_offline_db")
```

```

In [3]: db.seqno    ## queries the LOCALSEQNO table in DB
Out[3]:
{'CableMap': 213,
 'CalibFeeSpec': 113,
 'CalibPmtSpec': 29,
 'FeeCableMap': 3,
 'HardwareID': 172}

In [4]: db.fabseqno    ## a summarization of db.allseqno
Out[4]:
{'CableMap': 213,
 'CalibFeeSpec': 111,
 'CalibPmtSpec': 8,
 'FeeCableMap': 3,
 'HardwareID': 172}

In [5]: db.miscreants    ## assertions avoided by miscreant status
Out[5]: ('CalibPmtSpec', 'CalibFeeSpec')

```

**forced\_reloadcat\_(\*args, \*\*kwa)**

Forcible loading of a catalog ... FOR TESTING ONLY

**get\_allseqno()**

Provides a table name keyed dict containing lists of all SEQNO in each Vid table The tables included correspond to the read DBI tables (namely those in LOCALSEQNO)

**get\_fabseqno()**

Summarizes db.allseqno, by fabricating a dict keyed by table name containing the number of Vid SEQNO (from length of values in db.allseqno)

This dict can be compared with db.seqno, which is obtained from the LASTUSEDSEQNO entries in the LOCALSEQNO table:: Assuming kosher DBI handling of tables this fabricated dict db.fabseqno should match db.seqno, meaning that SEQNO start from 1 and have no gaps.

```

In [1]: from DybPython import DB

In [2]: db = DB("tmp_fake_offline_db")

In [3]: db.seqno    ## queries the LOCALSEQNO table in DB
Out[3]:
{'CableMap': 213,
 'CalibFeeSpec': 113,
 'CalibPmtSpec': 29,
 'FeeCableMap': 3,
 'HardwareID': 172}

In [4]: db.fabseqno    ## a summarization of db.allseqno
Out[4]:
{'CableMap': 213,
 'CalibFeeSpec': 111,
 'CalibPmtSpec': 8,
 'FeeCableMap': 3,
 'HardwareID': 172}

In [5]: db.miscreants    ## assertions avoided by miscreant status
Out[5]: ('CalibPmtSpec', 'CalibFeeSpec')

```

**get\_seqno()**

SEQNO accessor, reading and checking is done on first access to `self.seqno` with

```
db = DB()
print db.seqno    ## checks DB
print db.seqno    ## uses cached
del db._seqno     ## force a re-read and check
print db.seqno
```

**has\_table** (*tn*)

**Parameters** *tn* – table name

**Return exists** if table exists in the DB

**load\_** (*\*args, \*\*kwa*)

Loads tables from a mysqldump file into a target db, the target db is configured by the parameters in the for example `tmp_offline_db` section of the config file. For safety the name of the configured target database must begin with `tmp_`

---

**Note:** CAUTION IF THE TARGET DATABASE EXISTS ALREADY IT WILL BE DROPPED AND RECREATED BY THIS COMMAND

---

Usage example:

```
db.py tmp_offline_db load /tmp/offline_db.sql
```

**loadcsv** (*cat, tn*)

**Parameters**

- **cat** – AsciiCat instance
- **tn** – string payload table name or LOCALSEQNO

**mysql** (*\*args, \*\*kwa*)

**noop\_** (*\*args, \*\*kwa*)

Do nothing command, allowing to just instantiate the DB object and provide it for interactive prodding, eg:

```
~/v/db/bin/ipython -- ~/DybPython/db.py tmp_offline_db noop
```

```
In [1]: db("show tables")    ## high level
```

```
In [2]: db.llconn.query("select * from CalibPmtSpecVld")    ## lowlevel _mysql
```

```
In [3]: r = db.conn.store_result()
```

This also demonstrates standalone `db.py` usage, assuming svn checkout:

```
svn co http://dayabay.ihep.ac.cn/svn/dybsvn/dybgaudi/trunk/DybPython/python/DybPython
```

**optables**

List of tables that commands such as **rdumpcat** perform operations on, outcome depends on:

- 1.table selection from the `-t/--select` option
- 2.decoupled option setting
- 3.DBCONF section name, where name **offline\_db** is regarded as special

The default value of the table selection option constitutes the current standard set of DBI tables that should be reflected in the dybaux catalog.

When following the SOP in the now default “decoupled” mode the **offline\_db** `rdumpcat` needs to abide by the table selection in force, whereas when dumping from **tmp\_offline\_db** onto a dybaux checkout need to dump all of the subset. Rather than the default table selection.

This special casing avoids the need for the `-t` selection when `rdumpcating tmp_offline_db`

**outfile** (*tab*)

Path of raw outfile as dumped by `SELECT ... INTO OUTFILE`

**paytables**

list of selected DBI payload tables

**predump** ()

Checks performed before : **dump, dumpcat, rdumpcat**

**rcmpcat**\_ (\*args, \*\*kwa)

Just dumps a comparison between target DB and ascii catalog, allowing the actions an **rloadcat** will do to be previewed.

Compares DBI vitals such as `LASTUSEDSEQNO` between a DBI database and a DBI ascii catalog, usage:

```
./db.py tmp_offline_db rcmpcat ~/dybaux/catalog/tmp_offline_db
```

**rdumpcat**\_ (\*args, \*\*kwa)

Dumps DBI tables and merges `LOCALSEQNO` from `tmp_offline_db` into a pre-existing ascii catalog. Usage:

```
db.py -d tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db ## -d/--decoupled i
db.py tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
```

```
svn status ~/dybaux/catalog/tmp_offline_db ## see whats change
```

Features of the default `-d/--decoupled` option:

- 1.requires dumping into a pre-existing catalog
- 2.subset of tables present in the DB are dumped
- 3.partial `LOCALSEQNO.csv` is merged into the pre-existing catalog `LOCALSEQNO.csv`
- 4.performs safe writes, if the merge fails detritus files with names ending `.csv._safe` and `.csv._merged` will be left in the working copy

With alternate `-D/--nodecoupled` option must ensure that the table selection is appropriate to the content of the DB:

```
db.py -D -t CableMap,HardwareID offline_db rdumpcat ~/offline_db
```

To obtain the dybaux SVN catalog:

```
mkdir ~/dybaux
cd ~/dybaux ;
svn co http://dayabay.ihep.ac.cn/svn/dybaux/catalog
```

The ascii catalog is structured

```
~/dybaux/catalog/tmp_offline_db
    tmp_offline_db.cat
    CalibFeeSpec/
        CalibFeeSpec.csv
        CalibFeeSpecVld.csv
    CalibPmtSpec/
        CalibPmtSpec.csv
```

```

        CalibPmtSpecVld.csv
    ...
LOCALSEQNO/
    LOCALSEQNO.csv

```

The .csv files comprise a single header line with the table definition and remainder containing the row data.

#### ADVANCED USAGE OF ASCII CATALOGS IN CASCADES

The resulting catalog can be used in a DBI cascade by setting DBCONF to:

```
tmp_offline_db_ascii:offline_db
```

Assuming a section:

```

[tmp_offline_db_ascii]
host = localhost
user = whatever
password = whatever
db = tmp_offline_db#/path/to/catname/catname.cat

```

NB from `dybsvn:r9869` `/path/to/catname/catname.cat` can also be a remote URL such as

```

http://dayabay:youknowit\@dayabay.ihep.ac.cn/svn/dybaux/trunk/db/cat/zhe/trial/trial.cat
http://dayabay:youknowit\@dayabay.ihep.ac.cn/svn/dybaux/!svn/bc/8000/trunk/db/cat/zhe/trial/

```

When stuffing basic authentication credentials into the URL it is necessary to backslash escape the “@” to avoid confusing DBI(TUrl) Note the use of “!svn/bc/NNNN” that requests apache mod\_dav\_svn to provide a specific revision of the catalog, rather than the default latest.

#### ADVANTAGES OF CATALOG FORMAT OVER MYSQLDUMP SERIALIZATIONS

- effectively native DBI format that can be used in ascii cascades allowing previewing of future database after updates are made
- very simple/easily parsable .csv that can be read by multiple tools
- very simple diffs (DBI updates should be contiguous additional lines), unlike mysqldump, this means efficient storage in SVN
- no-variants/options that change the format (unlike mysqldump)
- no changes between versions of mysql
- much faster to load than mysqldumps

#### IMPLEMENTATION NOTES

1.*mysql* does not support remote *SELECT ... INTO OUTFILE* even with *OUTFILE=/dev/stdout*

2.*mysqldump -Tpath/to/dumpdir* has the same limitation

To workaroud these limitations a *csvdirect* approach is taken where low level mysql-python is used to perform a `select *` on selected tables and the strings obtained are written directly to the csv files of the catalog. Low-level mysql-python is used to avoid pointless conversion of strings from the underlying mysql C-api into python types and then back into strings.

**read\_desc** (*tabfile*)

Read first line of csv file containing the description

**read\_seqno** (*tab='LOCALSEQNO'*)

Read LASTUSEDSEQNO entries from table LOCALSEQNO

**rloadcat\_** (\*args, \*\*kwa)

Loads an ascii catalog into a possibly remote database. This is used by DB managers in the final step of the update SOP to propagate dybaux updates into offline\_db.

Usage:

```
./db.py tmp_offline_db rloadcat ~/dybaux/catalog/tmp_offline_db
```

Steps taken by **rloadcat**:

1. compares tables and *SEQNO* present in the ascii catalog with those in the DB and reports differences found. The comparison looks both at the *LOCALSEQNO* tables that DBI uses to hold the *LASTUSED-SEQNO* for each table and also by looking directly at all *SEQNO* present in the validity tables. The **rmecat** command does only these comparisons.
2. if updates are found the user is asked for consent to continue with updating
3. for the rows (*SEQNO*) that are added by the update the catalog validity tables *INSERTDATE* timestamps are *fastforwarded* in place to the current UTC time
4. catalog tables are imported into the DB with the *mysqlimport* tool. For payload and validity tables the *mysqlimport* option `--ignore` is used meaning that only new rows (as determined by their primary keys) are imported, other rows are ignored. For the *LOCALSEQNO* table the option `--replace` is used in order to replace the (TABLENAME, LASTUSEDSEQNO) entry.

**Returns** dictionary keyed by payload table names with values containing lists of *SEQNO* values

**Return type** dict

You might be tempted to use **rloadcat** as a faster alternative to **load** however this is not advised due to the extra things that **rloadcat** does such as update comparisons and fastforwarding and potentially merging in (when the decoupled option is used).

In comparison the **load** command blasts what comes before it, this can be done using **forced\_rloadcat** with the `--DROP` option:

```
./db.py --DROP tmp_offline_db forced_rloadcat ~/dybaux/catalog/tmp_offline_db
```

After which you can check operation via an **rdumpcat** back onto the working copy, before doing any updates:

```
./db.py tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
svn st ~/dybaux/catalog/tmp_offline_db    ## should show no changes
```

**seqno**

*SEQNO* accessor, reading and checking is done on first access to `self.seqno` with

```
db = DB()
print db.seqno    ## checks DB
print db.seqno    ## uses cached
del db._seqno     ## force a re-read and check
print db.seqno
```

**showpaytables**

list names of all DBI payload tables in DB as reported by `SHOW TABLES LIKE '%Vld'` with the 'Vld' chopped off

NB the result is cached so will become stale after deletions or creations unless `nocache=True` option is used

**showtables**

list names of all tables in DB as reported by SHOW TABLES, NB the result is cached so will become stale after deletions or creations unless *nocache=True* option is used

**tab** (*name*)

Parameters **name** – DBI payload table name

**tabfile** (*tab, catfold*)

path of table obtained from

**tables**

list of selected table names to operate on plus the mandatory LOCALSEQNO Poorly named should be *table\_selection*

**tmpdir**

Create new temporary directory for each instance, writable by ugo

**tmpfold**

Path to temporary folder, named after the DBCONF section. The base directory can be controlled by *tmpbase (-b)* option

**vdupe** (*tab*)

Currently is overreporting as needs to be balkanized by context

**vdupe\_** (*\*args, \*\*kwa*)

Report the first Vlds which feature duplicated VERSIONDATES:

```
mysql> SELECT SEQNO, VERSIONDATE, COUNT(VERSIONDATE) AS dupe FROM DemoVld GROUP BY VERSIONDATE
+-----+-----+-----+
| SEQNO | VERSIONDATE          | dupe |
+-----+-----+-----+
| 71    | 2011-08-04 05:55:47 | 2    |
| 72    | 2011-08-04 05:56:47 | 3    |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> select * from DemoVld ;
+-----+-----+-----+-----+-----+-----+
| SEQNO | TIMESTART          | TIMEEND          | SITEMASK | SIMMASK | SUBSITE | TASK |
+-----+-----+-----+-----+-----+-----+
| 70    | 2011-08-04 05:54:47 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
| 71    | 2011-08-04 06:15:46 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
| 72    | 2011-08-04 07:02:51 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
| 73    | 2011-08-04 05:54:47 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
| 74    | 2011-08-04 06:15:46 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
| 75    | 2011-08-04 05:54:47 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
| 76    | 2011-08-04 06:15:46 | 2038-01-19 03:14:07 | 127    | 1    | 0    | 0    |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

**vsssta** (*tab*)

Look at VERSIONDATE/TIMESTART/... within SSSTA groups

**wipe\_cache** ()

Wipe the cache forcing DB access to retrieve the info afresh This is needed when wish to check status after a DB load from the same process that performed the load.

## 23.2 DBAUX

### 23.2.1 DybPython.dbaux

\$Id: dbaux.py 16813 2012-04-27 08:01:24Z blyth \$

Performs actions based on working copy at various revision points.

action	notes
ls	lists commit times/messages
rcmpcat	compare ascii catalog with DB
rloadcat	load ascii catalog into DB

Usage examples:

```
./dbaux.py ls      4913
./dbaux.py ls      4913:4914
./dbaux.py ls      4913:4932
./dbaux.py ls      4913:4914 --author bv

./dbaux.py --workingcopy ~/mgr/tmp_offline_db --baseurl file:///tmp/repos/catalog ls 2:39
#
# using non default workingcopy path and baseurl
#
# NB baseurl must be the base of the repository
# TODO: avoid duplication by extracting baseurl from the working copy, or at least assert on
#

./dbaux.py rcmpcat 4913
./dbaux.py rcmpcat 4913:4932
./dbaux.py -r rcmpcat 4913

./dbaux.py rloadcat 4913
./dbaux.py --reset rloadcat 4913 ## -r/--reset deletes SVN working copy before `svn up`
```

To select non-contiguous revisions use *-a/-author* to pick just that authors commits within the revision range. Test with *ls*.

While testing in “tmp\_offline\_db” return to starting point with:

```
./db.py offline_db dump ~/offline_db.sql
./db.py tmp_offline_db load ~/offline_db.sql
```

While performing test loads into *tmp\_offline\_db*, multiple ascii catalog revisions can be loaded into DB with a single command:

```
./dbaux.py -c -r rloadcat 4913:4932
## -c/--cachesvlog improves rerun speed while testing
## -r/--reset starts from a clean revision each time,
ignoring fastforward changes done by **rloadcat**

./dbaux.py -c -r rloadcat 4913:4932
## a rerun will fail at the first revision and will do nothing
## as the DB is detected to be ahead of the catalog
```

However when performing the real definitive updates into *offline\_db* it is preferable to do things a bit differently:

```
./dbaux.py -c -r --dbconf offline_db rloadcat 4913:4932 --logpath dbaux-rloadcat-4913-4932.log
```

```
## -s/--sleep 3 seconds sleep between revisions, avoid fastforward insert times with the same
## --dbconf offline_db          target ~/.my.cnf section
```

### Checks after performing rloadcat(s)

Each **rloadcat** modifies the catalog inplace, changing the INSERTDATE times. However as are operating beneath the dybaux trunk it is not straightforward to commit these changes and record them as they are made. Instead propagate them from the database into the catalog by an **rdumpcat** following updates. This is also a further check of a sequence of **rloadcat**.

Dump the updated DB into the catalog with:

```
db.py offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db
db.py tmp_offline_db rdumpcat ~/dybaux/catalog/tmp_offline_db    ## when testing
```

Then check the status of the catalog, only expected tables .csv should be changed:

```
svn st ~/dybaux/catalog/tmp_offline_db

M      /home/blyth/dybaux/catalog/tmp_offline_db/CableMap/CableMapVld.csv
M      /home/blyth/dybaux/catalog/tmp_offline_db/HardwareID/HardwareIDVld.csv

      ## should only be INSERTDATE changes,
      ## the new times should be UTC now times spread out over the
      ## rloadcat operations

M      /home/blyth/dybaux/catalog/tmp_offline_db/tmp_offline_db.cat

      ## minor annoyance : changed order of entries in .cat
      ## ... to be fixed by standardizing order with sorted  TABLENAME
```

Following a sequence of definitive commits into *offline\_db* do an **OVERRIDE** commit into dybaux mentioning the revision range and author in the commit message. For example:

```
svn ci -m "fastforward updates following offline_db rloadcat of bv r4913:r4932 OVERRIDE" ~/dybaux/o
```

### Logfile Checks

Using the `--logpath <path>` option writes a log that is nearly the same as the console output. Checks to make on the logfile:

Check all commits are covered:

```
grep commit dbaux-rloadcat-4913-4932.log
```

Look at the *SEQNO* being loaded, verify no gaps and that the starting *SEQNO* is where expected:

```
egrep "CableMap.*new SEQNO" dbaux-rloadcat-4913-4932.log
egrep "HardwareID.*new SEQNO" dbaux-rloadcat-4913-4932.log
```

Examine fastforward times:

```
grep fastforward dbaux-rloadcat-4913-4932.log
```

## Manual Checks

Before loading a sequence of commits sample the ascii catalog at various revisions with eg:

```
svn up -r <revision> ~/dybaux/catalog/tmp_offline_db
cat ~/dybaux/catalog/tmp_offline_db/LOCALSEQNO/LOCALSEQNO.csv
```

Verify that the LASTUSEDSEQNO value changes are as expected compared to:

```
mysql> select * from LOCALSEQNO ;
+-----+-----+
| TABLENAME | LASTUSEDSEQNO |
+-----+-----+
| *          |                |
| CalibFeeSpec |                |
| CalibPmtSpec |                |
| FeeCableMap |                |
| CableMap    |                |
| HardwareID  |                |
+-----+-----+
6 rows in set (0.00 sec)
```

Expectations are:

1. incremental only ... no going back in *SEQNO*
2. no *SEQNO* gaps

The tools perform many checks and comparisons, but manual checks are advisable also, eg:

```
mysql> select distinct(INSERTDATE) from CableMapVld ;
mysql> select distinct(INSERTDATE) from HardwareIDVld
mysql> select distinct(SEQNO) from CableMap ;
mysql> select distinct(SEQNO) from CableMapVld ;
```

## reloadcat checks in various situations

Starting with r4913 and r4914 already loaded, try some operations.

1. reloadcat r4913 again:

```
./dbaux.py reloadcat 4913
...
AssertionError: ('ERROR LASTUSEDSEQNO in target exceeds that in ascii cat HardwareID ', 42, 58)
## the DB is ahead of the catalog ... hence the error
```

2. reloadcat r4914 again:

```
./dbaux.py reloadcat 4913
..
WARNING:DybPython.db:no updates (new tables or new SEQNO) are detected
## DB and catalog are level pegging ... hence "no updates" warning
```

## AVOIDED ISSUES

1. same process rcmpcat checking following an reloadcat fails as has outdated idea of DB content despite cache wiping on reloadcat. A subsequent rcmpcat in a new process succeeds. .. was avoided by creating a fresh DB instance after loads, forcing re-accessing to Database

## 23.2.2 DybPython.dbaux.Aux

**class** DybPython.dbaux.Aux (*args*)

Bases: object

**fresh\_db** ()

Pull up a new DB instance

**info**

parse/wrap output of *svn info -xml ...* caution rerun on each access

**ls** ()

Lists the revisions, author, time, commit message

**rcmpcat** ()

Loops over revisions:

1.*svn up -r* the working copy

2.runs **rcmpcat** comparing the ascii catalog with DB

**rloadcat** ()

Loops over revisions

1.*svn up -r* the working copy

2.runs **rcmpcat** to verify there are some updates to be loaded

3.invokes **rloadcat** loading ascii catalog into DB

4.runs **rcmpcat** agsin to verify load is complete

NB no confirmation is requested, thus before doing this perform an **rcmpcat** to verify expected updates

Rerunning an **rloadcat**

```
./dbaux.py rloadcat 4913    ## 1st time OK
./dbaux.py rloadcat 4913    ## 2nd time was giving conflicts ... now fails with unclean error
./dbaux.py --reset rloadcat 4913    ## blow away conflicts by deletion of working copy before
```

How to fix ?

1.When testing “svn revert” the changed validity tables throwing away the fastforward times ? via parsing “svn status”

**stat**

parse/wrap output of *svn status -xml ...* caution rerun on each access

**svnup** (*rev, reset=False, force=False*)

**Parameters**

- **rev** – revision number to bring working copy directory to
- **reset** – remove the directory first, wiping away uncommitted changes/conflicts

## 23.3 DBConf

### 23.3.1 DybPython.dbconf

When invoked as a script determines if the configuration named in the single argument exists.

Usage example:

```
python path/to/dbconf.py configname && echo configname exists || echo no configname
```

### 23.3.2 DBConf

```
class DybPython.dbconf.DBConf(sect=None, path=None, user=None, pswd=None, url=None,
                               host=None, db=None, fix=None, fixpass=None, restrict=None,
                               verbose=False, secure=False, from_env=False, nodb=False)
```

Bases: dict

Reads a section of the Database configuration file, storing key/value pairs into this dict. The default file *path* is `~/ .my.cnf` which is formatted like:

```
[testdb]
host      = dybdb1.ihep.ac.cn
database  = testdb
user      = dayabay
password  = youknowoit
```

The standard python `ConfigParser` is used, which supports `% (name) s` style replacements in other values.

Usage example:

```
from DybPython import DBConf
dbc = DBConf(sect="client", path="~/ .my.cnf" )
print dbc['host']

dbo = DBConf("offline_db")
assert dbo['host'] == "dybdb1.ihep.ac.cn"
```

**Warning:** As passwords are contained **DO NOT COMMIT** into any repository, and protect the file.

See also *Running* section of the Offline User Manual

Interpolates the DB connection parameter patterns gleaned from arguments, envvars or defaults (in that precedence order) into usable values using the context supplied by the *sect* section of the ini format config file at *path*

Optional keyword arguments:

Key-word	Description
<i>sect</i>	section in config file
<i>path</i>	colon delimited list of paths to config file
<i>user</i>	username
<i>pswd</i>	password
<i>url</i>	connection url
<i>host</i>	db host
<i>db</i>	db name
<i>fix</i>	triggers fixture loading into temporary spawned cascade and specifies paths to fixture files for each member of the cascade (semi-colon delimited)
<i>fix-pass</i>	skip the DB cascade dropping/creation that is normally done as part of cascade spawning (used in DBWriter/tests)
<i>re-strict</i>	constrain the names of DB that can connect to starting with a string, eg <code>tmp_</code> as a safeguard
<i>nodb</i>	used to connect without specifying the database this requires greater access privileges and is used to perform database dropping/creation

Correspondingly named envvars can also be used:

**DBCONF**  
**DBCONF\_PATH**  
**DBCONF\_USER**  
**DBCONF\_PSWD**  
**DBCONF\_URL**  
**DBCONF\_HOST**  
**DBCONF\_DB**  
**DBCONF\_FIX**  
**DBCONF\_FIXPASS**  
**DBCONF\_RESTRICT**

The `DBCONF` existence also triggers the `DybPython.dbconf.DBConf.Export()` in `dyb-gaudi:Database/DatabaseInterface/src/DbiCascader.cxx`

The `DBCONF_PATH` is a colon delimited list of paths that are user (~) and \$envvar OR \${envvar} expanded, some of the paths may not exist. When there are repeated settings in more than one file the last one wins.

In secure mode a single protected config file is required, the security comes with a high price in convenience

**classmethod Export** (*sect=None, \*\*extras*)

Exports the environment settings into environment of python process this is invoked by the C++ *DbiCascader* ctor

**configure\_cascade** (*sect, path*)

Interpret the *sect* argument comprised of either a single section name eg *offline\_db* or a colon delimited list of section names eg *tmp\_offline\_db:offline\_db* to provide easy cascade configuration. A single section is of course a special case of a cascade. The first(or only) section in zeroth slot is treated specially with its config parameters being propagated into *self*.

Caution any settings of *url, user, pswd, host, db* are overridden when the *sect* argument contains a colon.

**export\_** (*\*\*extras*)

Exports the interpolated configuration into corresponding *DBI* envvars :

`ENV_TSQL_USER`  
`ENV_TSQL_PSWD`  
`ENV_TSQL_URL`  
`ENV_TSQL_FIX` (added to allow `DBConf` to survive thru the env-glass )

And *DatabaseSvc* envvars for access to non-DBI tables via *DatabaseSvc* :

`DYB_DB_USER`  
`DYB_DB_PSWD`  
`DYB_DB_URL`

**classmethod from\_env** ()

Construct `DBConf` objects from environment :

`ENV_TSQL_URL`  
`ENV_TSQL_USER`

```
ENV_TSQL_PSWD
```

```
ENV_TSQL_FIX
```

**classmethod** `has_config` (*name\_=None*)

Returns if the named config is available in any of the available DBCONF files

For cascade configs (which comprise a colon delimited list of section names) all the config sections must be present.

As this module exposes this in its main, config sections can be tested on command line with:

```
./dbconf.py offline_db && echo y || echo n
./dbconf.py offline_dbx && echo y || echo n
./dbconf.py tmp_offline_db:offline_db && echo y || echo n
./dbconf.py tmp_offline_dbx:offline_db && echo y || echo n
```

**mysqldb\_parameters** (*nodb=False*)

Using the *nodb=True* option skips database name parameter, this is useful when creating or dropping a database

**classmethod** `prime_parser` ()

Prime parser with “today” to allow expansion of `% (today) s` in `~/ .my .cnf` allowing connection to a daily recovered database named after todays date

**classmethod** `read_cfg` (*path=None*)

Classmethod to read config file(s) as specified by *path* argument or `DBCONEF_PATH` using `ConfigParser`

## 23.4 DBCas

### 23.4.1 DybPython.dbcas

Pythonic representation of a DBI cascade, see *A Cascade of Databases* , than implements spawning of the cascade. Creating a pristine cascade that can be populated via fixtures.

**Advantages :**

- allows testing to be performed in fully controlled/repeatable DB cascade
- prevents littering production DB with testing detritus

Note such manipulations are not possible with the C++ *DbiCascader DbiConnection* as these fail to be instantiated if the DB does not exist.

**class** `DybPython.dbcas.DBCas` (*cnf, append=True*)

Bases: `list`

Represents a cascade of databases (a list of `DBCon` instances) created from a `DybPython.dbconf.DBConf` instance

**spawn** ()

Spawning a cascade creates the databases in the cascade with prefixed names and populates them with fixtures

**class** `DybPython.dbcas.DBCon` (*url, user, pswd, \*\*kwa*)

Bases: `dict`

Dictionary holding parameters to connect to a DB and provides functionality to drop/create databases and run updates/queries against them.

**process** (*sql*)

Attempts to create prepared statement from sql then processes it

**server**

If the connection attempt fails, try again without specifying the DB name, see `root:TMysqlServer`

---

**Todo**

Find way to avoid/capture the error after failure to connect

---

**spawn** (*fixpass=False*)

Create new DB with prefixed name and spawn a DBCon to talk to it with

When *fixpass* is True the DB is neither created or dropped, but it is assumed to exist. This is used when doing DBI double dipping, used for example in `dybgaudi:Database/DBWriter/tests`

**class** `DybPython.dbcas.DD`

Bases: `dict`

Compares directories contained cascade mysqldumps after first replacing the times from todays dates avoiding inevitable validity insert time differences

Successful comparison Requires the *DbiTest* and *DybDbiTest* dumps to be created on the same UTC day.

**get\_prep** ()

Initially this just obscured the times in UTC todays date (which appears in the Vld table INSERTDATE column) to allow comparison between *DbiTest* and *DybDbiTest* runs done on the same UTC day

However, now that are extending usage of the MYSQLDUMP reference comparisons to dumps of DB-Writer created DB from different days, need to obscure todays date fully

**prep**

Initially this just obscured the times in UTC todays date (which appears in the Vld table INSERTDATE column) to allow comparison between *DbiTest* and *DybDbiTest* runs done on the same UTC day

However, now that are extending usage of the MYSQLDUMP reference comparisons to dumps of DB-Writer created DB from different days, need to obscure todays date fully

## 23.5 dbsvn - DBI SVN Gatekeeper

### 23.5.1 `DybPython.dbsvn`

#### Usage examples

```
./dbsvn.py --help
    ## full list of options and this help text

./dbsvn.py ~/catdir -M
    ## check catalog and skip commit message test

./dbsvn.py ~/catdir -m "test commit message dybsvn:source:dybgaudi/trunk/CalibWritingPkg/DBUPDATE.t
    ## check catalog and commit message
```

This script performs basic validations of SVN commits intended to lead to DB updates, it is used in two situations:

1. On the SVN server as part of the pre-commit hook that allows/denies the commit
2. On the client, to allow testing of an intended commit before actually attempting the commit as shown above

NB this script DOES NOT perform commits, it only verifies them

### How this script fits into the workflow

```

cd ; svn co http://dayabay.ihep.ac.cn/svn/dybaux/catalog/tmp_offline_db
    ## check out catalog containing the subset of manually updated tables

cd ; svn co http://dayabay.phys.ntu.edu.tw/repos/newtest/catalog/tmp_offline_db/
    ## test catalog at NTU

./db.py offline_db rdumpcat ~/tmp_offline_db
    ## rdumpcat current offline_db on top of the SVN checkout and look for diffs

svn diff ~/tmp_offline_db
    ## COMPLAIN LOUDLY IF YOU SEE DIFFS HERE BEFORE YOU MAKE ANY UPDATES

./db.py tmp_joe_offline_db rdumpcat ~/tmp_offline_db      ## NB name switch
    ## write DBI catalog on top of working copy ~/tmp_offline_db

svn diff ~/tmp_offline_db
    ## see if changed files are as you expect

./dbsvn.py ~/tmp_offline_db
    ## use this script to check the "svn diff" to see if looks like a valid DBI update

./dbsvn.py ~/tmp_offline_db -m "Updating dybsvn:source:dybgaudi/trunk/CalibWritingPkg/DBUPDATE.txt@12000"
    ## fails as annotation link refers to dummy path, no such package and no change to that file at that revision

./dbsvn.py ~/tmp_offline_db -m "Annotation link dybsvn:source:dybgaudi/trunk/Database/DybDbiTest/test.txt@12000"
    ## check the "svn diff" and intended commit message, fails as no revision

./dbsvn.py ~/tmp_offline_db -m "Annotation link dybsvn:source:dybgaudi/trunk/Database/DybDbiTest/test.txt@12000"
    ## fails as no change to that file at that revision

./dbsvn.py ~/tmp_offline_db -m "Annotation link dybsvn:source:dybgaudi/trunk/Database/DybDbiTest/test.txt@12000"
    ## succeeds

svn ci ~/tmp/offline_db -m "Updating dybsvn:source:dybgaudi/trunk/CalibWritingPkg/DBUPDATE.txt@12000"
    ## attempt the actual commit

```

### What is validated by `dbsvn.py`

1. The commit message, eg “Updating dybsvn:source:dybgaudi/trunk/CalibWritingPkg/DBUPDATE.txt@12000”
  - (a) must provide valid dybsvn reference which includes dybgaudi/trunk package path and revision number
2. Which files (which represent tables) are changed
  - (a) author must have permission for these files/tables
  - (b) change must effect DBI file/tablepairs (payload, validity)
3. What changes are made:
  - (a) must be additions/subtractions only (allowing subtractions is for reverts)
  - (b) note that LOCALSEQNO (a DBI bookkeeping table) is a special case

## Rationale behind these validations

1. valid DBI updates
2. establish provenance and purpose
  - (a) what purpose for the update
  - (b) where it comes from (which revision of which code was used)
  - (c) precise link to producing code and documentation

## Commit denial

This script is invoked on the SVN server by the pre-commit hook (shown below) if any directories changed by the commit start with “catalog”. If this script exits normally with zero return code, the commit is allowed to proceed.

On the other hand, if this script returns a non-zero exit code, for example if an assert is tickled, then the commit is denied and stderr is returned to the failed committer.

## OVERRIDE commits

Administrators (configured using -X option on the server) can use the string “OVERRIDE” in commit messages to short circuit validation. This is needed for non-standard operations, currently:

1. adding/removing tables

A commit like the below from inside catalog will fail, assuming that the **dayabay** svn identity is not on the admin list:

```
svn --username dayabay ci -m "can dayabay use newestest OVERRIDE "
```

## Deployment of pre-commit hook on SVN server

Only SVN repository administrators need to understand this section.

The below commands are an example of creating a bash pre-commit wrapper. After changing the TARGET and apache user identity, the commands can be used to prepare the hook. Note that the pre-commit script is invoked by the server in a bare environment, so any customizations must be propagated in, deploy the code:

```
## on SVN server node
cd
svn co http://dayabay.ihep.ac.cn/svn/dybsvn/dybgaudi/trunk/DybPython/python/DybPython ## into $HOME
svn up ~/DybPython

export TARGET=/var/scm/repos/dybaux/hooks/pre-commit          ## dybaux hooks
export APACHE_USER=nobody.nobody                            ## customize to your server
sudo bash -c "cp $HOME/DybPython/{dbsvn,svndiff,dbvld}.py $(dirname $TARGET)/ && chown $APACHE_USER

DBSVN_XREF=/var/scm/svn/dybsvn python $HOME/DybPython/dbsvn.py HOOK ## check the hook is customized
DBSVN_XREF=/var/scm/svn/dybsvn python $HOME/DybPython/dbsvn.py HOOK | sudo bash -c "cat - > $TARGET
cat $TARGET
```

1. DBSVN\_XREF points to the dybsvn SVN repository, which is used to validate cross referencing links from dybaux to dybsvn
2. user nobody corresponds to the user which the SVN webservice process runs as
3. note that the dbsvn.py option -c/--refcreds is not used for dybaux as local access to dybsvn repository is used (with svnlook)

## Hook Deployment on server remote from dybsvn

The test deployed hook at NTU gets cross-referencing to dybsvn via `svn log` etc whereas, the real dybaux hook accesses dybsvn locally on the server using `svnlook`. Due to this different options are needed in hook deployment, specially as are using the default `DBSVN_XREF` of `http://dayabay.ihep.ac.cn/svn/dybsvn` need to enter `DBSVN_XREF_PASS`:

```
## on SVN server node
cd
svn co http://dayabay.ihep.ac.cn/svn/dybsvn/dybgaudi/trunk/DybPython/python/DybPython ## into $HOME
svn up ~/DybPython

export TARGET=/var/scm/repos/newtest/hooks/pre-commit ; export APACHE_USER=nobody.nobody
sudo bash -c "cp $HOME/DybPython/{dbsvn,svndiff,dbvld}.py $(dirname $TARGET)/ && chown $APACHE_USER

DBSVN_XREF_PASS=youknowit python $HOME/DybPython/dbsvn.py HOOK ## check the hook is customized as de
DBSVN_XREF_PASS=youknowit python $HOME/DybPython/dbsvn.py HOOK | sudo bash -c "cat - > $TARGET && ch
cat $TARGET
```

## Typical Problems with the Hook

### Mainly for admins

If the precommit hook is mis-configured the likely result is that attempts to commit will hang. For example the `dbsvn.py` invocation in the hook script needs to have:

1. a valid admin user (SVN identity)
2. local filesystem repository path for the cross reference `-r` option

The default cross reference path is the dybsvn URL which might hang on the server as the user(`root/nobody/...`) that runs the SVN repository normally does not have user permissions to access sibling repository dybsvn. (have switched to non-interactive now)

A pre-commit hook testing harness is available in bash functions `env:trunk/svn/svnprecommit.bash`

## Trac Config to limit large diff hangs

### Only for admins

The large diffs representing DB updates that are stored in dybaux can cause Trac/apache to hang on attempting to browse them in Trac. To avoid this the default `max_diff_bytes` needs to be reduced, do this for dybaux with:

```
env- ## env precursor
trac-
TRAC_INSTANCE=dybaux trac-edit
```

Modify down to 100000:

```
[changeset]
max_diff_bytes = 100000 # 10000000
max_diff_files = 0
```

## 23.5.2 DBIValidate

```
class DybPython.dbsvn.DBIValidate(diff, msg, author, opts)
    Bases: list
```

Basic validation of commit that represents an intended DB update

**dump\_diff** ()

Traverse the parsed diff hierarchy diff/delta/block/hunk to extract the validity diffs such as:

```
+30, "2010-09-22 12:26:59", "2038-01-19 03:14:07", 127, 3, 0, 1, -1, "2010-09-22 12:26:59", "2011-05-
```

deltas should have a single block for a valid update

**validate\_hunk** (*hunk*)

Check the Vld table diff validity entries have valid times and conform to overlay versioning compliance.

Turns out not to be possible to check for overlay versioning compliance from a delta as in the case of updates with changed timestart the offset from the first timestart gets used, see #868

**validate\_update** ()

Current checks do not verify tail addition

**validate\_validity** ()

Checks on the validity contextrange of updates, to verify:

1. Presence of valid dates in all four DBI date slots
2. Overlay versioning compliance, namely appropriate correspondence between TIMESTART and VERSIONDATE

## 23.6 DybDbiPre

### 23.6.1 Tab

**class** DybDbiPre.**Tab**

Bases: list

DybDbiPre.Tab instances are created by the parsing of .spec files (dybgaudi:Database/DybDbi/spec). Instances contain a list of dicts corresponding to each payload row in the DBI table together with a metadata dictionary for class level information.

To test the parsing of a .spec file, use for example:

```
cat $DYBDBIROOT/spec/GSimPmtSpec.spec | python $DYBDBIPREERoot/python/DybDbiPre/__init__.py
```

The instances are available in the django context used to fill templates dybgaudi:Database/DybDbi/templates used in the generation of:

1. DbTableRow subclasses allowing DBI to interact with the table
2. Documentation presenting the DBI tables in .tex and wiki formats
3. SQL scripts for table creation .sql

The meanings of the quantities in the .spec are ultimately determined by their usage in the templates, however some guideline definitions are listed below:

#### row level quantities

**name** column name as used in C++ getter and setter methods

**dbtype** MySQL column type name used in table description, such as *double* or *int(10) unsigned*

**codetype** type used in generated C++ code, eg *DayaBay::FeeChannelId*

**legacy** name of the column in database table

**description** short definition of the meaning of the column

**code2db** C++ converter function used to translate a value in code into a value stored in the DB, eg *.fullPacked-Data()*

**memb** name of the column data member in the C++ table row class, **WARNING, CURRENTLY NOT IN USE**

### class/table level properties

**meta** a token that identifies the key, value pairs on the line as metadata rather than a table row

**table** name of the payload Database table, eg *CalibFeeSpec*

**class** name of the generated DbiParam class, follow convention of naming with a **G** prefix eg *GCalibFeeSpec*

**CanL2Cache** set to *kFALSE*, L2 caching is for debugging only

**legacy** name of prior table when migrations are performed, **WARNING, CURRENTLY NOT IN USE**, set to table name

**rctx** default read context represented by a comma delimited string, see *dyb-gaudi:Database/DybDbi/src/DbiCtx.cxx*

**wctx** default write context range represented by a comma delimited string, see *dyb-gaudi:Database/DybDbi/src/DbiCtx.cxx*

### usage in templates

The class level and row level quantities are used in django templates with expressions of the form:

```
{{ t.meta.table }}
```

```
{% for r in t %}'{{ r.name }}' {{ r.dbtype }} default NULL COMMENT '{{ r.description }}',
```

**\_\_call\_\_** (*d*)

If fields in the *.spec* file include a “meta” key then the fieldname(ie key),value pairs are included into the meta dictionary

## 23.7 DybDbi

Making DBI easy to use from python:

### 23.7.1 DybDbi

DybDbi python package provides access to most DbiParam functionality, with generation of classes based on *.spec* files and wrapping of the python classes for easier usage, enabling access to model objects via:

```
from DybDbi import GCalibPmtSpec
from DybDbi import *
```

Example of introspecting the specification:

```

sk = GCalibPmtSpec.SpecKeys().aslist()    # list of row names
sk
    ['PmtId',
     'Describ',
     'Status',
     ...

sl = GCalibPmtSpec.SpecList().aslod()    # list of row maps ... list-of-dict
sl
    [{'code2db': '',
     'codetype': 'int',
     'dbtype': 'int(11)',
     'description': '',
     'legacy': 'PMTID',
     'memb': 'm_pmtId',
     'name': 'PmtId'},
     ...

sm = GCalibPmtSpec.SpecMap().asdod()    # map of row maps, keyed by name    dict-of-dict
sm
    {'AfterPulseProb': {'code2db': '',
                       'codetype': 'double',
                       'dbtype': 'float',
                       'description': 'Probability of afterpulsing',
                       'legacy': 'PMTAFTERPULSE',
                       'memb': 'm_afterPulseProb',
                       'name': 'AfterPulseProb'},
     ...

sm['TimeOffset']['description']    # access any aspect of spec "matrix" directly by name
    'Relative transit time offset'

sk = cls.SpecKeys().aslist()
sm = cls.SpecMap().asdod()
for k in sk:
    print sm[k]

```

### 23.7.2 DybDbi.Wrap

Wrapping is the principal technique used by *DybDbi* to provide simple pythonic usage of the underlying C++ classes (actually PyROOT proxies).

```

class DybDbi.Wrap(kls, attfn={})
    Bases: object

```

Control center for application of generic class manipulations based on the names of methods in contained kls. The manipulations do not require the classes to be imported into this scope.

Wrapping is applied to:

- all *genDbi* generated *DbiTableRow* subclasses and corresponding templated *DbiRpt* and *DbiWrt* (readers and writers)
- a selection of other *Dbi* classes that are useful interactively

```

define__repr__ ()
    Assign default repr ... override later if desired

```

**define\_create()**

Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**define\_csv()**

Provide csv manipulations as classmethods on the Row classes

**define\_listlike()**

Application of function RPT to DbIRpt<T> classes provides instances of that class with a list-like interface supporting access by index and slice, indices can be negative to provide access to the end.:

```
r[0]      first
r[-1]     last

r[0:9]    first 10
r[-3:]    last 3

r[0:2000:500]
r[-10:-1:2]    2-step thru the last 10

for x in r[2540:]:
    print x

for x in r[-10:]:
    print x
```

THOUGHTS : \* no need for generator implementation for large result set as already all in memory anyhow

**define\_properties()**

Define properties corresponding to Get\* and Set\* methods in the contained kls, providing attribute style access and setting

```
g = i.x
i.x = s
```

NB “getters” which take arguments GetWithArg(Int\_t naughty) have to be skipped via:

```
cls.__skip__ = ("WithArg",)
```

**define\_update()**

Provide dict like updating for DbTableRow subclasses, eg:

```
from DybDbi import GCalibPmtSpec
r = GCalibPmtSpec.Rpt()
z = r[0]
print z.asdict
print z.keys
z.update( Status=10 )
```

**get\_attfn(m)**

Returns function than when applied to an object returns (m,obj.Get<m>()) where m is the attribute name

**make\_\_repr\_\_()**

Provide a default `__repr__` function that presents the attribute names and values as a dict

### 23.7.3 DybDbi.CSV

**class** DybDbi.CSV(*path*, **\*\*kwargs**)

Bases: list

Reader/writer for .csv files. The contents are stored as a list of dicts.

#### Parameters

- **delimiter** – csv field divider
- **prefix** – string start of comment lines to be ignored, default #Table
- **descmarker** – strings used to identify the field description line
- **synth** – when defined, add extra field with this name to hold the csv source line number
- **fields** – impose fieldnames externally, useful for handling broken csv which cannot be fixed immediately

Read usage example:

```
src = CSV("$DBWRITERROOT/share/DYB_MC_AD1.txt", delimiter="\t" )
src.read()
for d in src:
    print d

len(src)
src[0]
src[-1]
src.fieldnames
```

On reading an invalid CSV an exception, with error report, is raised:

```
src = CSV("$DBWRITERROOT/share/DYB_SAB_AD1.txt", delimiter="\t" )
src.read()
```

Handling of common csv incorrectnesses is made:

- 1.description line fixed up to conform to the delimiter
- 2.description line extraneous characters removed (other than fieldnames and delimiters)
- 3.removes comments

Write usage example, field names are obtained from the dict keys:

```
out = CSV("/tmp/demo.csv", delimiter="\t" )
for d in list_of_dict_datasource:
    out.append(d)
out.write()
```

#### fieldnames

If fieldnames keyword argument is supplied return that otherwise return the names of the keys in the first contained dict. In order to control the order of fields, the argument has to be specified.

**write()**

### 23.7.4 DybDbi.Source

**class** DybDbi.Source(*f*, **delimiter**='t', **prefix**='#Table', **descmarker**='#[]', **synth**='srcline', **fields**=[ ])

Bases: list

Behaves like a file and holds the original text of the CSV. Applies some fixes to make readable as CSV:

1. removes lines that begin with the `prefix` argument, default `#Table`
2. determines the description line by looking for all characters of the `descmarker` argument, default `# []`
3. normalize the description line to conform to the delimiter

Normally used internally via CSV, but can be useful to debug broken .csv files interactively:

```
In [2]: import os
```

```
In [3]: from DybDbi import Source
```

```
In [4]: src=Source(open(os.path.expandvars("$DATASVCROOT/share/feeCableMap_MDC09a.txt")), de
```

```
In [5]: for _ in src:print _    ## have to interate to populate
```

```
In [7]: src
```

```
Out[7]:
```

```
Source stat: {'descline': 1, 'prefix': 0, 'total': 1585, 'payload': 1584}
```

```
cols: ['srcline', 'ChannelID', 'Description', 'ElecHardwareId', 'Description', 'SensorID', '
```

```
In [8]: src.descmarker
```

```
Out[8]: '#'
```

```
In [9]: src.descline( src[0] )    ## debugging the field extraction from the description line
```

```
Out[9]: ' ChannelID Description ElecHardwareId Description SensorID Description SensorHardwa
```

Note the severely invalid .csv (4 fields with the same name) workaround until the .csv can be fixed is to externally impose the fields:

```
fields = 'ChannelID Description0 ElecHardwareId Description1 SensorID Description2 SensorHardwar
src=Source(open(os.path.expandvars("$DATASVCROOT/share/feeCableMap_MDC09a.txt")), delimiter=" ",
```

### Parameters

- **delimiter** – csv field divider
- **prefix** – string start of lines to be ignored
- **descmarker** – strings used to identify the field description line
- **synth** – when defined, add extra field with this name to hold the csv source line number
- **fields** – when defined overrides the content of the descline

### **clean** (*line*)

shrink multiple spaces to a single space, and strip head and tail whitespace

### **descline** (*line*)

Remove the descmarker characters from the description line,

### **is\_descline** (*line*)

Checks if line contains all of the description markers

### **next** ()

On iterating though this “synthetic” file fixes and additions are made to render the “physicist-csv” as real csv

### 23.7.5 DybDbi.Mapper

**class** DybDbi.Mapper(*cls, csv\_fields, \*\*kwargs*)

Bases: dict

Establish the mapping between sets of fields (such as csv fields) and dbi attributes, usage:

```
ckf = ['status', '_srcline', 'afterPulse', 'sigmaSpe', 'pmtID', 'efficiency', 'darkRate', '_hasb  
mpr = Mapper( GCalibPmtSpec, ckf , afterPulse="AfterPulseProb", sigmaSpe="SigmaSpeHigh", prePuls  
print mpr
```

If a mapping cannot be made, an exception is thrown that reports the partial mapping constructed.

The automapping performed is dumb by design, only case insensitively identical names are auto mapped. Other differences between csv field names and dbi attributes must be manually provided in the keyword arguments.

The string codetype from the spec is promoted into the corresponding python type, to enable conversion of the csv dict (comprised of all strings) into a dbi dict with appropriate types for the values.

**automap** ()

Basic auto mapping, using case insensitive comparison and yielding case sensitive mapping from csv fields to dbi attributes

The index of the csv fieldname in the dbi attribute list is found with case insensitive string comparison

**check\_kv** (*kvl, expect, name*)

Check the keys/values are in the expected list

**convert\_csv2dbi** (*dcsv*)

Translate dict with csv fieldnames into dict with dbi attr names and appropriate types for insertion into the DBI Row cls instance

### 23.7.6 DybDbi.Ctx

**class** DybDbi.Ctx

Bases: ROOT.ObjectProxy

**AsString**

char\* Ctx::AsString(int ctx)

**FromIndex**

int Ctx::FromIndex(int i)

**FromString**

int Ctx::FromString(char\* str)

**FullMask**

int Ctx::FullMask()

**Length**

int Ctx::Length()

**MaskFromString**

int Ctx::MaskFromString(char\* str)

**MaxBits**

int Ctx::MaxBits()

**StringForIndex**

char\* Ctx::StringForIndex(int i)

**StringFromMask**

```
char* Ctx::StringFromMask(int mask)
```

**23.7.7 DybDbi.DbiCtx**

`DbiCtx` is a C++ class designed to facilitate DBI usage from python the `DbiRpt` and `DbiWrt` classes each contain an instance of `DbiCtx`

The `DbiCtx` instances have constituents corresponding to all possible arguments of all DBI reader and writer constructors:

1. readers `DbiResultPtr<T>`, see [databaseinterface:DbiResultPtr.h](#)
2. writers `DbiWriter<T>` see [databaseinterface:DbiWriter.h](#)

The precise constructor used is determined by the attribute settings made in the `DbiCtx` instance The attributes are divided into tables below according to recommended usage

**When reading from DB**

Attribute	type	notes
context	<code>DybDbi.Context</code>	composite setting
timestamp	<code>DybDbi.TimeStamp</code>	constituent of context
simflag		constituent of context
site		constituent of context
detectorid		constituent of context
subsite		
task		
dbno		
logcomment		
tablename		<i>Expert Usage Only</i> leave at default
aborttest		<i>Expert Usage Only</i>
findfulltimewindow		<i>Expert Usage Only</i>
sqlcontext		<i>sqlcontext</i> eg wideopen 1=1
datasql		<i>datasql</i>

**sqlcontext**

Replaces the validity context with the provided SQL `where` clause applied to the Validity table, for example the wideopen 1=1 caution this can be very memory expensive.

**datasql**

Applies the provided SQL `where` clause to the payload table

**Expert Usage Only**

Familiarity with the DBI implementation is required.

Use when writing to the DB

Attribute	type	notes
contextrange	DybDbi.ContextRange	constituent of contextrange constituent of contextrange constituent of contextrange constituent of contextrange leave as default -1
timestart	DybDbi.TimeStamp	
timeend	DybDbi.TimeStamp	
sitemask		
simmask		
aggn0		
task		
dbno		
logcomment		

Use with caution when reading from DB:

Attribute	notes
datasql	a payload where clause
tablename	genDbi default usually ok
versiondate	

Usage only advised for experts familiar with DbWriter<T> and DbResultPtr<T> ctors

Attribute	notes
seqno	
validityrec	
datafillopts	
dbname	

### 23.7.8 DybDbi.vld.versiondate

In *-transfix* mode copies all DBI tables from *tmp\_offline\_db* into *fix\_offline\_db* with *VERSIONDATE* changed to timestart floored scheme.

As yet no collision checks.

Try to back-predict versiondate for all validities in all Vld tables

Usage:

```
./versiondate.py --help
./versiondate.py
./versiondate.py CalibPmtHighGain
./versiondate.py --transfix
./versiondate.py --transfix -l DEBUG
./versiondate.py --transfix HardwareID -l DEBUG
./versiondate.py --transfix CableMap -l DEBUG
./versiondate.py --transfix CalibPmtSpec -l DEBUG
```

To confirm no change when timestart flooring is off:

```
echo select \* from tmp_offline_db.HardwareIDVld | mysql -t > tmp_HardwareID.txt
echo select \* from fix_offline_db.HardwareIDVld | mysql -t > fix_HardwareID.txt
diff tmp_HardwareID.txt fix_HardwareID.txt
```

**class** DybDbi.vld.versiondate.VFS (*field, value*)

Bases: list

Convenience class to format all column names in validity table and allow easy swap out of single fields

DybDbi.vld.versiondate.**check\_versiondate**(args)

For all DBI tables in source *tmp* DB invoke the *check\_versiondate\_tab*

DybDbi.vld.versiondate.**check\_versiondate\_tab**(tab)

Check if the versiondate of all validities matches that divined with QueryOverlayVersionDate, via SEQNO condition to allow backdating

Probably this is just confirming that overlay versioning was used

DybDbi.vld.versiondate.**setup**()

1.Establish coordinates of source *tmp* and target *fix* databases with safety asserts

2.drop and recreate the target DB *fix*

3.label DB instances with *dbno* for DBI usage

DybDbi.vld.versiondate.**transfix**(args)

Copies all DBI tables from *tmp* into *fix* changing the *VERSIONDATE* to conform to *TimeStartFlooredVersionDate* scheme.

Hmm currently the *VERSIONDATE* collision avoidance implemented in the DBI writer does not come into play here : **IT NEEDS TO BE SPOOFED HERE**

DybDbi.vld.versiondate.**transfix\_tab**(tmp,fix,tab,localseqno=False)

SEQNO by SEQNO transfers table entries from *tmp* to *fix* databases effectively replaying table history as it grows in the *fix* DB. Allowing changes to the *VERSIONDATE* scheme to be tested.

#### Parameters

- **tmp** – source DybPython.DB instance
- **fix** – target DybPython.DB instance
- **tab** – payload table name

Actions:

1.create payload and validity tables using DBI, note there is no need to drop the tables first as started by dropping the *fix* DB

2.loops over validity entries of *tab* in *tmp* DB in *SEQNO asc* order

For each *vld* entry calls *qovd\_transfix* prior to transferring the *vld* from the *tmp* to *fix* DB allowing the *VERSIONDATE* to be modified to use a different scheme.

The backdated *qovd* comparison with *versiondate* could be used to detect early entries that did not use overlay versioning:

```
qovd.GetSec() != versiondate.GetSec()
```

Note, formerly used application of an SQL condition to effect a *timstart* floor, inside:

```
qovd = qovd_transfix( kls, vrec, fix.dbno ).AsString("s")
```

But following C++ additions can now directly use *QueryOverlayVersionDate* and set the additional parameter *fTimeStartFlooredVersionDate*

### 23.7.9 DybDbi.vld.vlut

Compare validity lookup tables between different DB and option variations

Due to usage of *converter.tabular.TabularData* for presentation of the tables this must be run from the *docs* virtual python:

```
~/rst/bin/python vlut.py
~/v/docs/bin/python vlut.py
```

## DEFICIENCIES

1. top level `index.rst` of tables requires manual editing
2. running single context, has habit of messing up all context indices. Workaround is rerunning the `--ctx ALL`

## Production Run From Scratch

Clean start:

```
dybdbi
cd vld
./versiondate.py          ## transfixion of tmp_offline_db creating fix_offline_db
rm -rf /tmp/blyth/dbiscan ## clean start
time ~/rst/bin/python ./vlut.py
```

Full traverse takes:

```
real    103m41.969s
user    92m13.993s
sys     4m54.889s
```

## Payload Digest Based Comparisons

Goal is to compare very different DB for content:

1. `tmp_offline_db` with `CableMap,HardwareID` duplications eliminated and written with the `G*Fix` classes
2. `tmp_copy_db` copy of `offline_db`

Features:

1. payload digests should allow comparison without `SEQNO` correspondence
2. `TIMESTARTs` should align

Issues:

1. `insertdates`(table update history) does not align forcing to manually
  - (a) `opts['_insertdate_aligned'] = False`
  - (b) common junctures might also be defined to allow more than just the last comparison

## Presenting the LUTs with Sphinx

### Debugging

Unexplained difference between `tmp_offline_db` without extra ordering (formerly though to be implicit `VERSIONDATE desc, SEQNO asc`) and `SEQNO asc`

```
[blyth@belle7 agno-1_simflag2_site1_subsite2_task0]$ diff ./tmp_offline_db/vlut.rst ./orderingSEQNOasc_tmp_offline_db/vlut.rst
```

Promising no diffs between `tmp_offline_db` with added “SEQNO asc” validity ordering and the `fix_offline_db` (for the fixed the SEQNO diddling makes no difference as all same ... at least in ctxs covered so far). This means that enforcing extra SEQNO asc validity ordering on an horrible degenerate mess of duplicated VERSIONDATEs in `tmp_offline_db` succeeds to give the same VLUT as the transfixed one ... with the careful timestart floored version date with no degeneracy.

This is understandable as the validity ordering becomes “VERSIONDATE desc, SEQNO asc” so the larger SEQNO of degenerate sets wins. That seems like it should be mostly correct ... are there any edge cases ?

If this pans out to all ctxs then **the eagle has landed**

## Ngix Hookup

Hook up to nginx:

```
nginx-
cd `nginx-htdocs`
sudo ln -s $SITEROOT/./users/$USER/dbiscan/sphinx/_build/dirhtml dbiscan
```

Add location with `nginx-edit`:

```
location /dbiscan {
    autoindex on ;
    autoindex_exact_size off ;
    autoindex_localtime on ;
}
```

After restart of nginx can peruse the tables:

- <http://belle7.nuu.edu.tw/dbiscan/>

## Machinery Issues

1. rerun bug, not updating index have to manually: `rm /tmp/blyth/dbidigest/sphinx/CableMap/index.rst`

`DybDbi.vld.vlut.traverse_vlut` (vs)

For all tables common to the databases traverse all contexts in common, writing the vlut rst files and making comparisons.

Loads persisted scans created by `DybPython.vlut.Scan` and presents as LUTs (look up tables) in rst table format

**Parameters** vs – *VlutSpec* instance

### 23.7.10 DybDbi.vld.vsmry

Usage:

```
time ~/rst/bin/python vsmry.py $SITEROOT/./users/$USER/dbiscan/sphinx/CableMap/smry.pc
```

Smry are dicts of stat dicts, keyed by the VLUT relative path, eg:

```
CableMap/aggno-1_simflag2_site4_subsite5_task0/tmp_offline_db_cf_fix_offline_db/vlutorderingSEQNOdes
CableMap/aggno-1_simflag2_site4_subsite6_task0/tmp_offline_db_cf_fix_offline_db/vlut.rst {'ndif': 14
CableMap/aggno-1_simflag2_site4_subsite6_task0/tmp_offline_db_cf_fix_offline_db/vlutorderingSEQNOasc
CableMap/aggno-1_simflag2_site4_subsite6_task0/tmp_offline_db_cf_fix_offline_db/vlutorderingSEQNOdes
CableMap/aggno-1_simflag2_site4_subsite7_task0/tmp_offline_db_cf_fix_offline_db/vlut.rst {'ndif': 0}
```

CableMap/aggno-1\_simflag2\_site4\_subsite7\_task0/tmp\_offline\_db\_cf\_fix\_offline\_db/vlutorderingSEQNOasc  
 CableMap/aggno-1\_simflag2\_site4\_subsite7\_task0/tmp\_offline\_db\_cf\_fix\_offline\_db/vlutorderingSEQNOdesc

Down to handful of 3 pathological ctxs in 3 tables (Demo doesnt count) for whom the fix is not doing the biz:

```
CRITICAL:__main__:checking smry beneath /tmp/blyth/dbiscan/sphinx levels ['ctx']

CRITICAL:__main__:CableMap          aggno-1_simflag2_site32_subsite1_task0  tmp_offline_db_c
CRITICAL:__main__: http://belle7.nuu.edu.tw/dbiscan/CableMap/aggno-1_simflag2_site32_subsite1_ta

CRITICAL:__main__:CalibFeeSpec      aggno-1_simflag1_site32_subsite1_task0  tmp_offline_db_c
CRITICAL:__main__: http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1_simflag1_site32_subsite:
                http://belle7.nuu.edu.tw/dbiscan/CalibFeeSpec/aggno-1_simflag1_site32_subsite:

    ## this CalibFeeSpec ctx is the only CalibFeeSpec ctx (suggesting junk?)

CRITICAL:__main__:CalibPmtSpec      aggno-1_simflag1_site32_subsite2_task0  tmp_offline_db_c
CRITICAL:__main__: http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1_simflag1_site32_subsite:
                http://belle7.nuu.edu.tw/dbiscan/CalibPmtSpec/aggno-1_simflag1_site32_subsite:

    ## no order flip discrep within tmp_ and fix_ ... but there is between em ?

CRITICAL:__main__:Demo              aggno-1_simflag1_site127_subsite0_task0  tmp_offline_db_c
CRITICAL:__main__: http://belle7.nuu.edu.tw/dbiscan/Demo/aggno-1_simflag1_site127_subsite0_task0,
```

```
select * from CableMapVld where SIMMASK=2 and SITEMASK=32 and TASK=0 ;
select * from CalibFeeSpecVld where SIMMASK=1 and SITEMASK=32 and TASK=0 ;
select * from CalibPmtSpecVld where SIMMASK=1 and SITEMASK=32 and TASK=0 ;
```

**DEBUG/INFO/WARN/ERROR/FATAL**

DybDbi.vld.vsmry.**ctx\_count** (*dbconfs*)

To verify are seeing the appropriate number of distinct ctxs:

```
mysql> select distinct(CONCAT(SITEMASK,":",SIMMASK,":",SUBSITE,":",TASK,":",AGGREGATENO)) from C
+-----+
| (CONCAT(SITEMASK,":",SIMMASK,":",SUBSITE,":",TASK,":",AGGREGATENO)) |
+-----+
| 32:1:1:0:-1 |
| 32:1:2:0:-1 |
| 127:1:0:0:-1 |
| 127:2:0:0:-1 |
| 1:2:1:0:-1 |
| 1:1:5:0:-1 |
| 1:1:6:0:-1 |
| 1:1:1:0:-1 |
| 1:1:2:0:-1 |
+-----+
9 rows in set (0.00 sec)
```

DybDbi.vld.vsmry.**dump\_ctxsmry** ()

Dump tables for each *cfdir* expressing summary dif info in *cfdir/tn/name* hierarchy:

<b>tn</b>	<b>vlut.rst</b>	<b>vlutorderingSEQNOasc.rst</b>	<b>vlutorderingSEQNOdesc.rst</b>
CableMap	16/35	1/35	19/35
CalibFeeSpec	1/1	1/1	1/1
CalibPmtHighGain	0/6	0/6	0/6
CalibPmtPedBias	0/1	0/1	0/1

DybDbi.vld.vsmry.**dump\_difctx** (*cfdir*='tmp\_offline\_db\_cf\_fix\_offline\_db', *name*='vlut.rst')

For each *tn* within *cfdir/name* Dump tables listing *ctx* with differences and their ranges of difference in INSERT-DATE,TIMESTART space

DybDbi.vld.vsmry.**grow\_cf** (*tn*, *ctx*, *cfdir*, *name*, *stat*)

#### Parameters

- **tn** – table name
- **ctx** – context
- **cfdir** – comparison dir
- **name** – eg vlut.rst
- **stat** – statistics and range dict

Collect lists of all *ctx* and *ctx* with *ndif* > 0 into *cf* dict keyed into changed hierarchy *cfdir/tn/name*

DybDbi.vld.vsmry.**present\_smry** ()

Currently needs manual hookup to global index.rst

DybDbi.vld.vsmry.**squeeze\_tab** (*dohd*, *cols*, *kn*)

#### Parameters

- **dohd** – dict of hashdicts
- **cols** – presentation ordering of keys in the hashdicts
- **kn** – name of the *dohd* key that becomes added column for gang up referencing

Suppress duplicate value entries in a table by ganging

Simple lookups:

### 23.7.11 DybDbi.IRunLookup

**class** DybDbi.IRunLookup (*\*args*, *\*\*kwa*)

Bases: DybDbi.ilookup.ILookup

Specialization of DybDbi.ILookup, for looking for run numbers in GDaqRunInfo, usage:

```
iargs = (10,100,1000)
irl = IRunLookup( *iargs )
for ia in iargs:
    print ia, irl[ia]
```

### 23.7.12 DybDbi.ILookup

**class** DybDbi.ILookup (*\*args*, *\*\*kwa*)

Bases: dict

Example of use:

```
il = ILookup( 10,100,1000, kls='GDaqRunInfo', ifield="runNo", iattr="RunNo" )
# corresponds to datasql WHERE clause : runNo in (10,100,1000)
print il[10]
```

The positional arguments are used in `datasql` **IN** list, the query must result in the same number of entries as positional arguments. The `iattr` is needed as `DybDbi` attribute names are often different from fieldnames, this is used after the query with in memory lookup to arrange the results of the query by the argument values.

Effectively the positional arguments must behave like primary keys with each arg corresponding to one row.

### 23.7.13 `DybDbi.AdLogicalPhysical`

**class** `DybDbi.AdLogicalPhysical` (*timestamp=None, purgecache=False, DROP=False*)

Bases: `dict`

Provides access to logical/physical mappings from the DBI table `gendbi-gphysad`, with functionality to read the mappings at particular timestamps and write them with validity time range.

1. logical slots are expressed as tuples (`site, subsite`) such as (`Site.kSAB, DetectorId.kAD1`)

2. physical AD indices 1, 2, ... 8 corresponding to AD1, AD2, ... AD8

Mappings are stored within this dict keyed by logical slot. Reverse `physical->logical` lookups are provide by the `__call__` method:

```
alp = AdLogicalPhysical()
site, subsite = alp(1)          ## find where AD1 is
```

An input `physadid` of **None** is used to express a vacated slot, and results in the writing of a payloadless DBI validity. Such **None** are not read back into this dict on reading, it being regarded as a write signal only.

For usage examples see `dybgaudi:Database/DybDbi/tests/test_physad.py`

#### Parameters

- **timestamp** – time at which to lookup mapping, defaults of `None` is promoted to now (UTC naturally)
- **purgecache** – clear cache before reading, needed when which to read updates from same process that wrote them
- **DROP** – drop the table and zero the `LASTUSEDSEQNO` (**only use during development**)

Read current mappings from `PhysAd` DB table, usage:

```
alp = AdLogicalPhysical()
print alp
blp = AdLogicalPhysical(timestamp=TimeStamp(2011,10,10,0,0,0))
print blp
```

Direct lookup:

```
physadid = alp.get((site,subsite), None)
if physadid:
    print "(%s)s,%(subsite)s => %(physadid)s " % locals()
```

To update mappings in memory:

```
alp.update({(Site.kSAB,DetectorId.kAD1):1,(Site.kDayaBay,DetectorId.kAD2):2})
```

Vacating a formerly occupied slot is done using `None`:

```
alp.update({ (Site.kSAB, DetectorId.kAD1) : None, (Site.kDayaBay, DetectorId.kAD1) : 1})
```

To persist the update to the DB, within a particular timerange:

```
alp.write( timestart=TimeStamp.kNow() )
```

Read back by instantiating a new instance:

```
blp = AdLogicalPhysical( timestamp=... )
```

Reverse lookup from physical AD id 1, 2, 3..8 to logical slot:

```
sitesubsite = alp(1)          ## invokes the __call__ method for reverse lookup
if sitesubsite:
    site, subsite = sitesubsite
else:
    print "not found"
```

**check\_physical2logical()**

Self consistency check Test that the call returns the expected slot, verifying that the physical2logic dict is in step

**kls**

alias of GPhysAd

**classmethod lookup\_logical2physical** (timestamp, sitesubsite, simflag=1)

**Parameters**

- **timestamp** –
- **sitesubsite** –
- **simflag** –

**Return physadid, vrec**

Note that a payloadless DBI query result is interpreted to mean an empty logical slot resulting in the return of a physadid of **None**

Cannot use `kAnySubSite = -1` to avoid querying every slot as DBI non-aggregate reads always correspond to a single SEQNO

**write** (timestart=None, timeend=None)

Writes mappings expressed in this dict into DB

**Parameters**

- **timestart** –
- **timeend** –

Context basis classes from `dybgaudi:DataModel/Context/Context`

### 23.7.14 DybDbi.Context

The underlying C++ class is defined in `context:Context.h`.

**class DybDbi.Context** (int site, int flag, const TimeStamp& time='TimeStamp()', int det='kUnknown')

Bases: `ROOT.ObjectProxy`

`Context::Context()` `Context::Context(const Context& other)` `Context::Context(int site, int flag, const TimeStamp& time = TimeStamp(), int det = kUnknown)`

```
AsString
    std::string Context::AsString(char* option = "")

GetDetId
    int Context::GetDetId()

GetSimFlag
    int Context::GetSimFlag()

GetSite
    int Context::GetSite()

GetTimeStamp
    TimeStamp& Context::GetTimeStamp()

IsA
    TClass* Context::IsA()

IsValid
    bool Context::IsValid()

SetDetId
    void Context::SetDetId(int det)

SetSimFlag
    void Context::SetSimFlag(int flag)

SetSite
    void Context::SetSite(int site)

SetTimeStamp
    void Context::SetTimeStamp(const TimeStamp& ts)

ShowMembers
    void Context::ShowMembers(TMemberInspector&, char*)

detid
    int Context::GetDetId()

simflag
    int Context::GetSimFlag()

site
    int Context::GetSite()

timestamp
    TimeStamp& Context::GetTimeStamp()
```

### 23.7.15 DybDbi.ContextRange

The underlying C++ class is defined in `context:ContextRange.h`.

```
class DybDbi.ContextRange (const int siteMask, const int simMask, const TimeStamp& tstart, const
                          TimeStamp& tend)
    Bases: ROOT.ObjectProxy

    ContextRange::ContextRange(const ContextRange&) ContextRange::ContextRange() Context-
    tRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp&
    tend)

    AsString
        std::string ContextRange::AsString(char* option = "")
```

```

GetSimMask
    int ContextRange::GetSimMask()

GetSiteMask
    int ContextRange::GetSiteMask()

GetTimeEnd
    TimeStamp ContextRange::GetTimeEnd()

GetTimeStart
    TimeStamp ContextRange::GetTimeStart()

IsA
    TClass* ContextRange::IsA()

IsCompatible
    bool ContextRange::IsCompatible(const Context& cx) bool ContextRange::IsCompatible(Context* cx)

SetSimMask
    void ContextRange::SetSimMask(const int simMask)

SetSiteMask
    void ContextRange::SetSiteMask(const int siteMask)

SetTimeEnd
    void ContextRange::SetTimeEnd(const TimeStamp& tend)

SetTimeStart
    void ContextRange::SetTimeStart(const TimeStamp& tstart)

ShowMembers
    void ContextRange::ShowMembers(TMemberInspector&, char*)

TrimTo
    void ContextRange::TrimTo(const ContextRange& other)

simmask
    int ContextRange::GetSimMask()

sitemask
    int ContextRange::GetSiteMask()

timeend
    TimeStamp ContextRange::GetTimeEnd()

timestart
    TimeStamp ContextRange::GetTimeStart()

```

### 23.7.16 DybDbi.TimeStamp

Underlying C++ class is defined in `context:TimeStamp.h`

```

class DybDbi.TimeStamp (unsigned int year, unsigned int month, unsigned int day, unsigned int hour, un-
signed int min, unsigned int sec, unsigned int nsec=0, bool isUTC='true', int
secOffset=0)

```

Bases: `ROOT.ObjectProxy`

Pythonic extensions to underlying DBI TimeStamp assume that **all TimeStamps are expressing UTC times** (this is the default)

```
In [2]: ts = TimeStamp.kNow()

In [3]: ts.UTCtoDatetime.ctime()
Out[3]: 'Thu May 26 13:10:20 2011'

In [4]: ts.UTCtoNaiveLocalDatetime.ctime()
Out[4]: 'Thu May 26 13:10:20 2011'

In [5]: ts.UTCtoDatetime
Out[5]: datetime.datetime(2011, 5, 26, 13, 10, 20, tzinfo=<DydbDi.TimeStampExt.UTC object at 0xb...

In [6]: ts.UTCtoNaiveLocalDatetime          ## useful for comparisons with naive dateti
Out[6]: datetime.datetime(2011, 5, 26, 13, 10, 20)
```

TimeStamp::TimeStamp() TimeStamp::TimeStamp(const TimeStamp& source) TimeStamp::TimeStamp(const timespec& ts) TimeStamp::TimeStamp(const time\_t& t, const int nsec) TimeStamp::TimeStamp(unsigned int year, unsigned int month, unsigned int day, unsigned int hour, unsigned int min, unsigned int sec, unsigned int nsec = 0, bool isUTC = true, int secOffset = 0) TimeStamp::TimeStamp(unsigned int date, unsigned int time, unsigned int nsec, bool isUTC = true, int secOffset = 0) TimeStamp::TimeStamp(double seconds)

**Add**

void TimeStamp::Add(const TimeStamp& offset) void TimeStamp::Add(double seconds)

**AsString**

char\* TimeStamp::AsString(char\* option = "")

**CloneAndSubtract**

TimeStamp TimeStamp::CloneAndSubtract(const TimeStamp& offset)

**Copy**

void TimeStamp::Copy(TimeStamp& vldts)

**DumpTMStruct**

static void TimeStamp::DumpTMStruct(const tm& tmstruct)

**GetBOT**

static TimeStamp TimeStamp::GetBOT()

**GetDate**

int TimeStamp::GetDate(bool inUTC = true, int secOffset = 0, unsigned int\* year = 0, unsigned int\* month = 0, unsigned int\* day = 0)

**GetEOT**

static TimeStamp TimeStamp::GetEOT()

**GetNBOT**

static TimeStamp TimeStamp::GetNBOT()

**GetNanoSec**

int TimeStamp::GetNanoSec()

**GetSec**

long TimeStamp::GetSec()

**GetSeconds**

double TimeStamp::GetSeconds()

**GetTime**

int TimeStamp::GetTime(bool inUTC = true, int secOffset = 0, unsigned int\* hour = 0, unsigned int\* min = 0, unsigned int\* sec = 0)

```

GetTimeSpec
    timespec TimeStamp::GetTimeSpec()

GetZoneOffset
    static int TimeStamp::GetZoneOffset()

IsA
    TClass* TimeStamp::IsA()

IsLeapYear
    static bool TimeStamp::IsLeapYear(int year)

IsNull
    bool TimeStamp::IsNull()

MktimeFromUTC
    static long TimeStamp::MktimeFromUTC(tm* tmstruct)

Print
    void TimeStamp::Print(char* option = "")

ShowMembers
    void TimeStamp::ShowMembers(TMemberInspector&, char*)

Subtract
    void TimeStamp::Subtract(const TimeStamp& offset) void TimeStamp::Subtract(double seconds)

UTCtoDatetime
    From an assumed UTC TimeStamp return tz aware datetime

UTCtoNaiveLocalDatetime
    From an assumed UTC TimeStamp return naive local datetime

    ts = TimeStamp.kNow()
    'Thu, 26 May 2011 04:41:03 +0000 (GMT) +          0 nsec

    ts.UTCtoDatetime.ctime()
    'Thu May 26 12:41:03 2011'

    ts.UTCtoNaiveLocalDatetime.ctime()
    'Thu May 26 12:41:03 2011'

bot
    static TimeStamp TimeStamp::GetBOT()

date
    int TimeStamp::GetDate(bool inUTC = true, int secOffset = 0, unsigned int* year = 0, unsigned int* month
    = 0, unsigned int* day = 0)

eot
    static TimeStamp TimeStamp::GetEOT()

classmethod kNow()
    TimeStamp object representing current UTC time

nanosec
    int TimeStamp::GetNanoSec()

nbot
    static TimeStamp TimeStamp::GetNBOT()

sec
    long TimeStamp::GetSec()

```

**seconds**

double TimeStamp::GetSeconds()

**time**

int TimeStamp::GetTime(bool inUTC = true, int secOffset = 0, unsigned int\* hour = 0, unsigned int\* min = 0, unsigned int\* sec = 0)

**timespec**

timespec TimeStamp::GetTimeSpec()

**zoneoffset**

static int TimeStamp::GetZoneOffset()

### 23.7.17 DybDbi.ServiceMode

The underlying C++ class is defined in `context:ServiceMode.h`.

**class** DybDbi.ServiceMode (*const Context& context, int task*)

Bases: ROOT.ObjectProxy

ServiceMode::ServiceMode(const ServiceMode&) ServiceMode::ServiceMode() Service-  
Mode::ServiceMode(const Context& context, int task)**IsA**

TClass\* ServiceMode::IsA()

**ShowMembers**

void ServiceMode::ShowMembers(TMemberInspector&amp;, char\*)

**context**

Context&amp; ServiceMode::context()

**task**

int&amp; ServiceMode::task()

Convention basis classes from `:dybgaudi:DataModel/Conventions/Conventions`

### 23.7.18 DybDbi.Site

The underlying enum is defined in `conventions:Site.h`.

**class** DybDbi.Site

Bases: ROOT.ObjectProxy

**AsString**

char\* Site::AsString(int site)

**FromString**

int Site::FromString(char\* str)

**FullMask**

int Site::FullMask()

**MaskFromString**

int Site::MaskFromString(char\* str)

**StringFromMask**

char\* Site::StringFromMask(int mask)

### 23.7.19 DybDbi.SimFlag

The underlying enum is defined in `conventions:SimFlag.h`.

```
class DybDbi.SimFlag
    Bases: ROOT.ObjectProxy

    AsString
        char* SimFlag::AsString(int flag)

    FromString
        int SimFlag::FromString(char* str)

    FullMask
        int SimFlag::FullMask()

    StringFromMask
        char* SimFlag::StringFromMask(int mask)
```

### 23.7.20 DybDbi.DetectorId

```
class DybDbi.DetectorId
    Bases: ROOT.ObjectProxy

    AsString
        char* DetectorId::AsString(int id)

    FromString
        int DetectorId::FromString(char* str)

    FromString0
        int DetectorId::FromString0(char* str)

    isAD
        bool DetectorId::isAD(int id)

    isRPC
        bool DetectorId::isRPC(int id)

    isWaterShield
        bool DetectorId::isWaterShield(int id)
```

### 23.7.21 DybDbi.Detector

```
DybDbi.Detector
    alias of DayaBay::Detector
```

### 23.7.22 DybDbi.DetectorSensor

```
class DybDbi.DetectorSensor (unsigned int sensor_id, int site, int det)
    Bases: DybDbi.DayaBay::DetectorSensor

    DetectorSensor::DetectorSensor() DetectorSensor::DetectorSensor(unsigned int sensor_id, int site, int det)
    DetectorSensor::DetectorSensor(const DayaBay::DetectorSensor& sensor) DetectorSensor::DetectorSensor(int
    data)
```

### 23.7.23 DybDbi.FeeChannelId

**class** DybDbi.FeeChannelId(*int board, int connector, int site, int det*)

Bases: DybDbi.DayaBay::FeeChannelId

FeeChannelId::FeeChannelId() FeeChannelId::FeeChannelId(int board, int connector, int site, int det) FeeChannelId::FeeChannelId(const DayaBay::FeeChannelId& channel) FeeChannelId::FeeChannelId(int data)

### 23.7.24 DybDbi.FeeHardwareId

**class** DybDbi.FeeHardwareId(*int boardId, int connector*)

Bases: DybDbi.DayaBay::FeeHardwareId

FeeHardwareId::FeeHardwareId(const DayaBay::FeeHardwareId&) FeeHardwareId::FeeHardwareId() FeeHardwareId::FeeHardwareId(int boardId, int connector) FeeHardwareId::FeeHardwareId(int data)

### 23.7.25 DybDbi.PmtHardwareId

**class** DybDbi.PmtHardwareId(*unsigned int id, int hardware*)

Bases: DybDbi.DayaBay::PmtHardwareId

PmtHardwareId::PmtHardwareId(const DayaBay::PmtHardwareId&) PmtHardwareId::PmtHardwareId() PmtHardwareId::PmtHardwareId(unsigned int id, int hardware) PmtHardwareId::PmtHardwareId(int data)

Enums:

### 23.7.26 DybDbi.Dbi

**class** DybDbi.Dbi

Bases: ROOT.ObjectProxy

**GetTimeGate**

int Dbi::GetTimeGate(const string& tableName)

**GetVldDescr**

std::string Dbi::GetVldDescr(char\* tableName, Bool\_t isTemporary = false)

**MakeDateTimeString**

std::string Dbi::MakeDateTimeString(const TimeStamp& timeStamp)

**MakeTimeStamp**

TimeStamp Dbi::MakeTimeStamp(const string& sqlDateTime, bool\* ok = 0)

**NotGlobalSeqNo**

bool Dbi::NotGlobalSeqNo(UInt\_t seqNo)

**SetTimeGate**

void Dbi::SetTimeGate(const string& tableName, Int\_t timeGate)

**UsernameFromEnvironment**

std::string Dbi::UsernameFromEnvironment()

**timegate**

int Dbi::GetTimeGate(const string& tableName)

**vlddescr**

std::string Dbi::GetVldDescr(char\* tableName, Bool\_t isTemporary = false)

Generated row classes:

### 23.7.27 DybDbi.GPhysAd

**class** DybDbi.GPhysAd (*const GPhysAd& from*)

Bases: DybDbi.DbTableRow

This table can be read/written using `DybDbi.AdLogicalPhysical`

(adapted from Dan/Zhimin email 2011-01-17)

There are two ways to identify an AD in the experiment:

- 1.Location: SAB-AD1, ..., FAR-AD4
- 2.Physical ID: AD1, AD2, ..., AD8

#### Convention references

Convention	Reference
DCS	<a href="#">doc:3198</a>
DAQ	<a href="#">doc:3442 page 6</a>

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:

#### 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	.	.	.	.

**Detector/MainSys (Name and ID)**

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	.	.	.	.

GPhysAd::GPhysAd() GPhysAd::GPhysAd(const GPhysAd& from) GPhysAd::GPhysAd(int PhysAdId)

**AssignTimeGate**

static void GPhysAd::AssignTimeGate(Int\_t seconds, char\* alternateName = 0)

**Cache**

static DbtCache\* GPhysAd::Cache(char\* alternateName = 0)

**CanL2Cache**

bool GPhysAd::CanL2Cache()

**Close**

static void GPhysAd::Close(char\* filepath = 0l)

**Compare**

bool GPhysAd::Compare(const GPhysAd& that)

**classmethod Create (\*args, \*\*kwargs)**

Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**CreateTableRow**

DbtTableRow\* GPhysAd::CreateTableRow()

**CurrentTimeGate**

static int GPhysAd::CurrentTimeGate(char\* alternateName = 0)

**DoubleValueForKey**

double GPhysAd::DoubleValueForKey(char\* key, double defval = -0x00000000000000001)

**Fill**

void GPhysAd::Fill(DbtResultSet& rs, DbtValidityRec\* vrec)

**FloatValueForKey**

float GPhysAd::FloatValueForKey(char\* key, float defval = -0x00000000000000001)

**GetDatabaseLayout**

std::string GPhysAd::GetDatabaseLayout()

**GetDigest**

std::string GPhysAd::GetDigest()

**GetFields**

std::string GPhysAd::GetFields()

**GetPhysAdId**

```
int GPhysAd::GetPhysAdId()
```

**GetTableDescr**

```
static std::string GPhysAd::GetTableDescr(char* alternateName = 0)
```

**GetTableProxy**

```
static DbTableProxy& GPhysAd::GetTableProxy(char* alternateName = 0)
```

**GetValues**

```
std::string GPhysAd::GetValues()
```

**IntValueForKey**

```
int GPhysAd::IntValueForKey(char* key, int defval = -0x0000000000000001)
```

**IsA**

```
TClass* GPhysAd::IsA()
```

**Rpt**

```
static DbRpt<GPhysAd>* GPhysAd::Rpt(char* ctx = GPhysAd::MetaRctx)
```

**Save**

```
void GPhysAd::Save()
```

**SetPhysAdId**

```
void GPhysAd::SetPhysAdId(int PhysAdId)
```

**ShowMembers**

```
void GPhysAd::ShowMembers(TMemberInspector&, char*)
```

**SpecKeys**

```
static TList* GPhysAd::SpecKeys()
```

**SpecList**

```
static TList* GPhysAd::SpecList()
```

**SpecMap**

```
static TMap* GPhysAd::SpecMap()
```

**Store**

```
void GPhysAd::Store(DbOutRowStream& ors, DbValidityRec* vrec)
```

**Wrt**

```
static DbWrt<GPhysAd>* GPhysAd::Wrt(char* ctx = GPhysAd::MetaWctx)
```

**aggregateno**

```
int DbTableRow::GetAggregateNo()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
```

```
GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod** `csv_import` (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp& tend)

```
ql> select * from CalibPmtSpecVId ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaselayout**

std::string GPhysAd::GetDatabaseLayout()

**digest**

std::string GPhysAd::GetDigest()

**extracondition**

std::string DbfTableRow::GetExtraCondition()

**fields**

std::string GPhysAd::GetFields()

**name**

std::string GPhysAd::name()

**physadid**

int GPhysAd::GetPhysAdId()

**tabledescr**

static std::string GPhysAd::GetTableDescr(char\* alternateName = 0)

**tableproxy**

static DbfTableProxy& GPhysAd::GetTableProxy(char\* alternateName = 0)

**values**

std::string GPhysAd::GetValues()

### 23.7.28 DybDbf.GSimPmtSpec

**class** `DybDbf.GSimPmtSpec` (*DayaBay::DetectorSensor PmtId*, *string Describ*, *double Gain*, *double SigmaGain*, *double TimeOffset*, *double TimeSpread*, *double Efficiency*, *double PrePulseProb*, *double AfterPulseProb*, *double DarkRate*)

Bases: `DybDbf.DbfTableRow`

docstring

GSimPmtSpec::GSimPmtSpec() GSimPmtSpec::GSimPmtSpec(const GSimPmtSpec& from) GSimPmtSpec::GSimPmtSpec(DayaBay::DetectorSensor PmtId, string Describ, double Gain, double SigmaGain, double TimeOffset, double TimeSpread, double Efficiency, double PrePulseProb, double AfterPulseProb, double DarkRate)

**AssignTimeGate**

static void GSimPmtSpec::AssignTimeGate(Int\_t seconds, char\* alternateName = 0)

**Cache**

static DbCache\* GSimPmtSpec::Cache(char\* alternateName = 0)

**CanFixOrdering**

bool GSimPmtSpec::CanFixOrdering()

**CanL2Cache**

bool GSimPmtSpec::CanL2Cache()

**Close**

static void GSimPmtSpec::Close(char\* filepath = 0)

**Compare**

bool GSimPmtSpec::Compare(const GSimPmtSpec& that)

**classmethod Create (\*args, \*\*kwargs)**

Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**CreateTableRow**

DbTableRow\* GSimPmtSpec::CreateTableRow()

**CurrentTimeGate**

static int GSimPmtSpec::CurrentTimeGate(char\* alternateName = 0)

**DoubleValueForKey**

double GSimPmtSpec::DoubleValueForKey(char\* key, double defval = -0x0000000000000001)

**Fill**

void GSimPmtSpec::Fill(DbResultSet& rs, DbValidityRec\* vrec)

**FloatValueForKey**

float GSimPmtSpec::FloatValueForKey(char\* key, float defval = -0x0000000000000001)

**GetAfterPulseProb**

double GSimPmtSpec::GetAfterPulseProb()

**GetDarkRate**

double GSimPmtSpec::GetDarkRate()

**GetDatabaseLayout**

std::string GSimPmtSpec::GetDatabaseLayout()

**GetDescrib**

std::string GSimPmtSpec::GetDescrib()

**GetDigest**

std::string GSimPmtSpec::GetDigest()

**GetEfficiency**

double GSimPmtSpec::GetEfficiency()

**GetFields**

std::string GSimPmtSpec::GetFields()

**GetGain**  
double GSimPmtSpec::GetGain()

**GetPmtId**  
DayaBay::DetectorSensor GSimPmtSpec::GetPmtId()

**GetPrePulseProb**  
double GSimPmtSpec::GetPrePulseProb()

**GetSigmaGain**  
double GSimPmtSpec::GetSigmaGain()

**GetTableDescr**  
static std::string GSimPmtSpec::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
static DbiParamProxy& GSimPmtSpec::GetTableProxy(char\* alternateName = 0)

**GetTimeOffset**  
double GSimPmtSpec::GetTimeOffset()

**GetTimeSpread**  
double GSimPmtSpec::GetTimeSpread()

**GetValues**  
std::string GSimPmtSpec::GetValues()

**IntValueForKey**  
int GSimPmtSpec::IntValueForKey(char\* key, int defval = -0x00000000000000000001)

**IsA**  
TClass\* GSimPmtSpec::IsA()

**Rpt**  
static DbiParamProxy<GSimPmtSpec>\* GSimPmtSpec::Rpt(char\* ctx = GSimPmtSpec::MetaRctx)

**Save**  
void GSimPmtSpec::Save()

**SetAfterPulseProb**  
void GSimPmtSpec::SetAfterPulseProb(double AfterPulseProb)

**SetDarkRate**  
void GSimPmtSpec::SetDarkRate(double DarkRate)

**SetDescrib**  
void GSimPmtSpec::SetDescrib(string Describ)

**SetEfficiency**  
void GSimPmtSpec::SetEfficiency(double Efficiency)

**SetGain**  
void GSimPmtSpec::SetGain(double Gain)

**SetPmtId**  
void GSimPmtSpec::SetPmtId(DayaBay::DetectorSensor PmtId)

**SetPrePulseProb**  
void GSimPmtSpec::SetPrePulseProb(double PrePulseProb)

**SetSigmaGain**  
void GSimPmtSpec::SetSigmaGain(double SigmaGain)

**SetTimeOffset**

```
void GSimPmtSpec::SetTimeOffset(double TimeOffset)
```

**SetTimeSpread**

```
void GSimPmtSpec::SetTimeSpread(double TimeSpread)
```

**ShowMembers**

```
void GSimPmtSpec::ShowMembers(TMemberInspector&, char*)
```

**SpecKeys**

```
static TList* GSimPmtSpec::SpecKeys()
```

**SpecList**

```
static TList* GSimPmtSpec::SpecList()
```

**SpecMap**

```
static TMap* GSimPmtSpec::SpecMap()
```

**Store**

```
void GSimPmtSpec::Store(DbiOutRowStream& ors, DbiValidityRec* vrec)
```

**Wrt**

```
static DbiWrt<GSimPmtSpec>* GSimPmtSpec::Wrt(char* ctx = GSimPmtSpec::MetaWctx)
```

**afterpulseprob**

```
double GSimPmtSpec::GetAfterPulseProb()
```

**aggregateno**

```
int DbiTableRow::GetAggregateNo()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPulse")
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod csv\_import** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

```
ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart,
const TimeStamp& tend)
```

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
```

```

| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+
26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1 |
0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |

```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**darkrate**

```
double GSimPmtSpec::GetDarkRate()
```

**databaselayout**

```
std::string GSimPmtSpec::GetDatabaseLayout()
```

**describ**

```
std::string GSimPmtSpec::GetDescrib()
```

**digest**

```
std::string GSimPmtSpec::GetDigest()
```

**efficiency**

```
double GSimPmtSpec::GetEfficiency()
```

**extracondition**

```
std::string DbfTableRow::GetExtraCondition()
```

**fields**

```
std::string GSimPmtSpec::GetFields()
```

**gain**

```
double GSimPmtSpec::GetGain()
```

**name**

```
std::string GSimPmtSpec::name()
```

**pmtid**

```
DayaBay::DetectorSensor GSimPmtSpec::GetPmtId()
```

**prepulseprob**

```
double GSimPmtSpec::GetPrePulseProb()
```

**sigmagain**

```
double GSimPmtSpec::GetSigmaGain()
```

**tabledescr**

```
static std::string GSimPmtSpec::GetTableDescr(char* alternateName = 0)
```

**tableproxy**

```
static DbfTableProxy& GSimPmtSpec::GetTableProxy(char* alternateName = 0)
```

**timeoffset**

```
double GSimPmtSpec::GetTimeOffset()
```

**timespread**

```
double GSimPmtSpec::GetTimeSpread()
```

**values**

```
std::string GSimPmtSpec::GetValues()
```

### 23.7.29 DybDbi.GCalibPmtSpec

```

class DybDbi.GCalibPmtSpec (int PmtId, string Describ, int Status, double SpeHigh, double SigmaSpe-
                          High, double SpeLow, double TimeOffset, double TimeSpread, double Ef-
                          ficiency, double PrePulseProb, double AfterPulseProb, double DarkRate)
  Bases: DybDbi.DbiTableRow

  docstring

  GCalibPmtSpec::GCalibPmtSpec() GCalibPmtSpec::GCalibPmtSpec(const GCalibPmtSpec& from) GCal-
  ibPmtSpec::GCalibPmtSpec(int PmtId, string Describ, int Status, double SpeHigh, double SigmaSpeHigh, dou-
  ble SpeLow, double TimeOffset, double TimeSpread, double Efficiency, double PrePulseProb, double After-
  PulseProb, double DarkRate)

  AssignTimeGate
    static void GCalibPmtSpec::AssignTimeGate(Int_t seconds, char* alternateName = 0)

  Cache
    static Dbicache* GCalibPmtSpec::Cache(char* alternateName = 0)

  CanL2Cache
    bool GCalibPmtSpec::CanL2Cache()

  Close
    static void GCalibPmtSpec::Close(char* filepath = 0)

  Compare
    bool GCalibPmtSpec::Compare(const GCalibPmtSpec& that)

  classmethod Create (*args, **kwargs)
    Provide pythonic instance creation classmethod:

    i = GTableName.Create( AttributeName=100. , ... )

  CreateTableRow
    DbitableRow* GCalibPmtSpec::CreateTableRow()

  CurrentTimeGate
    static int GCalibPmtSpec::CurrentTimeGate(char* alternateName = 0)

  DoubleValueForKey
    double GCalibPmtSpec::DoubleValueForKey(char* key, double defval = -0x00000000000000001)

  Fill
    void GCalibPmtSpec::Fill(DbiResultSet& rs, DbivalidityRec* vrec)

  FloatValueForKey
    float GCalibPmtSpec::FloatValueForKey(char* key, float defval = -0x00000000000000001)

  GetAfterPulseProb
    double GCalibPmtSpec::GetAfterPulseProb()

  GetDarkRate
    double GCalibPmtSpec::GetDarkRate()

  GetDatabaseLayout
    std::string GCalibPmtSpec::GetDatabaseLayout()

  GetDescrib
    std::string GCalibPmtSpec::GetDescrib()

  GetDigest
    std::string GCalibPmtSpec::GetDigest()

```

**GetEfficiency**  
double GCalibPmtSpec::GetEfficiency()

**GetFields**  
std::string GCalibPmtSpec::GetFields()

**GetPmtId**  
int GCalibPmtSpec::GetPmtId()

**GetPrePulseProb**  
double GCalibPmtSpec::GetPrePulseProb()

**GetSigmaSpeHigh**  
double GCalibPmtSpec::GetSigmaSpeHigh()

**GetSpeHigh**  
double GCalibPmtSpec::GetSpeHigh()

**GetSpeLow**  
double GCalibPmtSpec::GetSpeLow()

**GetStatus**  
int GCalibPmtSpec::GetStatus()

**GetTableDescr**  
static std::string GCalibPmtSpec::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
static DbTableProxy& GCalibPmtSpec::GetTableProxy(char\* alternateName = 0)

**GetTimeOffset**  
double GCalibPmtSpec::GetTimeOffset()

**GetTimeSpread**  
double GCalibPmtSpec::GetTimeSpread()

**GetValues**  
std::string GCalibPmtSpec::GetValues()

**IntValueForKey**  
int GCalibPmtSpec::IntValueForKey(char\* key, int defval = -0x00000000000000001)

**IsA**  
TClass\* GCalibPmtSpec::IsA()

**Rpt**  
static DbRpt<GCalibPmtSpec>\* GCalibPmtSpec::Rpt(char\* ctx = GCalibPmtSpec::MetaRctx)

**Save**  
void GCalibPmtSpec::Save()

**SetAfterPulseProb**  
void GCalibPmtSpec::SetAfterPulseProb(double AfterPulseProb)

**SetDarkRate**  
void GCalibPmtSpec::SetDarkRate(double DarkRate)

**SetDescrib**  
void GCalibPmtSpec::SetDescrib(string Describ)

**SetEfficiency**  
void GCalibPmtSpec::SetEfficiency(double Efficiency)

```

SetPmtId
    void GCalibPmtSpec::SetPmtId(int PmtId)

SetPrePulseProb
    void GCalibPmtSpec::SetPrePulseProb(double PrePulseProb)

SetSigmaSpeHigh
    void GCalibPmtSpec::SetSigmaSpeHigh(double SigmaSpeHigh)

SetSpeHigh
    void GCalibPmtSpec::SetSpeHigh(double SpeHigh)

SetSpeLow
    void GCalibPmtSpec::SetSpeLow(double SpeLow)

SetStatus
    void GCalibPmtSpec::SetStatus(int Status)

SetTimeOffset
    void GCalibPmtSpec::SetTimeOffset(double TimeOffset)

SetTimeSpread
    void GCalibPmtSpec::SetTimeSpread(double TimeSpread)

ShowMembers
    void GCalibPmtSpec::ShowMembers(TMemberInspector&, char*)

SpecKeys
    static TList* GCalibPmtSpec::SpecKeys()

SpecList
    static TList* GCalibPmtSpec::SpecList()

SpecMap
    static TMap* GCalibPmtSpec::SpecMap()

Store
    void GCalibPmtSpec::Store(DbiOutRowStream& ors, DbiValidityRec* vrec)

Wrt
    static DbiWrt<GCalibPmtSpec>* GCalibPmtSpec::Wrt(char* ctx = GCalibPmtSpec::MetaWctx)

afterpulseprob
    double GCalibPmtSpec::GetAfterPulseProb()

aggregateno
    int DbiTableRow::GetAggregateNo()

classmethod csv_check (path, **kwargs)
    Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

    from DybDbi import GCalibPmtSpec
    GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls

    Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive
    auto mapping is applied to avoid the need for tedious full mapping).

classmethod csv_compare (path, **kwargs)
    compare entries in CSV file with those found in DB

classmethod csv_export (path, **kwargs)
    Export the result of a default context DBI query as a CSV file

```

### Parameters

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod** `csv_import` (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp& tend)

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**darkrate**

double GCalibPmtSpec::GetDarkRate()

**databaselayout**

std::string GCalibPmtSpec::GetDatabaseLayout()

**describ**

std::string GCalibPmtSpec::GetDescrib()

**digest**

std::string GCalibPmtSpec::GetDigest()

**efficiency**

double GCalibPmtSpec::GetEfficiency()

**extracondition**

std::string DbfTableRow::GetExtraCondition()

**fields**

std::string GCalibPmtSpec::GetFields()

**name**

std::string GCalibPmtSpec::name()

**pmtid**

int GCalibPmtSpec::GetPmtId()

**prepulseprob**

double GCalibPmtSpec::GetPrePulseProb()

**sigmaspehigh**

double GCalibPmtSpec::GetSigmaSpeHigh()

**spehigh**

double GCalibPmtSpec::GetSpeHigh()

**spelow**

double GCalibPmtSpec::GetSpeLow()

```

status
    int GCalibPmtSpec::GetStatus()

tabledescr
    static std::string GCalibPmtSpec::GetTableDescr(char* alternateName = 0)

tableproxy
    static DbTableProxy& GCalibPmtSpec::GetTableProxy(char* alternateName = 0)

timeoffset
    double GCalibPmtSpec::GetTimeOffset()

timespread
    double GCalibPmtSpec::GetTimeSpread()

values
    std::string GCalibPmtSpec::GetValues()

```

### 23.7.30 DybDbi.GCalibFeeSpec

```

class DybDbi.GCalibFeeSpec (DayaBay::FeeChannelId ChannelId, int Status, double AdcPedestalHigh,
double AdcPedestalHighSigma, double AdcPedestalLow, double Adc-
PedestalLowSigma, double AdcThresholdHigh, double AdcThreshold-
Low)

```

```

Bases: DybDbi.DbTableRow

```

```

docstring

```

```

GCalibFeeSpec::GCalibFeeSpec() GCalibFeeSpec::GCalibFeeSpec(const GCalibFeeSpec& from) GCal-
ibFeeSpec::GCalibFeeSpec(DayaBay::FeeChannelId ChannelId, int Status, double AdcPedestalHigh, double
AdcPedestalHighSigma, double AdcPedestalLow, double AdcPedestalLowSigma, double AdcThresholdHigh,
double AdcThresholdLow)

```

```

AssignTimeGate

```

```

    static void GCalibFeeSpec::AssignTimeGate(Int_t seconds, char* alternateName = 0)

```

```

Cache

```

```

    static DbCache* GCalibFeeSpec::Cache(char* alternateName = 0)

```

```

CanL2Cache

```

```

    bool GCalibFeeSpec::CanL2Cache()

```

```

Close

```

```

    static void GCalibFeeSpec::Close(char* filepath = 0l)

```

```

Compare

```

```

    bool GCalibFeeSpec::Compare(const GCalibFeeSpec& that)

```

```

classmethod Create (*args, **kwargs)

```

```

    Provide pythonic instance creation classmethod:

```

```

    i = GTableName.Create( AttributeName=100. , ... )

```

```

CreateTableRow

```

```

    DbTableRow* GCalibFeeSpec::CreateTableRow()

```

```

CurrentTimeGate

```

```

    static int GCalibFeeSpec::CurrentTimeGate(char* alternateName = 0)

```

```

DoubleValueForKey

```

```

    double GCalibFeeSpec::DoubleValueForKey(char* key, double defval = -0x00000000000000001)

```

**Fill**  
void GCalibFeeSpec::Fill(DbiResultSet& rs, DbValidityRec\* vrec)

**FloatValueForKey**  
float GCalibFeeSpec::FloatValueForKey(char\* key, float defval = -0x0000000000000001)

**GetAdcPedestalHigh**  
double GCalibFeeSpec::GetAdcPedestalHigh()

**GetAdcPedestalHighSigma**  
double GCalibFeeSpec::GetAdcPedestalHighSigma()

**GetAdcPedestalLow**  
double GCalibFeeSpec::GetAdcPedestalLow()

**GetAdcPedestalLowSigma**  
double GCalibFeeSpec::GetAdcPedestalLowSigma()

**GetAdcThresholdHigh**  
double GCalibFeeSpec::GetAdcThresholdHigh()

**GetAdcThresholdLow**  
double GCalibFeeSpec::GetAdcThresholdLow()

**GetChannelId**  
DayaBay::FeeChannelId GCalibFeeSpec::GetChannelId()

**GetDatabaseLayout**  
std::string GCalibFeeSpec::GetDatabaseLayout()

**GetDigest**  
std::string GCalibFeeSpec::GetDigest()

**GetFields**  
std::string GCalibFeeSpec::GetFields()

**GetStatus**  
int GCalibFeeSpec::GetStatus()

**GetTableDescr**  
static std::string GCalibFeeSpec::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
static DbTableProxy& GCalibFeeSpec::GetTableProxy(char\* alternateName = 0)

**GetValues**  
std::string GCalibFeeSpec::GetValues()

**IntValueForKey**  
int GCalibFeeSpec::IntValueForKey(char\* key, int defval = -0x0000000000000001)

**IsA**  
TClass\* GCalibFeeSpec::IsA()

**Rpt**  
static DbIRpt<GCalibFeeSpec>\* GCalibFeeSpec::Rpt(char\* ctx = GCalibFeeSpec::MetaRctx)

**Save**  
void GCalibFeeSpec::Save()

**SetAdcPedestalHigh**  
void GCalibFeeSpec::SetAdcPedestalHigh(double AdcPedestalHigh)

```

SetAdcPedestalHighSigma
    void GCalibFeeSpec::SetAdcPedestalHighSigma(double AdcPedestalHighSigma)

SetAdcPedestalLow
    void GCalibFeeSpec::SetAdcPedestalLow(double AdcPedestalLow)

SetAdcPedestalLowSigma
    void GCalibFeeSpec::SetAdcPedestalLowSigma(double AdcPedestalLowSigma)

SetAdcThresholdHigh
    void GCalibFeeSpec::SetAdcThresholdHigh(double AdcThresholdHigh)

SetAdcThresholdLow
    void GCalibFeeSpec::SetAdcThresholdLow(double AdcThresholdLow)

SetChannelId
    void GCalibFeeSpec::SetChannelId(DayaBay::FeeChannelId ChannelId)

SetStatus
    void GCalibFeeSpec::SetStatus(int Status)

ShowMembers
    void GCalibFeeSpec::ShowMembers(TMemberInspector&, char*)

SpecKeys
    static TList* GCalibFeeSpec::SpecKeys()

SpecList
    static TList* GCalibFeeSpec::SpecList()

SpecMap
    static TMap* GCalibFeeSpec::SpecMap()

Store
    void GCalibFeeSpec::Store(DbiOutRowStream& ors, DbiValidityRec* vrec)

Wrt
    static DbiWrt<GCalibFeeSpec>* GCalibFeeSpec::Wrt(char* ctx = GCalibFeeSpec::MetaWctx)

adcpedestalhigh
    double GCalibFeeSpec::GetAdcPedestalHigh()

adcpedestalhighsigma
    double GCalibFeeSpec::GetAdcPedestalHighSigma()

adcpedestallow
    double GCalibFeeSpec::GetAdcPedestalLow()

adcpedestallowsigma
    double GCalibFeeSpec::GetAdcPedestalLowSigma()

adcthresholdhigh
    double GCalibFeeSpec::GetAdcThresholdHigh()

adcthresholdlow
    double GCalibFeeSpec::GetAdcThresholdLow()

aggregateno
    int DbiTableRow::GetAggregateNo()

channelid
    DayaBay::FeeChannelId GCalibFeeSpec::GetChannelId()

```

**classmethod `csv_check`** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITERROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod `csv_compare`** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod `csv_export`** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

#### Parameters

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod `csv_import`** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp& tend)

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaseLayout**

std::string GCalibFeeSpec::GetDatabaseLayout()

**digest**

std::string GCalibFeeSpec::GetDigest()

**extracondition**

std::string DbiTableRow::GetExtraCondition()

**fields**

std::string GCalibFeeSpec::GetFields()

**name**

std::string GCalibFeeSpec::name()

**status**

int GCalibFeeSpec::GetStatus()

**tabledescr**

static std::string GCalibFeeSpec::GetTableDescr(char\* alternateName = 0)

```

tableproxy
    static DbitableProxy& GCalibFeeSpec::GetTableProxy(char* alternateName = 0)

values
    std::string GCalibFeeSpec::GetValues()
    
```

### 23.7.31 DybDbi.GFeeCableMap

```

class DybDbi.GFeeCableMap (DayaBay::FeeChannelId FeeChannelId, string FeeChannelDesc,
                          DayaBay::FeeHardwareId FeeHardwareId, string ChanHrdwDesc,
                          DayaBay::DetectorSensor SensorId, string SensorDesc,
                          DayaBay::PmtHardwareId PmtHardwareId, string PmtHrdwDesc)
    
```

Bases: DybDbi.DbiTableRow

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	code2db
DayaBay::FeeChannelId	DybDbi.FeeChannelId	conventions:Electronics.h	.fullPacked-Data()
DayaBay::FeeHardwareId	DybDbi.FeeHardwareId	conventions:Hardware.h	.id()
DayaBay::DetectorSensor	DybDbi.DetectorSensor	conventions:Detectors.h	.fullPacked-Data()
DayaBay::PmtHardwareId	DybDbi.PmtHardwareId	conventions:Hardware.h	.id()

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.

```

GFeeCableMap::GFeeCableMap() GFeeCableMap::GFeeCableMap(const GFeeCableMap& from)
GFeeCableMap::GFeeCableMap(DayaBay::FeeChannelId FeeChannelId, string FeeChannelDesc,
DayaBay::FeeHardwareId FeeHardwareId, string ChanHrdwDesc, DayaBay::DetectorSensor SensorId,
string SensorDesc, DayaBay::PmtHardwareId PmtHardwareId, string PmtHrdwDesc)
    
```

```

AssignTimeGate
    static void GFeeCableMap::AssignTimeGate(Int_t seconds, char* alternateName = 0)
    
```

```

Cache
    static Dbicache* GFeeCableMap::Cache(char* alternateName = 0)
    
```

```

CanL2Cache
    bool GFeeCableMap::CanL2Cache()
    
```

```

Close
    static void GFeeCableMap::Close(char* filepath = 0l)
    
```

```

Compare
    bool GFeeCableMap::Compare(const GFeeCableMap& that)
    
```

```

classmethod Create (*args, **kwargs)
    Provide pythonic instance creation classmethod:
    i = GTableName.Create( AttributeName=100. , ... )
    
```

**CreateTableRow**

DbiTableRow\* GFeeCableMap::CreateTableRow()

**CurrentTimeGate**

static int GFeeCableMap::CurrentTimeGate(char\* alternateName = 0)

**DoubleValueForKey**

double GFeeCableMap::DoubleValueForKey(char\* key, double defval = -0x00000000000000001)

**Fill**

void GFeeCableMap::Fill(DbiResultSet& rs, DbiValidityRec\* vrec)

**FloatValueForKey**

float GFeeCableMap::FloatValueForKey(char\* key, float defval = -0x00000000000000001)

**GetChanHrdwDesc**

std::string GFeeCableMap::GetChanHrdwDesc()

**GetDatabaseLayout**

std::string GFeeCableMap::GetDatabaseLayout()

**GetDigest**

std::string GFeeCableMap::GetDigest()

**GetFeeChannelDesc**

std::string GFeeCableMap::GetFeeChannelDesc()

**GetFeeChannelId**

DayaBay::FeeChannelId GFeeCableMap::GetFeeChannelId()

**GetFeeHardwareId**

DayaBay::FeeHardwareId GFeeCableMap::GetFeeHardwareId()

**GetFields**

std::string GFeeCableMap::GetFields()

**GetPmtHardwareId**

DayaBay::PmtHardwareId GFeeCableMap::GetPmtHardwareId()

**GetPmtHrdwDesc**

std::string GFeeCableMap::GetPmtHrdwDesc()

**GetSensorDesc**

std::string GFeeCableMap::GetSensorDesc()

**GetSensorId**

DayaBay::DetectorSensor GFeeCableMap::GetSensorId()

**GetTableDescr**

static std::string GFeeCableMap::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**

static DbiTableProxy& GFeeCableMap::GetTableProxy(char\* alternateName = 0)

**GetValues**

std::string GFeeCableMap::GetValues()

**IntValueForKey**

int GFeeCableMap::IntValueForKey(char\* key, int defval = -0x00000000000000001)

**IsA**

TClass\* GFeeCableMap::IsA()

**Rpt**

```
static DbIRpt<GFeeCableMap>* GFeeCableMap::Rpt(char* ctx = GFeeCableMap::MetaRctx)
```

**Save**

```
void GFeeCableMap::Save()
```

**SetChanHrdwDesc**

```
void GFeeCableMap::SetChanHrdwDesc(string ChanHrdwDesc)
```

**SetFeeChannelDesc**

```
void GFeeCableMap::SetFeeChannelDesc(string FeeChannelDesc)
```

**SetFeeChannelId**

```
void GFeeCableMap::SetFeeChannelId(DayaBay::FeeChannelId FeeChannelId)
```

**SetFeeHardwareId**

```
void GFeeCableMap::SetFeeHardwareId(DayaBay::FeeHardwareId FeeHardwareId)
```

**SetPmtHardwareId**

```
void GFeeCableMap::SetPmtHardwareId(DayaBay::PmtHardwareId PmtHardwareId)
```

**SetPmtHrdwDesc**

```
void GFeeCableMap::SetPmtHrdwDesc(string PmtHrdwDesc)
```

**SetSensorDesc**

```
void GFeeCableMap::SetSensorDesc(string SensorDesc)
```

**SetSensorId**

```
void GFeeCableMap::SetSensorId(DayaBay::DetectorSensor SensorId)
```

**ShowMembers**

```
void GFeeCableMap::ShowMembers(TMemberInspector&, char*)
```

**SpecKeys**

```
static TList* GFeeCableMap::SpecKeys()
```

**SpecList**

```
static TList* GFeeCableMap::SpecList()
```

**SpecMap**

```
static TMap* GFeeCableMap::SpecMap()
```

**Store**

```
void GFeeCableMap::Store(DbiOutRowStream& ors, DbValidityRec* vrec)
```

**Wrt**

```
static DbIWrt<GFeeCableMap>* GFeeCableMap::Wrt(char* ctx = GFeeCableMap::MetaWctx)
```

**aggregateno**

```
int DbTableRow::GetAggregateNo()
```

**chanhrdwdesc**

```
std::string GFeeCableMap::GetChanHrdwDesc()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod `csv_compare`** (*path*, *\*\*kwargs*)  
compare entries in CSV file with those found in DB

**classmethod `csv_export`** (*path*, *\*\*kwargs*)  
Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod `csv_import`** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp& tend)

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaselayout**

std::string GFeeCableMap::GetDatabaseLayout()

**digest**

std::string GFeeCableMap::GetDigest()

**extracondition**

std::string DbiTableRow::GetExtraCondition()

**feechanneldesc**

std::string GFeeCableMap::GetFeeChannelDesc()

**feechannelid**

DayaBay::FeeChannelId GFeeCableMap::GetFeeChannelId()

**feehardwareid**

DayaBay::FeeHardwareId GFeeCableMap::GetFeeHardwareId()

**fields**

std::string GFeeCableMap::GetFields()

**name**

std::string GFeeCableMap::name()

**pmthardwareid**

DayaBay::PmtHardwareId GFeeCableMap::GetPmtHardwareId()

**pmthrdwdesc**

std::string GFeeCableMap::GetPmtHrdwDesc()

```

sensordesc
    std::string GFeeCableMap::GetSensorDesc()

sensorid
    DayaBay::DetectorSensor GFeeCableMap::GetSensorId()

tabledescr
    static std::string GFeeCableMap::GetTableDescr(char* alternateName = 0)

tableproxy
    static DbiParamProxy& GFeeCableMap::GetTableProxy(char* alternateName = 0)

values
    std::string GFeeCableMap::GetValues()

```

### 23.7.32 DybDbi.GDaqRunInfo

```

class DybDbi.GDaqRunInfo (int RunNo, int TriggerType, string RunType, int DetectorMask, string Parti-
tionName, int SchemaVersion, int DataVersion, int BaseVersion)
    Bases: DybDbi.DbiParamTableRow

    docstring

    GDaqRunInfo::GDaqRunInfo()  GDaqRunInfo::GDaqRunInfo(const GDaqRunInfo& from)  GDaqRun-
    Info::GDaqRunInfo(int RunNo, int TriggerType, string RunType, int DetectorMask, string PartitionName, int
    SchemaVersion, int DataVersion, int BaseVersion)

AssignTimeGate
    static void GDaqRunInfo::AssignTimeGate(Int_t seconds, char* alternateName = 0)

Cache
    static DbiParamCache* GDaqRunInfo::Cache(char* alternateName = 0)

CanL2Cache
    bool GDaqRunInfo::CanL2Cache()

Close
    static void GDaqRunInfo::Close(char* filepath = 0l)

Compare
    bool GDaqRunInfo::Compare(const GDaqRunInfo& that)

classmethod Create (*args, **kwargs)
    Provide pythonic instance creation classmethod:

    i = GTableName.Create( AttributeName=100. , ... )

CreateTableRow
    DbiParamTableRow* GDaqRunInfo::CreateTableRow()

CurrentTimeGate
    static int GDaqRunInfo::CurrentTimeGate(char* alternateName = 0)

DoubleValueForKey
    double GDaqRunInfo::DoubleValueForKey(char* key, double defval = -0x00000000000000001)

Fill
    void GDaqRunInfo::Fill(DbiParamResultSet& rs, DbiParamValidityRec* vrec)

FloatValueForKey
    float GDaqRunInfo::FloatValueForKey(char* key, float defval = -0x00000000000000001)

```

**GetBaseVersion**  
int GDaqRunInfo::GetBaseVersion()

**GetDataVersion**  
int GDaqRunInfo::GetDataVersion()

**GetDatabaseLayout**  
std::string GDaqRunInfo::GetDatabaseLayout()

**GetDetectorMask**  
int GDaqRunInfo::GetDetectorMask()

**GetDigest**  
std::string GDaqRunInfo::GetDigest()

**GetFields**  
std::string GDaqRunInfo::GetFields()

**GetPartitionName**  
std::string GDaqRunInfo::GetPartitionName()

**GetRunNo**  
int GDaqRunInfo::GetRunNo()

**GetRunType**  
std::string GDaqRunInfo::GetRunType()

**GetSchemaVersion**  
int GDaqRunInfo::GetSchemaVersion()

**GetTableDescr**  
static std::string GDaqRunInfo::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
static DbTableProxy& GDaqRunInfo::GetTableProxy(char\* alternateName = 0)

**GetTriggerType**  
int GDaqRunInfo::GetTriggerType()

**GetValues**  
std::string GDaqRunInfo::GetValues()

**IntValueForKey**  
int GDaqRunInfo::IntValueForKey(char\* key, int defval = -0x0000000000000001)

**IsA**  
TClass\* GDaqRunInfo::IsA()

**Rpt**  
static DbIRpt<GDaqRunInfo>\* GDaqRunInfo::Rpt(char\* ctx = GDaqRunInfo::MetaRctx)

**Save**  
void GDaqRunInfo::Save()

**SetBaseVersion**  
void GDaqRunInfo::SetBaseVersion(int BaseVersion)

**SetDataVersion**  
void GDaqRunInfo::SetDataVersion(int DataVersion)

**SetDetectorMask**  
void GDaqRunInfo::SetDetectorMask(int DetectorMask)

**SetPartitionName**

```
void GDaqRunInfo::SetPartitionName(string PartitionName)
```

**SetRunNo**

```
void GDaqRunInfo::SetRunNo(int RunNo)
```

**SetRunType**

```
void GDaqRunInfo::SetRunType(string RunType)
```

**SetSchemaVersion**

```
void GDaqRunInfo::SetSchemaVersion(int SchemaVersion)
```

**SetTriggerType**

```
void GDaqRunInfo::SetTriggerType(int TriggerType)
```

**ShowMembers**

```
void GDaqRunInfo::ShowMembers(TMemberInspector&, char*)
```

**SpecKeys**

```
static TList* GDaqRunInfo::SpecKeys()
```

**SpecList**

```
static TList* GDaqRunInfo::SpecList()
```

**SpecMap**

```
static TMap* GDaqRunInfo::SpecMap()
```

**Store**

```
void GDaqRunInfo::Store(DbiOutRowStream& ors, DbiValidityRec* vrec)
```

**Wrt**

```
static DbiWrt<GDaqRunInfo>* GDaqRunInfo::Wrt(char* ctx = GDaqRunInfo::MetaWctx)
```

**aggregateno**

```
int DbiTableRow::GetAggregateNo()
```

**baseversion**

```
int GDaqRunInfo::GetBaseVersion()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITERROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

```
classmethod csv_import (path, **kwargs)
```

Import CSV file into Database Using default writer context for now

```
ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart,
const TimeStamp& tend)
```

```
ql> select * from CalibPmtSpecVId ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

#### **databaselayout**

```
std::string GDaqRunInfo::GetDatabaseLayout()
```

#### **dataversion**

```
int GDaqRunInfo::GetDataVersion()
```

#### **detectormask**

```
int GDaqRunInfo::GetDetectorMask()
```

#### **digest**

```
std::string GDaqRunInfo::GetDigest()
```

#### **extracondition**

```
std::string DbfTableRow::GetExtraCondition()
```

#### **fields**

```
std::string GDaqRunInfo::GetFields()
```

#### **name**

```
std::string GDaqRunInfo::name()
```

#### **partitionname**

```
std::string GDaqRunInfo::GetPartitionName()
```

#### **runno**

```
int GDaqRunInfo::GetRunNo()
```

#### **runtype**

```
std::string GDaqRunInfo::GetRunType()
```

#### **schemaversion**

```
int GDaqRunInfo::GetSchemaVersion()
```

#### **tabledescr**

```
static std::string GDaqRunInfo::GetTableDescr(char* alternateName = 0)
```

#### **tableproxy**

```
static DbfTableProxy& GDaqRunInfo::GetTableProxy(char* alternateName = 0)
```

#### **triggertype**

```
int GDaqRunInfo::GetTriggerType()
```

#### **values**

```
std::string GDaqRunInfo::GetValues()
```

### 23.7.33 DybDbi.GDaqCalibRunInfo

**class** DybDbi.GDaqCalibRunInfo (const GDaqCalibRunInfo& from)

Bases: DybDbi.DbiTableRow

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:

- doc:3442

- doc:3603

GDaqCalibRunInfo::GDaqCalibRunInfo() GDaqCalibRunInfo::GDaqCalibRunInfo(const GDaqCalibRunInfo& from)

**AssignTimeGate**

static void GDaqCalibRunInfo::AssignTimeGate(Int\_t seconds, char\* alternateName = 0)

**Cache**

static Dbicache\* GDaqCalibRunInfo::Cache(char\* alternateName = 0)

**CanL2Cache**

bool GDaqCalibRunInfo::CanL2Cache()

**Close**

static void GDaqCalibRunInfo::Close(char\* filepath = 0l)

**Compare**

bool GDaqCalibRunInfo::Compare(const GDaqCalibRunInfo& that)

**classmethod Create** (\*args, \*\*kwargs)

Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**CreateTableRow**

DbiTableRow\* GDaqCalibRunInfo::CreateTableRow()

**CurrentTimeGate**

static int GDaqCalibRunInfo::CurrentTimeGate(char\* alternateName = 0)

**DoubleValueForKey**

double GDaqCalibRunInfo::DoubleValueForKey(char\* key, double defval = -0x0000000000000001)

**Fill**

void GDaqCalibRunInfo::Fill(DbiResultSet& rs, DbivalityRec\* vrec)

**FloatValueForKey**

float GDaqCalibRunInfo::FloatValueForKey(char\* key, float defval = -0x0000000000000001)

**GetAdNo**

int GDaqCalibRunInfo::GetAdNo()

**GetDatabaseLayout**

std::string GDaqCalibRunInfo::GetDatabaseLayout()

**GetDetectorId**

int GDaqCalibRunInfo::GetDetectorId()

**GetDigest**

std::string GDaqCalibRunInfo::GetDigest()

**GetDuration**  
int GDaqCalibRunInfo::GetDuration()

**GetFields**  
std::string GDaqCalibRunInfo::GetFields()

**GetHomeA**  
int GDaqCalibRunInfo::GetHomeA()

**GetHomeB**  
int GDaqCalibRunInfo::GetHomeB()

**GetHomeC**  
int GDaqCalibRunInfo::GetHomeC()

**GetLedFreq**  
int GDaqCalibRunInfo::GetLedFreq()

**GetLedNumber1**  
int GDaqCalibRunInfo::GetLedNumber1()

**GetLedNumber2**  
int GDaqCalibRunInfo::GetLedNumber2()

**GetLedPulseSep**  
int GDaqCalibRunInfo::GetLedPulseSep()

**GetLedVoltage1**  
int GDaqCalibRunInfo::GetLedVoltage1()

**GetLedVoltage2**  
int GDaqCalibRunInfo::GetLedVoltage2()

**GetLtbMode**  
int GDaqCalibRunInfo::GetLtbMode()

**GetRunNo**  
int GDaqCalibRunInfo::GetRunNo()

**GetSourceIdA**  
int GDaqCalibRunInfo::GetSourceIdA()

**GetSourceIdB**  
int GDaqCalibRunInfo::GetSourceIdB()

**GetSourceIdC**  
int GDaqCalibRunInfo::GetSourceIdC()

**GetTableDescr**  
static std::string GDaqCalibRunInfo::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
static DbTableProxy& GDaqCalibRunInfo::GetTableProxy(char\* alternateName = 0)

**GetValues**  
std::string GDaqCalibRunInfo::GetValues()

**GetZPositionA**  
int GDaqCalibRunInfo::GetZPositionA()

**GetZPositionB**  
int GDaqCalibRunInfo::GetZPositionB()

```

GetZPositionC
    int GDaqCalibRunInfo::GetZPositionC()

IntValueForKey
    int GDaqCalibRunInfo::IntValueForKey(char* key, int defval = -0x00000000000000001)

IsA
    TClass* GDaqCalibRunInfo::IsA()

Rpt
    static DbtRpt<GDaqCalibRunInfo>* GDaqCalibRunInfo::Rpt(char* ctx = GDaqCalibRun-
    Info::MetaRctx)

Save
    void GDaqCalibRunInfo::Save()

SetAdNo
    void GDaqCalibRunInfo::SetAdNo(int AdNo)

SetDetectorId
    void GDaqCalibRunInfo::SetDetectorId(int DetectorId)

SetDuration
    void GDaqCalibRunInfo::SetDuration(int Duration)

SetHomeA
    void GDaqCalibRunInfo::SetHomeA(int HomeA)

SetHomeB
    void GDaqCalibRunInfo::SetHomeB(int HomeB)

SetHomeC
    void GDaqCalibRunInfo::SetHomeC(int HomeC)

SetLedFreq
    void GDaqCalibRunInfo::SetLedFreq(int LedFreq)

SetLedNumber1
    void GDaqCalibRunInfo::SetLedNumber1(int LedNumber1)

SetLedNumber2
    void GDaqCalibRunInfo::SetLedNumber2(int LedNumber2)

SetLedPulseSep
    void GDaqCalibRunInfo::SetLedPulseSep(int LedPulseSep)

SetLedVoltage1
    void GDaqCalibRunInfo::SetLedVoltage1(int LedVoltage1)

SetLedVoltage2
    void GDaqCalibRunInfo::SetLedVoltage2(int LedVoltage2)

SetLtbMode
    void GDaqCalibRunInfo::SetLtbMode(int LtbMode)

SetRunNo
    void GDaqCalibRunInfo::SetRunNo(int RunNo)

SetSourceIdA
    void GDaqCalibRunInfo::SetSourceIdA(int SourceIdA)

SetSourceIdB
    void GDaqCalibRunInfo::SetSourceIdB(int SourceIdB)

```

**SetSourceIdC**

```
void GDaqCalibRunInfo::SetSourceIdC(int SourceIdC)
```

**SetZPositionA**

```
void GDaqCalibRunInfo::SetZPositionA(int ZPositionA)
```

**SetZPositionB**

```
void GDaqCalibRunInfo::SetZPositionB(int ZPositionB)
```

**SetZPositionC**

```
void GDaqCalibRunInfo::SetZPositionC(int ZPositionC)
```

**ShowMembers**

```
void GDaqCalibRunInfo::ShowMembers(TMemberInspector&, char*)
```

**SpecKeys**

```
static TList* GDaqCalibRunInfo::SpecKeys()
```

**SpecList**

```
static TList* GDaqCalibRunInfo::SpecList()
```

**SpecMap**

```
static TMap* GDaqCalibRunInfo::SpecMap()
```

**Store**

```
void GDaqCalibRunInfo::Store(DbiOutRowStream& ors, DbiValidityRec* vrec)
```

**Wrt**

```
static DbkWrt<GDaqCalibRunInfo>* GDaqCalibRunInfo::Wrt(char* ctx = GDaqCalibRun-
Info::MetaWctx)
```

**adno**

```
int GDaqCalibRunInfo::GetAdNo()
```

**aggregateno**

```
int DbitableRow::GetAggregateNo()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod csv\_import** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

```
ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart,
const TimeStamp& tend)
```

```
ql> select * from CalibPmtSpecVId ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

#### **databaseLayout**

```
std::string GDaqCalibRunInfo::GetDatabaseLayout()
```

#### **detectorid**

```
int GDaqCalibRunInfo::GetDetectorId()
```

#### **digest**

```
std::string GDaqCalibRunInfo::GetDigest()
```

#### **duration**

```
int GDaqCalibRunInfo::GetDuration()
```

#### **extracondition**

```
std::string DbfTableRow::GetExtraCondition()
```

#### **fields**

```
std::string GDaqCalibRunInfo::GetFields()
```

#### **homea**

```
int GDaqCalibRunInfo::GetHomeA()
```

#### **homeb**

```
int GDaqCalibRunInfo::GetHomeB()
```

#### **homec**

```
int GDaqCalibRunInfo::GetHomeC()
```

#### **ledfreq**

```
int GDaqCalibRunInfo::GetLedFreq()
```

#### **lednumber1**

```
int GDaqCalibRunInfo::GetLedNumber1()
```

#### **lednumber2**

```
int GDaqCalibRunInfo::GetLedNumber2()
```

#### **ledpulsesep**

```
int GDaqCalibRunInfo::GetLedPulseSep()
```

#### **ledvoltage1**

```
int GDaqCalibRunInfo::GetLedVoltage1()
```

#### **ledvoltage2**

```
int GDaqCalibRunInfo::GetLedVoltage2()
```

#### **ltbmode**

```
int GDaqCalibRunInfo::GetLtbMode()
```

**name**  
std::string GDaqCalibRunInfo::name()

**runno**  
int GDaqCalibRunInfo::GetRunNo()

**sourceida**  
int GDaqCalibRunInfo::GetSourceIdA()

**sourceidb**  
int GDaqCalibRunInfo::GetSourceIdB()

**sourceidc**  
int GDaqCalibRunInfo::GetSourceIdC()

**tabledescr**  
static std::string GDaqCalibRunInfo::GetTableDescr(char\* alternateName = 0)

**tableproxy**  
static DbTableProxy& GDaqCalibRunInfo::GetTableProxy(char\* alternateName = 0)

**values**  
std::string GDaqCalibRunInfo::GetValues()

**zpositiona**  
int GDaqCalibRunInfo::GetZPositionA()

**zpositionb**  
int GDaqCalibRunInfo::GetZPositionB()

**zpositionc**  
int GDaqCalibRunInfo::GetZPositionC()

### 23.7.34 DybDbi.GDaqRawDataFileInfo

**class** DybDbi.GDaqRawDataFileInfo (*int RunNo, int FileNo, string FileName, string StreamType, string Stream, string FileState, int FileSize, string CheckSum, string TransferState*)

Bases: DybDbi.DbTableRow

docstring

GDaqRawDataFileInfo::GDaqRawDataFileInfo()      GDaqRawDataFileInfo::GDaqRawDataFileInfo(const GDaqRawDataFileInfo& from) GDaqRawDataFileInfo::GDaqRawDataFileInfo(int RunNo, int FileNo, string FileName, string StreamType, string Stream, string FileState, int FileSize, string CheckSum, string TransferState)

**AssignTimeGate**

static void GDaqRawDataFileInfo::AssignTimeGate(Int\_t seconds, char\* alternateName = 0)

**Cache**

static DbCache\* GDaqRawDataFileInfo::Cache(char\* alternateName = 0)

**CanL2Cache**

bool GDaqRawDataFileInfo::CanL2Cache()

**Close**

static void GDaqRawDataFileInfo::Close(char\* filepath = 0l)

**Compare**

bool GDaqRawDataFileInfo::Compare(const GDaqRawDataFileInfo& that)

**classmethod Create** (\*args, \*\*kwargs)

Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**CreateTableRow**

```
DbiTableRow* GDaqRawDataFileInfo::CreateTableRow()
```

**CurrentTimeGate**

```
static int GDaqRawDataFileInfo::CurrentTimeGate(char* alternateName = 0)
```

**DoubleValueForKey**

```
double GDaqRawDataFileInfo::DoubleValueForKey(char* key, double defval = -0x00000000000000001)
```

**Fill**

```
void GDaqRawDataFileInfo::Fill(DbiResultSet& rs, DbiValidityRec* vrec)
```

**FloatValueForKey**

```
float GDaqRawDataFileInfo::FloatValueForKey(char* key, float defval = -0x00000000000000001)
```

**GetChecksum**

```
std::string GDaqRawDataFileInfo::GetChecksum()
```

**GetDatabaseLayout**

```
std::string GDaqRawDataFileInfo::GetDatabaseLayout()
```

**GetDigest**

```
std::string GDaqRawDataFileInfo::GetDigest()
```

**GetFields**

```
std::string GDaqRawDataFileInfo::GetFields()
```

**GetFileName**

```
std::string GDaqRawDataFileInfo::GetFileName()
```

**GetFileNo**

```
int GDaqRawDataFileInfo::GetFileNo()
```

**GetFileSize**

```
int GDaqRawDataFileInfo::GetFileSize()
```

**GetFileState**

```
std::string GDaqRawDataFileInfo::GetFileState()
```

**GetRunNo**

```
int GDaqRawDataFileInfo::GetRunNo()
```

**GetStream**

```
std::string GDaqRawDataFileInfo::GetStream()
```

**GetStreamType**

```
std::string GDaqRawDataFileInfo::GetStreamType()
```

**GetTableDescr**

```
static std::string GDaqRawDataFileInfo::GetTableDescr(char* alternateName = 0)
```

**GetTableProxy**

```
static DbiTableProxy& GDaqRawDataFileInfo::GetTableProxy(char* alternateName = 0)
```

**GetTransferState**

```
std::string GDaqRawDataFileInfo::GetTransferState()
```

**GetValues**

```
std::string GDaqRawDataFileInfo::GetValues()
```

**IntValueForKey**  
int GDaqRawDataFileInfo::IntValueForKey(char\* key, int defval = -0x0000000000000001)

**IsA**  
TClass\* GDaqRawDataFileInfo::IsA()

**Rpt**  
static DbIRpt<GDaqRawDataFileInfo>\* GDaqRawDataFileInfo::Rpt(char\* ctx = GDaqRawDataFileInfo::MetaRctx)

**Save**  
void GDaqRawDataFileInfo::Save()

**SetChecksum**  
void GDaqRawDataFileInfo::SetChecksum(string CheckSum)

**SetFileName**  
void GDaqRawDataFileInfo::SetFileName(string FileName)

**SetFileNo**  
void GDaqRawDataFileInfo::SetFileNo(int FileNo)

**SetFileSize**  
void GDaqRawDataFileInfo::SetFileSize(int FileSize)

**SetFileState**  
void GDaqRawDataFileInfo::SetFileState(string FileState)

**SetRunNo**  
void GDaqRawDataFileInfo::SetRunNo(int RunNo)

**SetStream**  
void GDaqRawDataFileInfo::SetStream(string Stream)

**SetStreamType**  
void GDaqRawDataFileInfo::SetStreamType(string StreamType)

**SetTransferState**  
void GDaqRawDataFileInfo::SetTransferState(string TransferState)

**ShowMembers**  
void GDaqRawDataFileInfo::ShowMembers(TMemberInspector&, char\*)

**SpecKeys**  
static TList\* GDaqRawDataFileInfo::SpecKeys()

**SpecList**  
static TList\* GDaqRawDataFileInfo::SpecList()

**SpecMap**  
static TMap\* GDaqRawDataFileInfo::SpecMap()

**Store**  
void GDaqRawDataFileInfo::Store(DbiOutRowStream& ors, DbiValidityRec\* vrec)

**Wrt**  
static DbIWrt<GDaqRawDataFileInfo>\* GDaqRawDataFileInfo::Wrt(char\* ctx = GDaqRawDataFileInfo::MetaWctx)

**aggregateno**  
int DbitableRow::GetAggregateNo()

**checksum**  
std::string GDaqRawDataFileInfo::GetChecksum()

**classmethod `csv_check`** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITERROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod `csv_compare`** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod `csv_export`** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod `csv_import`** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp& tend)

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+
-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1 |
10 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaseLayout**

std::string GDaqRawDataFileInfo::GetDatabaseLayout()

**digest**

std::string GDaqRawDataFileInfo::GetDigest()

**extracondition**

std::string DbTableRow::GetExtraCondition()

**fields**

std::string GDaqRawDataFileInfo::GetFields()

**filename**

std::string GDaqRawDataFileInfo::GetFileName()

**fileno**

int GDaqRawDataFileInfo::GetFileNo()

**filesize**

int GDaqRawDataFileInfo::GetFileSize()

**filestate**  
std::string GDaqRawDataFileInfo::GetFileState()

**name**  
std::string GDaqRawDataFileInfo::name()

**runno**  
int GDaqRawDataFileInfo::GetRunNo()

**stream**  
std::string GDaqRawDataFileInfo::GetStream()

**streamtype**  
std::string GDaqRawDataFileInfo::GetStreamType()

**tabledescr**  
static std::string GDaqRawDataFileInfo::GetTableDescr(char\* alternateName = 0)

**tableproxy**  
static DbTableProxy& GDaqRawDataFileInfo::GetTableProxy(char\* alternateName = 0)

**transferstate**  
std::string GDaqRawDataFileInfo::GetTransferState()

**values**  
std::string GDaqRawDataFileInfo::GetValues()

### 23.7.35 DybDbi.GDbiLogEntry

**class DybDbi.GDbiLogEntry**  
Bases: genDbi.DbiLogEntry

GDbiLogEntry::GDbiLogEntry()

**Cache**  
static DbCache\* GDbiLogEntry::Cache(char\* alternateName = 0)

**Close**  
static void GDbiLogEntry::Close(char\* filepath = 0l)

**classmethod Create** (\*args, \*\*kwargs)  
Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**CreateTableRow**  
DbTableRow\* GDbiLogEntry::CreateTableRow()

**DoubleValueForKey**  
double GDbiLogEntry::DoubleValueForKey(char\* key, double defval = -0x00000000000000001)

**FloatValueForKey**  
float GDbiLogEntry::FloatValueForKey(char\* key, float defval = -0x00000000000000001)

**GetDigest**  
std::string GDbiLogEntry::GetDigest()

**GetFields**  
std::string GDbiLogEntry::GetFields()

**GetTableProxy**  
static DbTableProxy& GDbiLogEntry::GetTableProxy(char\* alternateName = 0)

**GetValues**

```
std::string GDbiLogEntry::GetValues()
```

**IntValueForKey**

```
int GDbiLogEntry::IntValueForKey(char* key, int defval = -0x00000000000000001)
```

**IsA**

```
TClass* GDbiLogEntry::IsA()
```

**Rpt**

```
static Dbirpt<GDbiLogEntry>* GDbiLogEntry::Rpt(char* ctx = GDbiLogEntry::MetaRctx)
```

**Save**

```
void GDbiLogEntry::Save()
```

**ShowMembers**

```
void GDbiLogEntry::ShowMembers(TMemberInspector&, char*)
```

**Wrt**

```
static Dbirpt<GDbiLogEntry>* GDbiLogEntry::Wrt(char* ctx = GDbiLogEntry::MetaWctx)
```

**aggregateno**

```
int DbilogEntry::GetAggregateNo()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPulse")
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod csv\_import** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

```
ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart,
const TimeStamp& tend)
```

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1 |
10 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaselayout**`std::string DbILogEntry::GetDatabaseLayout()`**digest**`std::string GDbILogEntry::GetDigest()`**extracondition**`std::string DbitableRow::GetExtraCondition()`**fields**`std::string GDbILogEntry::GetFields()`**hostname**`std::string& DbILogEntry::GetHostName()`**lognumseqno**`int DbILogEntry::GetLogNumSeqNo()`**logseqnomax**`int DbILogEntry::GetLogSeqNoMax()`**logseqnomin**`int DbILogEntry::GetLogSeqNoMin()`**logtablename**`std::string& DbILogEntry::GetLogTableName()`**name**`std::string GDbILogEntry::name()`**processname**`std::string& DbILogEntry::GetProcessName()`**reason**`std::string& DbILogEntry::GetReason()`**servername**`std::string& DbILogEntry::GetServerName()`**simmask**`int DbILogEntry::GetSimMask()`**sitemask**`int DbILogEntry::GetSiteMask()`**subsite**`int DbILogEntry::GetSubSite()`**tableproxy**`static DbitableProxy& GDbILogEntry::GetTableProxy(char* alternateName = 0)`**task**`int DbILogEntry::GetTask()`**updatetime**`TimeStamp DbILogEntry::GetUpdateTime()`**username**`std::string& DbILogEntry::GetUserName()`

**values**

```
std::string GDbILogEntry::GetValues()
```

**23.7.36 DybDbi.GDcsAdTemp**

```
class DybDbi.GDcsAdTemp(float Temp1, float Temp2, float Temp3, float Temp4)
```

```
Bases: DybDbi.DbiTableRow
```

AD Temperature monitoring table:

```
mysql> 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

```
GDcsAdTemp::GDcsAdTemp() GDcsAdTemp::GDcsAdTemp(const GDcsAdTemp& from) GDc-
sAdTemp::GDcsAdTemp(float Temp1, float Temp2, float Temp3, float Temp4)
```

**AssignTimeGate**

```
static void GDcsAdTemp::AssignTimeGate(Int_t seconds, char* alternateName = 0)
```

**Cache**

```
static DbiCache* GDcsAdTemp::Cache(char* alternateName = 0)
```

**CanL2Cache**

```
bool GDcsAdTemp::CanL2Cache()
```

**Close**

```
static void GDcsAdTemp::Close(char* filepath = 0l)
```

**Compare**

```
bool GDcsAdTemp::Compare(const GDcsAdTemp& that)
```

**classmethod Create (\*args, \*\*kwargs)**

Provide pythonic instance creation classmethod:

```
i = GTableName.Create( AttributeName=100. , ... )
```

**CreateTableRow**

```
DbiTableRow* GDcsAdTemp::CreateTableRow()
```

**CurrentTimeGate**

```
static int GDcsAdTemp::CurrentTimeGate(char* alternateName = 0)
```

**DoubleValueForKey**

```
double GDcsAdTemp::DoubleValueForKey(char* key, double defval = -0x00000000000000001)
```

**Fill**

```
void GDcsAdTemp::Fill(DbiResultSet& rs, DbiValidityRec* vrec)
```

**FloatValueForKey**  
float GDcsAdTemp::FloatValueForKey(char\* key, float defval = -0x0000000000000001)

**GetDatabaseLayout**  
std::string GDcsAdTemp::GetDatabaseLayout()

**GetDigest**  
std::string GDcsAdTemp::GetDigest()

**GetFields**  
std::string GDcsAdTemp::GetFields()

**GetTableDescr**  
static std::string GDcsAdTemp::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
static DbiParamProxy& GDcsAdTemp::GetTableProxy(char\* alternateName = 0)

**GetTemp1**  
float GDcsAdTemp::GetTemp1()

**GetTemp2**  
float GDcsAdTemp::GetTemp2()

**GetTemp3**  
float GDcsAdTemp::GetTemp3()

**GetTemp4**  
float GDcsAdTemp::GetTemp4()

**GetValues**  
std::string GDcsAdTemp::GetValues()

**IntValueForKey**  
int GDcsAdTemp::IntValueForKey(char\* key, int defval = -0x0000000000000001)

**IsA**  
TClass\* GDcsAdTemp::IsA()

**Rpt**  
static DbiParamRpt<GDcsAdTemp>\* GDcsAdTemp::Rpt(char\* ctx = GDcsAdTemp::MetaRctx)

**Save**  
void GDcsAdTemp::Save()

**SetTemp1**  
void GDcsAdTemp::SetTemp1(float Temp1)

**SetTemp2**  
void GDcsAdTemp::SetTemp2(float Temp2)

**SetTemp3**  
void GDcsAdTemp::SetTemp3(float Temp3)

**SetTemp4**  
void GDcsAdTemp::SetTemp4(float Temp4)

**ShowMembers**  
void GDcsAdTemp::ShowMembers(TMemberInspector&, char\*)

**SpecKeys**  
static TList\* GDcsAdTemp::SpecKeys()

**SpecList**

```
static TList* GDcsAdTemp::SpecList()
```

**SpecMap**

```
static TMap* GDcsAdTemp::SpecMap()
```

**Store**

```
void GDcsAdTemp::Store(DbiOutRowStream& ors, DbiValidityRec* vrec)
```

**Wrt**

```
static DbiWrt<GDcsAdTemp>* GDcsAdTemp::Wrt(char* ctx = GDcsAdTemp::MetaWctx)
```

**aggregateno**

```
int DbiTableRow::GetAggregateNo()
```

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITERROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod csv\_import** (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

```
ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart,
const TimeStamp& tend)
```

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1 |
10 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaselayout**

```
std::string GDcsAdTemp::GetDatabaseLayout()
```

**digest**

```
std::string GDcsAdTemp::GetDigest()
```

```

extracondition
    std::string GDcsAdTemp::GetExtraCondition()

fields
    std::string GDcsAdTemp::GetFields()

name
    std::string GDcsAdTemp::name()

tabledescr
    static std::string GDcsAdTemp::GetTableDescr(char* alternateName = 0)

tableproxy
    static DbTableProxy& GDcsAdTemp::GetTableProxy(char* alternateName = 0)

temp1
    float GDcsAdTemp::GetTemp1()

temp2
    float GDcsAdTemp::GetTemp2()

temp3
    float GDcsAdTemp::GetTemp3()

temp4
    float GDcsAdTemp::GetTemp4()

values
    std::string GDcsAdTemp::GetValues()
    
```

### 23.7.37 DybDbi.GDcsPmtHv

**class** DybDbi.GDcsPmtHv (*int Ladder, int Column, int Ring, float Voltage, int Pw*)

Bases: DybDbi.DbTableRow

PMT High Voltage monitoring table:

```

mysql> 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)
    
```

```

GDcsPmtHv::GDcsPmtHv()      GDcsPmtHv::GDcsPmtHv(const GDcsPmtHv& from)
GDcsPmtHv::GDcsPmtHv(int Ladder, int Column, int Ring, float Voltage, int Pw)
    
```

**AssignTimeGate**

```
static void GDcsPmtHv::AssignTimeGate(Int_t seconds, char* alternateName = 0)
```

**Cache**

```
static DbCache* GDcsPmtHv::Cache(char* alternateName = 0)
```

**CanL2Cache**  
 bool GDcsPmtHv::CanL2Cache()

**Close**  
 static void GDcsPmtHv::Close(char\* filepath = 0l)

**Compare**  
 bool GDcsPmtHv::Compare(const GDcsPmtHv& that)

**classmethod Create** (\*args, \*\*kwargs)  
 Provide pythonic instance creation classmethod:  
 i = GTableName.Create( AttributeName=100. , ... )

**CreateTableRow**  
 DbTableRow\* GDcsPmtHv::CreateTableRow()

**CurrentTimeGate**  
 static int GDcsPmtHv::CurrentTimeGate(char\* alternateName = 0)

**DoubleValueForKey**  
 double GDcsPmtHv::DoubleValueForKey(char\* key, double defval = -0x00000000000000001)

**Fill**  
 void GDcsPmtHv::Fill(DbResultSet& rs, DbValidityRec\* vrec)

**FloatValueForKey**  
 float GDcsPmtHv::FloatValueForKey(char\* key, float defval = -0x00000000000000001)

**GetColumn**  
 int GDcsPmtHv::GetColumn()

**GetDatabaseLayout**  
 std::string GDcsPmtHv::GetDatabaseLayout()

**GetDigest**  
 std::string GDcsPmtHv::GetDigest()

**GetFields**  
 std::string GDcsPmtHv::GetFields()

**GetLadder**  
 int GDcsPmtHv::GetLadder()

**GetPw**  
 int GDcsPmtHv::GetPw()

**GetRing**  
 int GDcsPmtHv::GetRing()

**GetTableDescr**  
 static std::string GDcsPmtHv::GetTableDescr(char\* alternateName = 0)

**GetTableProxy**  
 static DbTableProxy& GDcsPmtHv::GetTableProxy(char\* alternateName = 0)

**GetValues**  
 std::string GDcsPmtHv::GetValues()

**GetVoltage**  
 float GDcsPmtHv::GetVoltage()

**IntValueForKey**  
 int GDcsPmtHv::IntValueForKey(char\* key, int defval = -0x00000000000000001)

**IsA**

TClass\* GDcsPmtHv::IsA()

**Rpt**

static DbIRpt<GDcsPmtHv>\* GDcsPmtHv::Rpt(char\* ctx = GDcsPmtHv::MetaRctx)

**Save**

void GDcsPmtHv::Save()

**SetColumn**

void GDcsPmtHv::SetColumn(int Column)

**SetLadder**

void GDcsPmtHv::SetLadder(int Ladder)

**SetPw**

void GDcsPmtHv::SetPw(int Pw)

**SetRing**

void GDcsPmtHv::SetRing(int Ring)

**SetVoltage**

void GDcsPmtHv::SetVoltage(float Voltage)

**ShowMembers**

void GDcsPmtHv::ShowMembers(TMemberInspector&, char\*)

**SpecKeys**

static TList\* GDcsPmtHv::SpecKeys()

**SpecList**

static TList\* GDcsPmtHv::SpecList()

**SpecMap**

static TMap\* GDcsPmtHv::SpecMap()

**Store**

void GDcsPmtHv::Store(DbiOutRowStream& ors, DbValidityRec\* vrec)

**Wrt**

static DbIWrt<GDcsPmtHv>\* GDcsPmtHv::Wrt(char\* ctx = GDcsPmtHv::MetaWctx)

**aggregateno**

int DbITableRow::GetAggregateNo()

**column**

int GDcsPmtHv::GetColumn()

**classmethod csv\_check** (*path*, *\*\*kwargs*)

Check the validity of CSV file and correspondence with CSV fields and DBI attributes:

```
from DybDbi import GCalibPmtSpec
GCalibPmtSpec.csv_check( "$DBWRITEROOT/share/DYB_%s_AD1.txt" % "SAB", afterPulse="AfterPuls
```

Manual mapping is required if field names do not match DBI attribute names (primitive case insensitive auto mapping is applied to avoid the need for tedious full mapping).

**classmethod csv\_compare** (*path*, *\*\*kwargs*)

compare entries in CSV file with those found in DB

**classmethod csv\_export** (*path*, *\*\*kwargs*)

Export the result of a default context DBI query as a CSV file

**Parameters**

- **path** – path of output file
- **fieldnames** – optionally specify the field order with a list of fieldnames

---

**Note:** make the output more human readable with regular column widths

---

**classmethod** `csv_import` (*path*, *\*\*kwargs*)

Import CSV file into Database Using default writer context for now

ContextRange::ContextRange(const int siteMask, const int simMask, const TimeStamp& tstart, const TimeStamp& tend)

```
ql> select * from CalibPmtSpecVld ; +-----+-----+-----+-----+
+-----+-----+-----+-----+ | SEQNO | TIMESTART | TIMEEND
| SITEMASK | SIMMASK | SUBSITE | TASK | AGGREGATENO | VERSIONDATE | INSERT-
DATE | +-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+ | 26 | 2011-01-22 08:15:17 | 2020-12-30 16:00:00 | 127 | 1 | 0 | 0 | -1 |
2011-01-22 08:15:17 | 2011-02-25 08:10:15 | 18 | 2010-06-21 07:49:24 | 2038-01-19 03:14:07 | 32 | 1 | 1
| 0 | -1 | 2010-06-21 15:50:24 | 2010-07-19 12:49:29 |
```

HMM... Better to make this a classmethod on the writer rather than the Row class... OR do not shrinkwrap .. just leave as example

**databaselayout**

std::string GDcsPmtHv::GetDatabaseLayout()

**digest**

std::string GDcsPmtHv::GetDigest()

**extracondition**

std::string DbiTableRow::GetExtraCondition()

**fields**

std::string GDcsPmtHv::GetFields()

**ladder**

int GDcsPmtHv::GetLadder()

**name**

std::string GDcsPmtHv::name()

**pw**

int GDcsPmtHv::GetPw()

**ring**

int GDcsPmtHv::GetRing()

**tabledescr**

static std::string GDcsPmtHv::GetTableDescr(char\* alternateName = 0)

**tableproxy**

static DbiTableProxy& GDcsPmtHv::GetTableProxy(char\* alternateName = 0)

**values**

std::string GDcsPmtHv::GetValues()

**voltage**

float GDcsPmtHv::GetVoltage()

## 23.8 DybPython

turns python/DybPython/ into a Python package

## 23.9 DybPython.Control

**class** `DybPython.Control.NuWa`

This is the main program to run NuWa offline jobs.

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

Usage:

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

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:

- 1.`configure(argv=[])` - if exists, executed at configuration time
- 2.`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 specifically specified in any input-stream map.

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

In addition to the command line, arguments can be given in a text file with one line per argument. This file can then be given to `nuwa.py` on the command line prefaced with an '@' or a '+'.  
Create a NuWa instance.

**add\_input\_file** (*fname*)

Add file name or list of file names to `self.input_files`, expanding if it is a `.list` file.

**add\_service\_redirect** (*alias, name*)

Make `alias` an alias for given service. Should be called during configuration only

**cmdline** (*argv*)

Parse command line

**configure\_args** ()

spin over all non-option arguments

**configure\_dbconf** ()

Existence of `DBCONF` envvar is used as a signal to switch between Static and DB services, so pull it out separate for clarity

**configure\_dbi** ()

For motivation for `DbiSvc` level configuration, see [dybsvn:ticket:842](#)

**configure\_dyb\_services** ()  
Configure common Dyb services

**configure\_framework** ()  
Set up framework level defaults

**configure\_ipython** ()  
If ipython not available or are already inside ipython, setup a dummy embedded ipython ipshell function, otherwise setup the real thing.

**configure\_mod** (*modname*, *modargs=None*)  
Configure this module, add to job

**configure\_optmods** ()  
load and configure() “-m” modules here

**configure\_python\_features** ()  
Set up python features

**configure\_visualization** ()  
Configure for “quanjing/panoramix” visualization

**known\_input\_type** (*fname*)  
Return True if file name has a recognized extension.

**run\_post\_user** (*appMgr*)  
Run time addition of Python Algs so they are in correct module-order

## 23.10 DybPython.dbicnf

An example using commandline parsing and pattern match against filenames, allowing smart DBI writer scripts to be created that minimize code duplication.

However make sure that arguments used are still captured into the repository either by creating one line scripts that invoke the flexible scripts. Or arranging for flexible scripts to read driver files.

```
class DybPython.dbicnf.DbiCnf (*args, **kwa)
    Bases: dict
```

DbiCnf is a dict holding parameters that are inputs to defining the DBI writer and ingredients like contextrange etc..

All outputs of this class such as `timestart`, `cr` etc.. are implemented as dynamically invoked properties, meaning that the only important state held is in this dict in the form of raw python types : str, int, datetime.

This dict is composed with class defaults, ctor arguments, commandline parsed results, path parameter regular expression parsed tokens, interactive updating.

Precedence in decreasing order:

- 1.commandline arguments
- 2.after ctor updates
- 3.ctor keyword arguments
- 4.basis defaults in `DbiCnf.defaults`

Usage in writer scripts:

```

from DybPython import DbiCnf
cnf = DbiCnf()
cnf()          ## performs the parse

from DybDbi import GCalibPmtSpec, CSV
wrt = cnf.writer( GCalibPmtSpec )

src = CSV( cnf.path )
for r in src:
    instance = GCalibPmtSpec.Create( **r )
    wrt.Write( instance )

if not cnf.dummy:
    assert wrt.close()

```

Debugging/checking usage in ipython:

```

from DybPython import DbiCnf
cnf = DbiCnf(key=val,key2=val2)
cnf['key3'] = 'val3'

cnf()      ## performs command line parse
cnf("All_AD1_Data.csv --task 20 --runtimestart 10 --dbconf tmp_offline_db:offline_db ")  ## tes
print cnf
cnf['runtimestart'] = 10
cnf.timestart
cnf['runtimestart'] = 1000
cnf.timestart          ## will do timestart lookup for the changed run

```

The simplest and recommended usage is to define a standard *.csv* file naming convention. For example when using the default context pattern:

```
"^(?P<site>All|DayaBay|Far|LingAo|Mid|SAB)_(?P<subsite>AD1|AD2|AD3|AD4|All|IWS|OWS|RPC|Unknown)_"
```

The tokens *site*, *subsite* and *simflag* are extracted from basenames such as the below by the pattern matching.

- 1.SAB\_AD1\_Data.csv
- 2.SAB\_AD2\_Data.csv

**Parameters** *kwa* – ctor keyword arguments override class defaults `DbiCnf.defaults` updating into *self*

**cr**

Convert the strings into enum value, and datetimes into TimeStamps in order to create the ContextRange instance

**Returns** context range instance

**logging\_** (*args*)

Hmm need some work ...

**parse\_path** (*path\_*, *ptn*, *nomatch*)

Extract context metadata from the path using the regular expression string supplied.

**Parameters**

- **path** – path to *.csv* source file
- **ptn** – regular expression string that can contain tokens for any config parameters

**Rtype dict** dict of strings extracted from the path

**simflag**

Convert string simflag into enum integer

**simmask**

Convert string simflag into enum integer (note the simflag is interpreted as the mask)

**site**

Convert string site into enum integer

**sitemask**

Convert string site into enum integer if multi-site masks are needed will have to revisit this

**subsite**

Convert string subsite/DetectorId into enum integer

**timeend**

**timestart**

**writer** (*kls*)

Create a pre-configured DybDbi writer based on arguments and source csv filename parsing and creates the corresponding DB table if it does not exist.

**Parameters** *kls* – DybDbi class, eg GCalibPmtHighGain

**class** DybPython.dbicnf.**TimeAction** (*option\_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: argparse.Action

Converts string date representations into datetimes

## 23.11 DbiDataSvc

### 23.11.1 DbiDataSvc

## 23.12 NonDbi

### 23.12.1 NonDbi

#### SQLAlchemy Ecosystem

Requirements, the currently non-standard SQLAlchemy external, install with:

```
./dybinst trunk external SQLAlchemy
```

After installation many examples are available at:

```
external/build/LCG/SQLAlchemy-0.6.7/examples/
```

Reading from DB `dybgaudi:Database/NonDbi/tests/read.py`:

```
from NonDbi import session_, Movie, Director
session = session_("tmp_offline_db", echo=False)
for m in session.query(Movie).all():
    print m
```

Writing to DB dybgaudi:Database/NonDbi/tests/write.py:

```
from NonDbi import session_, Movie, Director
session = session_("tmp_offline_db", echo=False)
m1 = Movie("Star Trek", 2009)
m1.director = Director("JJ Abrams")
d2 = Director("George Lucas")
d2.movies = [Movie("Star Wars", 1977), Movie("THX 1138", 1971)]
try:
    session.add(m1)
    session.add(d2)
    session.commit()
except:
    session.rollback()
```

### Deficiencies

Problems with multiple sessions, may need rearrangement

- <http://www.sqlalchemy.org/docs/orm/session.html#session-frequently-asked-questions>

### Accessing Non Dbi tables with SQLAlchemy

The `kls_` method on the SQLAlchemy `session` returns an SQLAlchemy class mapped to the specified table. Usage:

```
from NonDbi import session_
s = session_("fake_dcs")
kls = s.kls_("DBNS_SAB_TEMP")
n = s.query(kls).count()
```

### Accessing DBI pairs with SQLAlchemy

The `dbikls_` method on the SQLAlchemy `session` has been shoe-horned in using some esoteric python. It returns an SQLAlchemy class mapped to the join of payload and validity tables. Usage:

```
from NonDbi import session_
session = session_("tmp_offline_db")
YReactor = session.dbikls_("Reactor")

# Use dynamic class in standard SQLAlchemy ORM manner
n = session.query(YReactor).count()
a = session.query(YReactor).filter(YReactor.SEQNO==1).one()
print vars(a)          ## instances of the class have all payload and validity attributes
```

Esotericness includes : closures, dynamic addition of instance methods and dynamic class generation. The advantage of this approach is that there are no static ".spec" or declarative table definitions, everything is dynamically created from the Database schema. This dynamism is also a disadvantage as the static files can be useful places for adding functionality.

### Reference for SQLAlchemy querying

- <http://www.sqlalchemy.org/docs/orm/tutorial.html#querying>

## How to add a class/table

1. follow patten of examples in `movie.py` and `director.py`
2. import the declarative classes into `__init__` of **NonDbi**
3. write tests to check functionality

## References

### Declarative SQLAlchemy

- <http://www.sqlalchemy.org/docs/orm/tutorial.html#creating-table-class-and-mapper-all-at-once-declaratively>

### Hierarchy using self referential one-to-many:

- <http://www.sqlalchemy.org/docs/orm/relationships.html#adjacency-list-relationships>

For a self-contained script to quickstart model prototyping see :

- <http://www.blog.pythonlibrary.org/2010/02/03/another-step-by-step-sqlalchemy-tutorial-part-2-of-2/>

## SQLite tips

SQLite is useful for quick tests without need to connect to a remote DB, the DB lives inside a file or even in memory:

```
sqlite3 tutorial.db
SQLite version 3.3.6
Enter ".help" for instructions
sqlite> .tables
addresses  users
sqlite> .help
.databases          List names and files of attached databases
.dump ?TABLE? ...  Dump the database in an SQL text format
.echo ON|OFF        Turn command echo on or off
.exit               Exit this program
...

sqlite> .schema addresses
CREATE TABLE addresses (
  id INTEGER NOT NULL,
  email_address VARCHAR NOT NULL,
  user_id INTEGER,
  PRIMARY KEY (id),
  FOREIGN KEY(user_id) REFERENCES users (id)
);
```

## Implementation Notes

Try adopting SA split model layout promulgated at

- [http://docs.pylonsproject.org/projects/pyramid\\_cookbook/dev/sqla.html](http://docs.pylonsproject.org/projects/pyramid_cookbook/dev/sqla.html)
- <http://blogs.symora.com/nmishra/2010/02/28/configure-pylons-with-sqlalchemy-and-separate-files-for-models/>

With motivation:

1. keep model classes in separate files

```
class NonDbi.MetaDB(dbconf=None)
```

```
Bases: object
```

Create one *MetaDB* instance per database connection , usage:

```
off_ = MetaDB("tmp_offline_db")
off = off_()                                ## call to pull up a session

daq_ = MetaDB("tmp_daqdb")
daq = daq_()

YCableMap = off_.dbikls_("CableMap")        ## NB now on the MetaDB instance rather than the session
print off.query(YCableMap).count()

YSTF = daq_.kls_("SFO_TZ_FILE")
print daq.query(YSTF).count()
```

No need to diddle with the session kls this way, although could if decide to get sugary.

The initial *session\_* approach has difficulties when dealing with multiple DB/sessions, multiple `Session.configure` causes warnings

The contortions were caused by:

1. sharing metadata with declarative base ?
2. having a single vehicle on which to plant API (the session)

Try again unencumbered by declarative base compatibility and the meta module

```
session()
```

Binding is deferred until the last moment

```
NonDbi.cfg_(sect, path='~/my.cnf')
```

Provide a dict of config paramertes in section *sect*

```
NonDbi.dj_init_(dbconf='tmp_offline_db', djapp='NonDbi.dj.dataset')
```

Check Django compatibility by trying to use it to talk to the SQLAlchemy generated model

```
NonDbi.engine_(dbconf='tmp_offline_db', echo=False)
```

Creates SQLAlchemy engine for *dbconf*, usage:

```
from NonDbi import engine_
engine = engine_("tmp_offline_db")
print engine.table_names()
```

```
NonDbi.session_(dbconf='tmp_offline_db', echo=False, drop_all=False, drop_some=[], create=False)
```

Creates SQLAlchemy connection to DB and drops and/or creates all tables from the active declarative models  
Returns *session* through which DB can be queries or updates

#### Parameters

- **dbconf** – section in `~/my.cnf` with DB connection parameters
- **echo** – emit the SQL commands being performed
- **drop\_all** – drop all active NonDbi tables **CAUTION: ALL TABLES**
- **drop\_some** – drop tables corresponding to listed mapped classes
- **create** – create all tables if not existing

SQLAlchemy innards are managed in the `meta` module

## 23.13 Scraper

In addition to this API reference documentation, see the introductory documentation at *Scraping source databases into offline\_db*

- Scraper
- Table specific scraper module examples
  - Scraper.pmthv
    - \* Scraper.pmthv.PmtHv
    - \* Scraper.pmthv.PmtHvSource
    - \* Scraper.pmthv.PmtHvScraper
    - \* Scraper.pmthv.PmtHvFaker
  - Scraper.adtemp
    - \* Scraper.adtemp.AdTemp
    - \* Scraper.adtemp.AdTempSource
    - \* Scraper.adtemp.AdTempScraper
    - \* Scraper.adtemp.AdTempFaker
- Scrapers in development
  - Scraper.adlidsensor
    - \* Scraper.adlidsensor.AdLidSensor
- Scraper.dcs : source DB naming conventions
- Scraper.base : directly used/subclassed
  - Scraper.base.main()
  - Scraper.base.Regime
  - Scraper.base.DCS
  - Scraper.base.Scraper
  - Scraper.base.Target
  - Scraper.base.Faker
- Other classes used internally
  - Scraper.base.sourcevector.SourceVector
  - Scraper.base.aparser.AParser : argparser/configparser amalgam
  - Scraper.base.parser.Parser :
  - Scraper.base.sa.SA : details of SQLAlchemy connection

### 23.13.1 Scraper

Generic Scraping Introduction at *Scraping source databases into offline\_db*

### 23.13.2 Table specific scraper module examples

**Scraper.pmthv**

PMT HV scraping specialization

**Scraper.pmthv.PmtHv**

```
class Scraper.pmthv.PmtHv(*args, **kwa)
    Bases: Scraper.base.regime.Regime
```

Regime frontend class with simple prescribed interface, takes the `cfg` argument into this dict and no args in call. This allows the frontend to be entirely generic.

#### `Scraper.pmthv.PmtHvSource`

**class** `Scraper.pmthv.PmtHvSource` (*srcdb*)

Bases: `list`

**Parameters** `srcdb` – source DB instance of `Scraper.base.DCS`

List of source SA classes that map tables/joins in `srcdb` Accommodates a table naming irregularity HVPw rather than HV\_Pw

#### `Scraper.pmthv.PmtHvScraper`

**class** `Scraper.pmthv.PmtHvScraper` (*srcs, target, cfg*)

Bases: `Scraper.base.scraper.Scraper`

##### **Parameters**

- **srcs** – list of source SA classes
- **target** – *Target* instance that encapsulates the `DybDbi` class
- **cfg** – instance of relevant *Regime* subclass (which isa dict holding config)

Config options:

##### **Parameters**

- **maxiter** – maximum iterations or 0 for no limit
- **interval** – timedelta cursor step size
- **maxage** – timedelta maximum age, beyond which even an unchanged row gets written
- **sleep** – timedelta sleep between scrape update sampling

**changed** (*sv*)

**Parameters** `sv` – source vector instance `Scraper.base.sourcevector.SourceVector`

Decide if sufficient change to propagate based on differences between the first and last elements of `SourceVector` instance argument

**propagate** (*sv*)

**Parameters** `sv` – source vector instance `Scraper.base.sourcevector.SourceVector`

Yield write ready `DybDbi` target dicts to base class, note that a single source vector instance is yielding multiple target dicts. The keys of the target dict must match the specified attributes of the `DybDbi` target class.

Here the output is based entirely on the last element of the source vector. A smarter implementation might average the first and last to smooth variations. The python `yield` command makes it possible to iterate over a what is returned by a function/method.

**seed** (*sc*)

Used for seeding target DB when testing into empty tables

**Parameters** `sc` – source class, potentially different seeds will be needed for each source that feeds into a single target

**Scraper.pmthv.PmtHvFaker**

**class** Scraper.pmthv.PmtHvFaker (srcs, cfg)

Bases: Scraper.base.faker.Faker

Creates fake instances and inserts them into sourcedb

**fake** (inst, id, dt)

Invoked from base class call method, set attributes of source instance to form a fake

**Parameters**

- **inst** – source instance
- **id** – id to assign to the instance instance

**Scraper.adtemp**

AD Temperature scraping specialization

**Scraper.adtemp.AdTemp**

**class** Scraper.adtemp.AdTemp (\*args, \*\*kwa)

Bases: Scraper.base.regime.Regime

Regime frontend class with simple prescribed interface, takes the cfg argument into this dict and no args in call ... allowing the frontend to be entirely generic

**Scraper.adtemp.AdTempSource**

**class** Scraper.adtemp.AdTempSource (srcdb)

Bases: list

A list of SQLAlchemy dynamic classes

Coordinates of source table/joins

**Scraper.adtemp.AdTempScraper**

**class** Scraper.adtemp.AdTempScraper (srcs, target, cfg)

Bases: Scraper.base.scraperscraper.Scraper

Specialization of generic scraper for AD temperature tables

**Parameters**

- **srcs** – list of source SA classes
- **target** – *Target* instance that encapsulates the DybDbi class
- **cfg** – instance of relevant *Regime* subclass (which isa dict holding config)

Config options:

**Parameters**

- **maxiter** – maximum iterations or 0 for no limit
- **interval** – timedelta cursor step size

- **maxage** – timedelta maximum age, beyond which even an unchanged row gets written
- **sleep** – timedelta sleep between scrape update sampling

**changed** (*sv*)

returns changed decision to base class

Caution DB/SQLAlchemy is providing decimal.Decimal values... unify types to float before comparison to avoid surprises

**propagate** (*sv*)

yields one or more target dicts ready for writing to target DB

#### Scraper.adtemp.AdTempFaker

**class** Scraper.adtemp.**AdTempFaker** (*srcs, cfg*)

Bases: Scraper.base.faker.Faker

**fake** (*inst, id, dt*)

Invoked from base class, sets source instance attributes to form a fake

#### Parameters

- **inst** – source instance
- **id** – suggested id to use
- **dt** – suggested date\_time to use

Note the settings can not easily be done in the framework as the *inst* can represent a join of multiple tables, requiring specialized action.

## 23.13.3 Scrapers in development

### Scraper.adlidsensor

AD lid sensors scraping specialization

Discussion from Wei:

1. we were discussing scrapping the average, its standard deviation, the minimum and the maximum within each hour.
2. It seems average once per hour is sufficient. (Note: reactor flux will be available sparser than 1/hour).

reference	
<a href="#">doc:6673</a>	discussion
<a href="#">doc:6996</a>	for the current status given by David W.
<a href="#">doc:6983</a>	summarizes the lid sensor data so far.

#### Scraper.adlidsensor.AdLidSensor

**class** Scraper.adlidsensor.**AdLidSensor** (*\*args, \*\*kwa*)

Bases: Scraper.base.regime.Regime

Regime frontend class with simple prescribed interface, takes the *cfg* argument into this dict and no args in call ... allowing the frontend to be entirely generic

### 23.13.4 `Scraper.dcs` : source DB naming conventions

Encapsulation of naming conventions for tables and fields used in DCS database

### 23.13.5 `Scraper.base` : directly used/subclassed

Functions/classes subclassed or used directly by specific table scrapers

#### `Scraper.base.main()`

`Scraper.base.main()`

Scraper/Faker frontend

Parses the config into the `cfg` dict and imports, instantiates and calls the regime class identified by the `cfg`. This pattern minimises code duplication and retains flexibility.

- 1.bulk behaviour controlled via single argument pointing to section of config file `$$SCRAPER_CFG` which defines all the default settings of options
- 2.config includes which classes to import into the main and invoke... so need simple common interface for frontends in all regimes : `pmthv/adtemp`

Note that the default *section* and its settings are listed together with the option names to change these defaults by:

```
scr.py --help
scr.py -s adtemp_testscrape --help    ## show defaults for this section
scr.py -s adtemp_faker --help
```

Typical Usage:

```
scr.py -s adtemp_scraper
scr.py -s pmthv_scraper
```

```
scr.py -s adtemp_faker
scr.py -s pmthv_faker
```

During testing/development options can be added to change the behavior

The primary argument points to the section of `.scraper.cfg` which configures the details of the scrape:

```
[adtemp_scraper]

regime = Scraper.adtemp:AdTemp
kls = GDcsAdTemp
mode = scraper

source = fake_dcs
target = offline_db_dummy

interval = 10s
sleep = 3s
maxage = 10m

threshold = 1.0
maxiter = 100

dbi_loglevel = INFO
```

### Scraper.base.Regime

**class** Scraper.base.Regime (\*args, \*\*kwa)

Bases: dict

The regime class ctor takes the cfg as its sole argument, which being a dict takes the cfg into itself.

**initialize** ()

Preparations done prior to calling the regime class, including:

**setsignals** ()

signal handling following the approach of supervisor

### Scraper.base.DCS

Specialization of SA providing SQLAlchemy access to source DCS DB

**class** Scraper.base.DCS (dbconf)

Bases: Scraper.base.sa.SA

SQLAlchemy connection to database, performing table reflection and mappings from tables

Specializations:

1. standard query ordering, assuming a *date\_time* attribute in tables

**qafter** (kls, cut)

*date\_time* ordered query for instances at or beyond the time cut:

t0 t1 t2 t3 (t4 t5 t6 t7 t8 t9 ... )

#### Parameters

- **kls** – source SQLAlchemy mapped class
- **cut** – local time cutoff datetime

**qbefore** (kls, cut)

*date\_time* ordered query for instances before the cut

**subbase** (dtn)

subclass to use, that can be dependent on table coordinate

### Scraper.base.Scraper

**class** Scraper.base.Scraper (srcs, target, cfg)

Bases: Scraper.base.propagator.Propagator

Base class holding common scrape features, such as the scrape logic which assumes:

1. source instances correspond to fixed time measurement *snapshots*
2. target instances represent source measurements over time ranges
- 3.2 source instances are required to form one target instance, the target validity is derived from the datetimes of two source instances

Initialisation in *Propagator* superclass

#### Parameters

- **srcs** – list of source SA classes

- **target** – *Target* instance that encapsulates the DybDbi class
- **cfg** – instance of relevant *Regime* subclass (which isa dict holding config)

Config options:

#### Parameters

- **maxiter** – maximum iterations or 0 for no limit
- **interval** – timedelta cursor step size
- **maxage** – timedelta maximum age, beyond which even an unchanged row gets written
- **sleep** – timedelta sleep between scrape update sampling

#### **changed** (*sv*)

Override in subclasses to return if a significant change in source instances is observed. This together with age checks is used to decide if the propagate method is called.

**Parameters** *sv* – source vector containing two source instances to interrogate for changes

#### **propagate** (*sv*)

Override this method in subclasses to yield one or more write ready target dicts derived from the *sv[-1]* source instance or *sv[-1].aggd* aggregate dict

**Parameters** *sv* – source vector containing two source instances to propagate to one target write

#### **tunesleep** (*i*)

Every *self.tunesleepmod* iterations check lags behind sources and adjust sleep time accordingly. Allowing to turn up the beat in order to catchup.

Tune heuristic uses an effective heartbeat, which is the time between entries of interest to the scrapee, ie time between source updates scaled by offset+1

Only makes sense to tune after a write, as it is only then that **tcursor** gets moved ahead. When are close to current the sleep time can correspond to the timecursor **interval** when behind sleep to allow swift catchup

#### POSSIBLE ISSUES

1. if ebeatlag never gets to 0, the sleep time will sink to the minimum

(a) minimum was formerly 0.1, adjusted to max(0.5, ebeatsec/10.) out of concern for excessive querying

(b) adjusting to ebeatsec would be too conservative : would prevent catchup

### Scraper.base.Target

**class** Scraper.base.Target (\*args, \*\*kwa)

Bases: dict

Encapsulate DybDbi dealings here to avoid cluttering Scraper

Relevant config parameters

**Parameters** **timefloor** – None or a datetime or string such as ‘2010-09-18 22:57:32’ used to limit the expense of validity query

**instance** (\*\*kwa)

Might fail with TypeError if kwa cannot be coerced, eg from aggregate queries returning None when zero samples

If the attribute names are not expected for the target kls they are skipped. This will be the case for the system attributes *\_date\_time\_min* *\_date\_time\_max*

**lastvld** (*source*)

Last validity record in target database for context corresponding to *source* class. Query expense is restricted by the *timefloor*. If *timefloor* is None a sequence of progressively more expensive queries are performed to get the target last validity.

**Parameters**

- **source** – SA mapped class
- **timefloor** – time after which to look for validity entries in target database or None

Note this is called only at scraper initialization, in order for the scraper to find its time cursor.

**seed** (*srcs, scraper*)

Scraping needs an entry in the target DB in order to discern where in time the scraping is up to ... hence when testing into empty DB/Tables need to plant a seed entry using DybDbi

This is invoked at scraper instantiation when the conditions are met:

1. `seed_target_tables` is configured True
2. Commandline option `--ALLOW_DROP_CREATE_TABLE` is used

Seed entries are written to the target table. The seed validity range is configured with the options: *seed\_timestart* *seed\_timeend* and the payload entry is specified by the *def seed()* method implemented in the scraper class.

Neither of these should be used in production. Attempts to perform seeding under supervisor raises an exception, to enforce this restriction.

When testing seeding start from scratch with eg:

```
mysql> drop table DcsAdTemp, DcsAdTempVld ;
mysql> update LOCALSEQNO set LASTUSEDSEQNO=0 where TABLENAME='DcsAdTemp' ;
```

**writer** (*sv, localstart=None, localend=None*)

Prepare DybDbi writer for target class, with `contextrange/subsite` appropriate for the source instance

Use of non-default `localstart` and `localend` type is typically only used for `aggregate_group_by` quering where the instance datetimes such as `sv[0].date_time` do not correspond to the `contextrange` of the aggregate dict.

**Parameters**

- **sv** – source vector instance that contains instances of an SA mapped class
- **localstart** – default of *None* corresponds to `sv[0].date_time`
- **localend** – default of *None* corresponds to `sv[-1].date_time`

**Scraper.base.Faker**

**class** `Scraper.base.Faker` (*srcs, cfg*)

Bases: list

create fake source instances and insert them

### 23.13.6 Other classes used internally

#### Scraper.base.sourcevector.SourceVector

**class** Scraper.base.sourcevector.SourceVector (*scraper, source*)

Bases: list

This is a simply a holder for source instances and the timecursor, the action is driven by the *Scraper* instance, which invokes the `SourceVector.__call__` method to perform the sampling, ie querying for new instances at times beyond the `tcursor`

As each instance is collected the prior last instance is discarded until sufficient deviation (in age or value) between the first and last is seen. Deviation results in this *SourceVector* being collapsed to start again from the last sample. This also is driven from the *Scraper* by setting the *tcursor* property.

Manages:

- 1.0,1 or 2 source class instances
- 2.timecursor
- 3.lastresult enum from last `_sample`

Actions:

- 1.checks to see if conditions are met to propagate collected source instances into target, in `__call__` method

#### Parameters

- **scraper** – parent scraper
- **source** – SA mapped class

**iinst** (*i*)

**Parameters** *i* – instance index into source vector

**Rtype** `source` returns instance or None

**lag** ()

**Returns** `timedelta` instance representing scraper lag or None if no last entry beyond the `tcursor`

Query source to find datetime of last entry and return the time difference `last - tcursor` indicating how far behind the scraper is. This will normally be positive indicating that the scraper is behind.

It would be inefficient to do this on every iteration

**lastentry** ()

Query source to find last entry with *date\_time* greater than the timecursor When the `tcursor` is approaching the cached last entry time, need to query otherwise just use cached

**Returns** SA instance or None

**lastresult\_**

progress string representing *lastresult* enum integer

**set\_tcursor** (*tc*)

Assigning to this *sv.tcursor* not only changes the cursor but also collapses the *SourceVector* ready to collect new sampled source instances.

**smry** ()

Example:

```
SV 4    (43, 46) 2011-01-10 10:02:00 full      unchanged (10:34:28 10:34:34)
# calls  ids      tcursor      status      lastresult      times
```

Shows the status of source vector including the id and date\_time of source entries sv[0] and sv[-1]

**calls** iteration count

**ids** id of source entries

**tcursor** timecursor, stepping through time. Changes only at each propagation

**status** fullness of source vector: empty/partial/full (**full** means 2 entries)

**lastresult** possibilities: "noupdate","notfull","overage","changed","unchanged","init","lastfirst"

**times** date\_time of source entries, changes as each sample is made

**status**

enum status integer

**status\_**

status string representing *status* enum integer

**tcursor**

Assigning to this *sv.tcursor* not only changes the cursor but also collapses the SourceVector ready to collect new sampled source instances.

### Scrapper.base.aparser.AParser : argparser/configparser amalgam

**class** Scrapper.base.aparser.AParser (\*args, \*\*kwargs)

Bases: `argparse.ArgumentParser`

Primes an argparser with defaults read from a section of an ConfigParser style config file and sets up logging

Operates via 2-stage parsing

Usage:

```
parser = AParser(defpath="~/scrapper.cfg", defsect="default")
parser.add_argument( '-m', '--maxiter', help="maximum iterations, or 0 for no limit")
parser.set_defaults( maxiter=0 )
args = parser()
print args
```

Draws upon:

- <http://blog.vwelch.com/2011/04/combining-configparser-and-argparse.html>
- <http://www.doughellmann.com/PyMOTW/argparse/>

### Scrapper.base.parser.Parser :

**class** Scrapper.base.parser.Parser (\*args, \*\*kwargs)

Bases: `Scrapper.base.aparser.AParser`

To see all the available options and defaults for a particular config sections:

```
scr.py --help
scr.py -s adtemp_scraper --help
scr.py -s pmthv_scraper --help
```

Performs two stage parsing, with the first stage driven by `-s/--sect` option to specify the section name within a configuration file. The path at which a config file is read from can be controlled by `SCRAPER_CFG`, with default value:

```
echo $SCRAPER_CFG      ## path of default config file
--> $SITEROOT/dybgaudi/Database/Scraper/python/Scraper/.scraper.cfg
--> $SCRAPERROOT/python/Scraper/.scraper.cfg
```

Note that the first stage of parsing occurs in the `AParser.__init__` which:

- 1.provides config section name and path
- 2.primes the base `AParser` dict with defaults read from that section

The 2nd stage parse typically does nothing, as it is preferable to keep config at defaults read from file. This commandline control is mainly for testing/debugging.

Note the configparser/argparser mismatch in boolean handling:

- 1.argparse typically has “store\_true/store\_false” actions for convenient/brief commandline control
- 2.configparser and config file understandability requires True/False strings

Have sided with configparser as the commandline interface beyond the `-s` is mainly for developer usage. However some options, such as `-dryrun` which make little sense in config files, buck this tendency.

**classmethod `config`** (*defsect='adtemp\_scraper', defpath=None, noargs=False*)  
 Convenience classmethod for config access

## Scraper.base.sa.SA : details of SQLAlchemy connection

**class** `Scraper.base.sa.SA` (*dbconf*)  
 Bases: `object`

Manages SQLAlchemy DB connections, orchestrates reflection on tables, dynamic class creation and mapping from tables to classes. These are all done lazily, when a class is requested via `.cls(xtn)`

**cls** (*xtn*)  
 Return mapped dynamic class from a xtn instance

**reflect** (*tn*)  
 Reflect on the table, recording it in the meta

**table** (*tn*)  
 Return the sqlalchemy.schema.Table representation of a table, reflect upon the table if not already done

## 23.14 DybTest

### 23.14.1 dybtest

### 23.14.2 dybtest.histref

This provides the connector between the the Run machinery in `run.py` and `histo` comparisons in `cfroot.py`

Simply adding a `histref` argument

```
Run("nuwa.py ...." , histref="path/to/myhistname.root" )
```

with value that points to a .root file containing the histograms, switches on the comparison of histograms with reference.

Created histograms become the “blessed” reference when the above is invoked and the reference path does not exist :  
path/to/histref\_myhistname.root

Thus to bless the current histos simply delete the reference and rerun.

NB splitting up the time consuming steps and the histo creators is perfectly acceptable, as is use of python scripts or root .C histo creators (although using python modules with nuwa.py is recommended), eg:

```
def test_time_consuming_creation():
    Run("nuwa.py ..... ")

def test_quick_nuwa_ana():
    Run("nuwa.py ..." , histref="histos1.root" )
def test_quick_py_ana():
    Run("python blah.py", histref="histos2.root" )
def test_quick_root_ana():
    Run("root -b -q maker.C ", histref="histos3.root" )
```

### 23.14.3 dybtest.cfroot

Usage examples:

```
cfr = CfRoot(['ex1.root','ex2.root','ex3.root'], ['TH1F','TH2F'] )
rcr = cfr()          ## compare all correspondings histos between the files
if rcr == 0:print "consistent"
print cfr
```

Compare the last hist between the files, by accessing a list of keys:

```
cfh = CfHist(cfr[-1])
rch = cfh()
if rch == 0:print "consistent"
print cfh
```

**class** dybtest.cfroot.CfHist (keys)

Facilitate comparisons between multiple histograms specified by lists of keys. Consistency is assessed by roots KolmogorovTest with fixed cut of 0.9. To change that:

```
from dybtest.cfroot import CfHist
CfHist.kolmogorov_cut = 0.95
```

**class** dybtest.cfroot.CfRoot (paths, cls)

Facilitate comparisons between multiple root files by holding KeyList's into each of them, allowing list access to corresponding objects from all the files

Usage examples:

```
cf = CfRoot(['ex1.root','ex2.root','ex3.root'], ['TH1F','TH2F'] )
rc = cf()
print cf

for i in len(cf):
    cfi = cf[i]
```

**class** dybtest.cfroot.KeyList (path, cls)

Recursive walk the TDirectory structure inside a single TFile providing list access to all keys that hold instances of classes within the cls list or all classes if cls is an empty list

Usage examples:

```
k1 = KeyList( "path/to/histos.root" , ['TH1F', 'TH2F'] )
list(k1)
print len(k1)
for k in k1:
    print k
print k1[0], k1[-1]
```

`dybtest.cfroot.TKey_GetCoords (self)`

**provides filename, directory within the file and key name::** ['ex2.root', 'red/aa/bb/cc', 'h2', 'TH1F' ]

`dybtest.cfroot.TKey_GetIdentity (self)`

skip the file name

### 23.14.4 dybtest.capture

for `gbl.cout/gbl.stringstream` to be available/usable from python observe that must kickstart ROOT for example with “from ROOT import TObject” prior to “from GaudiPython import gbl”

When GaudiPython comes first get:

AttributeError: class `_global_cpp` has no attribute 'cout'

**class** `dybtest.capture.Capture (arg='')`

Bases: `object`

Allows capturing of logging output in a generic way, allowing tests to be made on the logging output. This can be a shortcut way of testing as functionality can be tested without exposing underlying classes to python.

Usage example:

```
from dybtest import Capture

def test_dbimaketimestamp():
    c = Capture("capture Dbi.MakeTimeStamp ... ")
    t = Dbi.MakeTimeStamp("")
    t = Dbi.MakeTimeStamp("")
    c()
    assert str(c).find("Bad date string") > -1
```

Redirect cout into contained stringstream



# DOCUMENTATION

Documenting the documentation.

## 24.1 About This Documentation

Latex sources are translated into reStructuredText <sup>1</sup> using converter <sup>2</sup>, which is used by the Sphinx <sup>3</sup> documentation generator. The html render includes an integrated search function that is OpenSearch <sup>4</sup> enabled, allowing you to search from your browsers search field in supported browsers.

---

**Note:** Use the **Show Source** link in the sidebar of every html page to get familiar with reStructuredText and Sphinx

---

### 24.1.1 Build Instructions for Sphinx based documentation

#### Who needs to build the Sphinx docs ?

The Sphinx based documentation is built automatically by the dybinst slave, thus latex source editors need not build the Sphinx docs themselves. Committed latex sources should be automatically converted at the next slave build. However usage of latex commands/environments unknown to the converter will break the build.

People using the *Autodoc : pulling reStructuredText from docstrings* feature or those wishing to make significant additions to the documentation will benefit from being able to build the documentation themselves in order to achieve the desired presentation of their docstrings.

#### Once only virtualenv setup

1. Get into nuwa environment and check that virtualenv is in your path:

```
which virtualenv          ## should be the NuWa one
```

2. Create virtual python environment, *spawned* from nuwa python eg:

```
mkdir -p ~/v  
virtualenv ~/v/docs
```

For background info on virtualenv see <http://www.virtualenv.org/en/latest/>

---

<sup>1</sup> <http://docutils.sourceforge.net/rst.html>

<sup>2</sup> <https://github.com/scb-/converter>

<sup>3</sup> <http://sphinx.pocoo.org>

<sup>4</sup> <http://www.opensearch.org>

## Installation of Sphinx and converter into virtual python

The virtualenv comes with **pip** and **easy\_install** as standard, install sphinx and converter:

```
. ~/v/docs/bin/activate          # activate the docs virtualenv
pip install sphinx
pip install -e git+git://github.com/scb-/converter.git#egg=converter
pip install -e git+git@github.com:scb-/converter.git#egg=converter    ## if you have the key
```

Several sphinx pre-requisites will be installed by pip : *Pygments*, *Jinja2* and *docutils*

## Additional dependencies

---

**Note:** dependency removed

These additional dependencies on numpy and matplotlib has been removed in order to simplify the setup of a documentation building system.

---

Additional dependencies are required for some sphinx extensions *Sandbox Testing reST/Sphinx*:

```
pip install -E ~/v/docs -e git+git://github.com/scb-/numpy.git#egg=numpy    ## my fork of numpy
pip -v install -e svn+https://matplotlib.svn.sourceforge.net/svnroot/matplotlib/trunk/matplotlib/#egg=matplotlib
```

---

## Todo

test return to numpy original, now that my changes are integrated

---

## Sphinx Quickstart/Configuration

The results of the *sphinx-quickstart* are stored in the *conf.py* and *Makefile* in the [dybgaudi:Documentation/OfflineUserManual/tex](#) directory. These have been customized, and thus the quickstart procedure should **not** be repeated.

## Building docs

Steps to build the docs:

1. Enter nuwa environment and activate the *docs* virtualpython:

```
. ~/v/docs/bin/activate
which python    ## should be ~/v/docs/bin/python
```

2. Enter the *tex* directory:

```
cd NuWa-trunk/dybgaudi/Documentation/OfflineUserManual/tex
```

3. Convert *tex* sources into *rst* and then derive *html*, *tex* and *pdf*:

```
make
```

4. Get out of the virtual python:

```
deactivate
```

5. Check the resulting documentation, at <http://daya0001.rcf.bnl.gov/oum/>

## Partial Builds

While editing documentation it is useful to perform quick partial builds in order to quickly preview changes to parts of the document. Do so using non-default *make* targets such as *api* and *sop*.

## Possible Problems

If on building you find the Latex error:

```
(/opt/local/share/texmf-dist/tex/latex/base/inputenc.sty
! LaTeX Error: File 'utf8x.def' not found.
```

You can use a machine with a newer latex/tetex distribution, or kludge your Sphinx:

```
perl -pi -e 's,utf8x,utf8,' ~/v/docs/lib/python2.7/site-packages/sphinx/ext/pngmath.py
```

## 24.1.2 Sphinx Customizations/Primer

The general usage of Sphinx and reStructuredText are well documented:

- *Sphinx*
- *reStructuredText Primer*
- quick primer [an\\_example\\_pypi\\_project](#)

This document covers customizations made for the Offline User Manual and the features these customizations provide. Use the **Show Source** links in the html sidebar of every page to see more usage examples.

## External Links

Commands are defined in `dybgaudi:Documentation/OfflineUserManual/tex/main.tex` to facilitate referencing external links from latex sources. Corresponding sphinx extlinks are configured in `dybgaudi:Documentation/OfflineUserManual/tex/conf.py` to allow similar usage from reStructuredText sources:

```
extlinks = {
    'dybsvn': ('http://dayabay.ihep.ac.cn/tracs/dybsvn/intertrac/%s', 'dybsvn:'),
    'source': ('http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/%s', 'source:'),
    'dybgaudi': ('http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/dybgaudi/trunk/%s', 'dybgaudi:'),
    'dybaux': ('http://dayabay.ihep.ac.cn/tracs/dybaux/intertrac/%s', 'dybaux:'),
    'wiki': ('https://wiki.bnl.gov/dayabay/index.php?title=%s', 'wiki:'),
    'doc': ('http://dayabay.ihep.ac.cn/cgi-bin/DocDB/ShowDocument?docid=%s', 'doc:'),
    'docdb': ('http://dayabay.ihep.ac.cn/cgi-bin/DocDB/ShowDocument?docid=%s', 'doc:'),
}
```

latex source	reStructuredText source	render
<code>\dybsvn{ticket:666}</code>	<code>:dybsvn: `ticket:666`</code>	<code>dybsvn:ticket:666</code>
<code>\dybaux{source:catalog}</code>	<code>:dybaux: `source:catalog`</code>	<code>dybaux:source:catalog</code>
<code>\doc{999}</code>	<code>:doc: `999`</code>	<code>doc:999</code>
<code>\wiki{Database}</code>	<code>:wiki: `Database`</code>	<code>wiki:Database</code>

An inline example of the reStructuredText source to create such links, using *dybsvn* and *docdb* roles:

```
Beastly :dybsvn:`ticket:666` and lucky :docdb:`888`
```

Futher details at `sphinx.ext.extlinks`.

## Intersphinx

A facility for simple linking to **objects** (in a very general sense) from other Sphinx documented projects is implemented in the extension module `sphinx.ext.intersphinx`. This for example allows inline linking to a python class `zipfile.ZipFile` and a matplotlib module `matplotlib.pyplot` without specifying the precise target URL.

Spelling out the source (also use the):

reST source	render
<code>:py:mod: `sphinx.ext.intersphinx`</code>	<code>sphinx.ext.intersphinx</code>
<code>:mod: `sphinx.ext.intersphinx`</code>	<code>sphinx.ext.intersphinx</code>
<code>:py:class: `zipfile.ZipFile`</code>	<code>zipfile.ZipFile</code>
<code>:py:mod: `matplotlib.pyplot`</code>	<code>matplotlib.pyplot</code>
<code>:meth: `matplotlib.pyplot.acorr`</code>	<code>matplotlib.pyplot.acorr()</code>
<code>:mod: `numpy`</code>	<code>numpy</code>
<code>:class: `numpy.ndarray`</code>	<code>numpy.ndarray</code>
<code>:rst:dir: `math`</code>	<code>math</code>
<code>:rst:role: `math`</code>	<code>math</code>
<code>:rst:directive: `math`</code>	<b>FAILS</b>

Note that the `:py:` is not strictly needed as **py** is the default domain.

This is configured in `conf.py` with:

```
intersphinx_cache_limit = 10      # days to keep the cached inventories
intersphinx_mapping = {
    'sphinx': ('http://sphinx.pocoo.org', None),
    'python': ('http://docs.python.org/2.7', None),
    'matplotlib': ('http://matplotlib.sourceforge.net', None),
    'numpy': ('http://docs.scipy.org/doc/numpy', None),
}
```

## Object Inventories

A simple script to dump the content of intersphinx inventories is at `docs/inventory.py`. Use it to find reference targets, for example:

```
./docs/inventory.py sphinx | grep reStructured
rst-primer = (u'Sphinx', u'1.0.6', u'./docs/inv/sphinx/objects.inv/rest.html#rst-primer', u'reStru

./docs/inventory.py sphinx std:label
std:label
basic-domain-markup = (u'Sphinx', u'1.0.6', u'./docs/inv/sphinx/objects.inv/domains.html#basic-dor
build-config = (u'Sphinx', u'1.0.6', u'./docs/inv/sphinx/objects.inv/config.html#build-config', u
builders = (u'Sphinx', u'1.0.6', u'./docs/inv/sphinx/objects.inv/builders.html#builders', u'Avail
builtin-themes = (u'Sphinx', u'1.0.6', u'./docs/inv/sphinx/objects.inv/theming.html#builtin-themes
...

./docs/inventory.py self std:label      ## "self" refers to the Offline User Manual inventory
std:label
api-main = (u'Offline User Manual', u'0.1', u'./_build/dirhtml/objects.inv/api/main/#api-main', u
ch:framework = (u'Offline User Manual', u'0.1', u'./_build/dirhtml/objects.inv/framework/main/#ch
ch:source = (u'Offline User Manual', u'0.1', u'./_build/dirhtml/objects.inv/sourcecode/main/#ch-
...
```

Shows that can refer to the primer as shown in the below table, labels begging `std:` are referred to with the `ref` role others use the dedicated role for the object type.

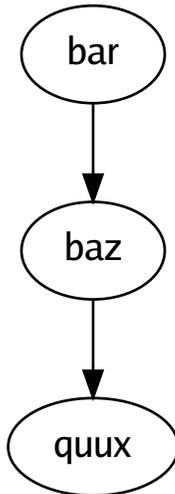
reST source	render	note
<code>:ref:`rst-primer`</code>	<i>reStructuredText Primer</i>	no need to specify sphinx
<code>:ref:`invocation`</code>	<i>Invocation of sphinx-build</i>	label from resolving inventory
<code>:ref:`&lt;sphinx:invocation&gt;`</code>	<code>&lt;sphinx:invocation&gt;</code>	specify inventory <b>FAILS</b>
<code>:ref:`from sphinx&lt;sphinx:invocation&gt;`</code>	<i>from sphinx</i>	my label, defines src proj

## Graphviz Figures

Sphinx graphviz extension provides `graphviz`, `graph` and `digraph`, allowing source like:

```
.. digraph:: foo
    "bar" -> "baz" -> "quux";
```

To generates a figure:



For details see `sphinx.ext.graphviz`, for an intro take your pick from [google:graphviz dot tutorial](#) eg [wikipedia:Dot\\_language](#)

### 24.1.3 Autodoc : pulling reStructuredText from docstrings

Sphinx has an `sphinx.ext.autodoc` feature that allows reStructuredText to be extracted out of docstrings in your python code. reStructuredText was designed for usage in docstrings, featuring a very light weight markup that is very readable in its source form.

See pages beneath *NuWa Python API* and use **Show Source** or view sources at [dyb-gaudi:Documentation/OfflineUserManual/tex/api](#) to see how the autodoc directives are used

- `automodule`
- `autoclass`

And examine the docstrings at for example:

- `dybgaudi:DybPython/python/DybPython/db.py`

### How to AutoDocifying a module

Create a `.rst` named after your module in the hierarchy beneath `dybgaudi:Documentation/OfflineUserManual/tex/api`. Entire modules can be autodoc-ified with a couple of lines (pay attention to the `__all__` setting for the python modules):

```
.. automodule:: <python-dotted-path>
   :members:
```

But the resulting docs are liable to include too much. Creating useful docs requires draconian editing to only include what is instructive, beyond that the source code should be consulted.

### Creating Useful AutoDocumentation

Creating useful API docs requires full control of what is included (which classes/functions/methods) and how they are grouped/ordered/presented/headered/indexed. Autodoc `.rst` files need to be source files rather than generated files in order to provide this control.

### tips for docstring preparation

A gotcha when improving the docstrings is to forget to `cmt <pkg>_python` after changing them, as Sphinx is reading them from `sys.path` not the sources.

### docstring debugging

1. relative indentation is significant to reST, thus use consistent indentation for your docstrings. An example of docstring cleanup is `dybsvn:r10222`
2. error reporting seems to get incorrect line nos in docstrings, use non-existing roles eg `:red: `dummy`` to instrument the docstrings

### general reST debugging

1. leave a blank line before literal blocks
2. for the colon pointing to a literal block to be visible abutt the the double colon against the last word of the prior para
3. inline literals cannot start or end with a space

## 24.1.4 Doxygen : automated documentation of C++ API

### Doxygen Integration with Breathe

A possible future enhancement is to integrate doxygen documentation of C++ code into the Offline User Manual.

The `breathe` project provides an extension to reStructuredText and Sphinx that enables reading and rendering of Doxygen xml output.

It is usable at whatever granularity is desired (similar to Sphinx autodoc) enabling the problem of boring and unreadable auto-generated API docs to be avoided, albeit with some effort from the documenters.

**See Also:***Doxygen Publishing*

## 24.1.5 Publishing Documentation on Separate Webserver

### Configure Target

Snippet from `~/ .dybinstrc`:

```
sphinx_vpy=$HOME/rst
sphinx_pub=C:/tmp/oum
```

The `sphinx_vpy` configures the location of the virtual python in which Sphinx and dependencies are installed. The presence of `sphinx_pub` causes the generated html to be **rsynced** to the target node directory specified.

The example `sphinx_pub` assumes a C host alias in the `~/ .ssh/config`:

```
host C
    user blyth
    hostname target.node.domain
    protocol 2
```

### Test Dybinst build

**Disable until passwordless SSH operational**

Prevent the slave hanging by commenting out the `sphinx_pub=...` until passwordless SSH is operational

Test interactively with:

```
./dybinst trunk docs sphinx
```

If that pauses for password entry then passwordless SSH is not configured and/or an ssh-agent is not running. See [env:wiki>PasswordLessSSH](http://env:wiki>PasswordLessSSH)

**Warning:** ssh-agent must be manually restarted after rebooting the slave node to avoid slave hangs

### Doxygen Publishing

In a similar manner the `dox` documentation derived by `doxygen` can be published to another node with

```
doxyman_pub=C:/tmp/dox
```

To debug `doxygen` building, test interactively with:

```
./dybinst trunk docs doxyman
```

Implemented with [dybsvn:r11922](https://github.com/robertadams/dybsvn/pull/11922), issues with `doxygen` docs discussed in [dybsvn:ticket:655](https://github.com/robertadams/dybsvn/issues/655)

## 24.1.6 Sandbox Testing reST/Sphinx

Examine the source with the **Show Source** links in the html sidebar to see the reST markup used to create this.

## Matplotlib extensions

---

**Note:** matplotlib dependency removal

In order to simplify documentation building, the dependency on matplotlib has been removed requiring all live ipython blocks to be converted to dead code blocks

---

### *live ipython session with ipython directive*

See *Ipython Directive*

#### **live ipython**

The commands are actually performed when the documentation is built, ensuring uptodate ... but risking errors in the documentation

```
In [136]: x = 2
```

```
In [137]: x**3
```

The session remembers its scope values `x` and numbers its In Out

```
In [4]: x
```

#### **dead ipython**

See *ipython sessions*

```
In [69]: lines = plot([1,2,3])
```

```
In [70]: setp(lines)
alpha: float
animated: [True | False]
antialiased or aa: [True | False]
...snip
```

#### **inline plots**

See *Inserting matplotlib plots*

Before removal of matplotlib dependency plots could be included inline with:

```
.. plot::
   :include-source:

   import matplotlib.pyplot as plt
   import numpy as np
   x = np.random.randn(1000)
   plt.hist( x, 20)
   plt.grid()
   plt.title(r'Normal:  $\mu=0.2f$ ,  $\sigma=0.2f$ '%(x.mean(), x.std()))
   plt.show()
```

## Syntax Highlighting

Pygments emits Lexer name not known for C++ or Python or C, instead use `cpp`, `python`, or `c`

```
public:
    static const CLID& classID() {
        return DayaBay::CLID_GenHeader;
    }

    GenHeaderCnv(ISvcLocator* svc);
    virtual ~GenHeaderCnv();

def __init__(self):
    pass

int main(int argc, char argv[])
```

## math

Latex math markup used by the `math` directive.

### math equation

$$V(t) = VoltageScale \cdot \frac{(e^{-t/t_0} - e^{-t/t_1})}{(t_1 - t_0)} \quad (24.1)$$

$$t_0 = 3.6ns$$

$$t_1 = 5.4ns$$

equation (??) is propagated from label

### math eqnarray

$$y = ax^2 + bx + c \quad (24.2)$$

$$f(x) = x^2 + 2xy + y^2 \quad (24.3)$$

labels are not ferreted out of the math (??) ... just passed to latex to create a png presumably

## raw html css usage via reST custom roles

Section contains invisble content that create custom roles `r`, `v` and `g` that are used to style cells of the below tables.

### table styled with custom role

Table 24.1: Frozen Delights!

Treat	Quantity	Description
Albatross	2.99	On a stick!
Crunchy Frog	1.49	If we took the bones out, it wouldn't be crunchy, now would it?
Gannet Ripple	1.99	On a stick!

The role results in the html:

```
<tr>
  <td><span class="custom">Gannet Ripple</span></td>
  <td>1.99</td>
  <td>On a stick!</td>
</tr>
```

## longtable

Name & Synonyms	Type	Track	Ver- tex	Stats	Description
timet	double	X	X	X	Time of the vertex/track start
x global_x	double	X	X	X	Global X position of the vertex/track start/step
y global_y	d	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
EnergyLostSince- LastVertex	double		X		Energy difference sine the last created SimVertex
AngleFromLastVertex	double		X		Change in direction since the last created SimVertex (degrees)

## figures

A code block can be placed in the legend of a figure.

The `ref` role is used to refer to the fig by its label `f:test_simtrack_accessors`

## tabledoc

Generate the below list of tabledoc directives with something like

```
echo show tables | mysql dcs | perl -p -e 's,(\S*),.. tabledoc:: dcs $1, ' -
```

## 24.1.7 Dayabay Sphinx Extensions

Sphinx extensions allow arbitrary *rst* generating python to be performed on building the documentation. Allowing documentation or other output to be dynamically generated.

### Table Doc Directive

Invoking the **tabledoc** directive (from `OfflineUserManual.sphinxext.tabledoc`) with `dbconf` (section name in `~/ .my.cnf`) and `tablename` arguments:

```
.. tabledoc:: offline_db LOCALSEQNO
```

Performs a live DB description lookup and converts the *MySQL-python* output into an *rst* table.

Figure 24.1: `f:test_simtrack_accessors`  
**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;
    ...
}
```

Extra	Default	Field	Key	Null	Type
	None	TABLENAME	PRI	NO	char(64)
	None	LASTUSEDSEQNO		YES	int(11)

This is configured in the Sphinx *conf.py* with:

```
extensions += [ 'OfflineUserManual.sphinxext.tabledoc' ]
```

Further details in *Sphinx Extensions*

## DBI Validity Record

Invoke directive with:

```
.. dbivld:: tmp_offline_db Demo 1,10
```

Yielding a table:

dbivld directive : within dbconf tmp\_offline\_db no such table Demo

## DBI Context Query

Invoke directive with:

```
.. dbictx:: tmp_offline_db Demo
   :site: 127
   :simflag: 1
   :task: 0
   :subsite: 0
```

Yielding table:

dbictx directive : within dbconf tmp\_offline\_db no such table Demo

## DBI Validity Lookup Table

Invoke directive with:

```
.. dbivlut:: tmp_offline_db Demo
```

## 24.2 Todolist

Collection of todo notes sprinkled across the documentation provided by `todolist`

---

### Todo

Find way to avoid/capture the error after failure to connect

---

(The *original entry* is located in `api/dbcas.rst`, line 53.)

---

### Todo

test return to numpy original, now that my changes are integrated

---

(The *original entry* is located in docs/build.rst, line 60.)

---

**Todo**

Provide a way for non-administrators to do this style of debugging, perhaps with an extra DBI log file ?

---

(The *original entry* is located in sop/dbdebug.rst, line 134.)

---

**Todo**

plant internal reference targets to genDbi documentation

---

(The *original entry* is located in sop/dbspec.rst, line 133.)

---

**Todo**

enforce usage of overlay date in pre-commit hook

---

(The *original entry* is located in sop/dbwrite.rst, line 138.)

---

**Todo**

try changing implementation of enums to make them usable from python

---

(The *original entry* is located in sop/dbwrite.rst, line 300.)

---

## 24.3 References



# UNRECOGNIZED LATEX COMMANDS

None



# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*



# BIBLIOGRAPHY

[g4dyb] Reference target needed for g4dyb



# PYTHON MODULE INDEX

## d

- DbiDataSvc, 417
- DybDbi, 349
- DybDbi.vld.versiondate, 356
- DybDbi.vld.vlut, 357
- DybDbi.vld.vsmry, 359
- DybDbiPre, 348
- DybPython, 414
- DybPython.Control, 414
- DybPython.db, 327
- DybPython.dbaux, 337
- DybPython.dbcas, 343
- DybPython.dbconf, 340
- DybPython.dbicnf, 415
- DybPython.dbsvn, 344
- dybtest, 431
- dybtest.capture, 433
- dybtest.cfroot, 432
- dybtest.histref, 431

## n

- NonDbi, 417

## s

- Scraper, 421
- Scraper.adlidsensor, 424
- Scraper.adtemp, 423
- Scraper.dcs, 425
- Scraper.pmthv, 421



# INDEX

## Symbols

`__call__()` (DybDbiPre.Tab method), 349

## A

`adcpedestalhigh` (DybDbi.GCalibFeeSpec attribute), 385  
`adcpedestalhighsigma` (DybDbi.GCalibFeeSpec attribute), 385  
`adcpedestallow` (DybDbi.GCalibFeeSpec attribute), 385  
`adcpedestallowsigma` (DybDbi.GCalibFeeSpec attribute), 385  
`adthresholdhigh` (DybDbi.GCalibFeeSpec attribute), 385  
`adthresholdlow` (DybDbi.GCalibFeeSpec attribute), 385  
`Add` (DybDbi.TimeStamp attribute), 366  
`add_input_file()` (DybPython.Control.NuWa method), 414  
`add_service_redirect()` (DybPython.Control.NuWa method), 414  
`AdLidSensor` (class in Scraper.adlidsensor), 424  
`AdLogicalPhysical` (class in DybDbi), 362  
`adno` (DybDbi.GDaqCalibRunInfo attribute), 398  
`AdTemp` (class in Scraper.adtemp), 423  
`AdTempFaker` (class in Scraper.adtemp), 424  
`AdTempScraper` (class in Scraper.adtemp), 423  
`AdTempSource` (class in Scraper.adtemp), 423  
`afterpulseprob` (DybDbi.GCalibPmtSpec attribute), 381  
`afterpulseprob` (DybDbi.GSimPmtSpec attribute), 377  
`aggregateno` (DybDbi.GCalibFeeSpec attribute), 385  
`aggregateno` (DybDbi.GCalibPmtSpec attribute), 381  
`aggregateno` (DybDbi.GDaqCalibRunInfo attribute), 398  
`aggregateno` (DybDbi.GDaqRawDataFileInfo attribute), 402  
`aggregateno` (DybDbi.GDaqRunInfo attribute), 393  
`aggregateno` (DybDbi.GDbiLogEntry attribute), 405  
`aggregateno` (DybDbi.GDcsAdTemp attribute), 409  
`aggregateno` (DybDbi.GDcsPmtHv attribute), 412  
`aggregateno` (DybDbi.GFeeCableMap attribute), 389  
`aggregateno` (DybDbi.GPhysAd attribute), 373  
`aggregateno` (DybDbi.GSimPmtSpec attribute), 377  
`allseqno` (DybPython.db.DB attribute), 329  
`AParser` (class in Scraper.base.aparser), 430  
`AssignTimeGate` (DybDbi.GCalibFeeSpec attribute), 383

`AssignTimeGate` (DybDbi.GCalibPmtSpec attribute), 379  
`AssignTimeGate` (DybDbi.GDaqCalibRunInfo attribute), 395  
`AssignTimeGate` (DybDbi.GDaqRawDataFileInfo attribute), 400  
`AssignTimeGate` (DybDbi.GDaqRunInfo attribute), 391  
`AssignTimeGate` (DybDbi.GDcsAdTemp attribute), 407  
`AssignTimeGate` (DybDbi.GDcsPmtHv attribute), 410  
`AssignTimeGate` (DybDbi.GFeeCableMap attribute), 387  
`AssignTimeGate` (DybDbi.GPhysAd attribute), 372  
`AssignTimeGate` (DybDbi.GSimPmtSpec attribute), 375  
`AsString` (DybDbi.Context attribute), 363  
`AsString` (DybDbi.ContextRange attribute), 364  
`AsString` (DybDbi.Ctx attribute), 354  
`AsString` (DybDbi.DetectorId attribute), 369  
`AsString` (DybDbi.SimFlag attribute), 369  
`AsString` (DybDbi.Site attribute), 368  
`AsString` (DybDbi.TimeStamp attribute), 366  
`automap()` (DybDbi.Mapper method), 354  
`Aux` (class in DybPython.dbaux), 340

## B

`baseversion` (DybDbi.GDaqRunInfo attribute), 393  
`bot` (DybDbi.TimeStamp attribute), 367  
`BUILD_REVISION`, 190

## C

`Cache` (DybDbi.GCalibFeeSpec attribute), 383  
`Cache` (DybDbi.GCalibPmtSpec attribute), 379  
`Cache` (DybDbi.GDaqCalibRunInfo attribute), 395  
`Cache` (DybDbi.GDaqRawDataFileInfo attribute), 400  
`Cache` (DybDbi.GDaqRunInfo attribute), 391  
`Cache` (DybDbi.GDbiLogEntry attribute), 404  
`Cache` (DybDbi.GDcsAdTemp attribute), 407  
`Cache` (DybDbi.GDcsPmtHv attribute), 410  
`Cache` (DybDbi.GFeeCableMap attribute), 387  
`Cache` (DybDbi.GPhysAd attribute), 372  
`Cache` (DybDbi.GSimPmtSpec attribute), 375  
`CanFixOrdering` (DybDbi.GSimPmtSpec attribute), 375  
`CanL2Cache` (DybDbi.GCalibFeeSpec attribute), 383  
`CanL2Cache` (DybDbi.GCalibPmtSpec attribute), 379

- CanL2Cache (DybDbi.GDaqCalibRunInfo attribute), 395
- CanL2Cache (DybDbi.GDaqRawDataFileInfo attribute), 400
- CanL2Cache (DybDbi.GDaqRunInfo attribute), 391
- CanL2Cache (DybDbi.GDcsAdTemp attribute), 407
- CanL2Cache (DybDbi.GDcsPmtHv attribute), 410
- CanL2Cache (DybDbi.GFeeCableMap attribute), 387
- CanL2Cache (DybDbi.GPhysAd attribute), 372
- CanL2Cache (DybDbi.GSimPmtSpec attribute), 375
- Capture (class in dybtest.capture), 433
- cfg\_() (in module NonDbi), 420
- CfHist (class in dybtest.cfront), 432
- CfRoot (class in dybtest.cfront), 432
- changed() (Scraper.adtemp.AdTempScraper method), 424
- changed() (Scraper.base.Scraper method), 427
- changed() (Scraper.pmtHv.PmtHvScraper method), 422
- chanhrdwdesc (DybDbi.GFeeCableMap attribute), 389
- channelid (DybDbi.GCalibFeeSpec attribute), 385
- check\_() (DybPython.db.DB method), 330
- check\_allseqno() (DybPython.db.DB method), 330
- check\_kv() (DybDbi.Mapper method), 354
- check\_physical2logical() (DybDbi.AdLogicalPhysical method), 363
- check\_seqno() (DybPython.db.DB method), 330
- check\_versiondate() (in module DybDbi.vld.versiondate), 356
- check\_versiondate\_tab() (in module DybDbi.vld.versiondate), 357
- checksum (DybDbi.GDaqRawDataFileInfo attribute), 402
- clean() (DybDbi.Source method), 353
- CloneAndSubtract (DybDbi.TimeStamp attribute), 366
- Close (DybDbi.GCalibFeeSpec attribute), 383
- Close (DybDbi.GCalibPmtSpec attribute), 379
- Close (DybDbi.GDaqCalibRunInfo attribute), 395
- Close (DybDbi.GDaqRawDataFileInfo attribute), 400
- Close (DybDbi.GDaqRunInfo attribute), 391
- Close (DybDbi.GDbiLogEntry attribute), 404
- Close (DybDbi.GDcsAdTemp attribute), 407
- Close (DybDbi.GDcsPmtHv attribute), 411
- Close (DybDbi.GFeeCableMap attribute), 387
- Close (DybDbi.GPhysAd attribute), 372
- Close (DybDbi.GSimPmtSpec attribute), 375
- cmdline() (DybPython.Control.NuWa method), 414
- column (DybDbi.GDcsPmtHv attribute), 412
- Compare (DybDbi.GCalibFeeSpec attribute), 383
- Compare (DybDbi.GCalibPmtSpec attribute), 379
- Compare (DybDbi.GDaqCalibRunInfo attribute), 395
- Compare (DybDbi.GDaqRawDataFileInfo attribute), 400
- Compare (DybDbi.GDaqRunInfo attribute), 391
- Compare (DybDbi.GDcsAdTemp attribute), 407
- Compare (DybDbi.GDcsPmtHv attribute), 411
- Compare (DybDbi.GFeeCableMap attribute), 387
- Compare (DybDbi.GPhysAd attribute), 372
- Compare (DybDbi.GSimPmtSpec attribute), 375
- config() (Scraper.base.parser.Parser class method), 431
- configure\_args() (DybPython.Control.NuWa method), 414
- configure\_cascade() (DybPython.dbconf.DBConf method), 342
- configure\_dbconf() (DybPython.Control.NuWa method), 414
- configure\_dbi() (DybPython.Control.NuWa method), 414
- configure\_dyb\_services() (DybPython.Control.NuWa method), 414
- configure\_framework() (DybPython.Control.NuWa method), 415
- configure\_ipython() (DybPython.Control.NuWa method), 415
- configure\_mod() (DybPython.Control.NuWa method), 415
- configure\_optmods() (DybPython.Control.NuWa method), 415
- configure\_python\_features() (DybPython.Control.NuWa method), 415
- configure\_visualization() (DybPython.Control.NuWa method), 415
- Context (class in DybDbi), 363
- context (DybDbi.ServiceMode attribute), 368
- ContextRange (class in DybDbi), 364
- convert\_csv2dbi() (DybDbi.Mapper method), 354
- Copy (DybDbi.TimeStamp attribute), 366
- count\_() (DybPython.db.DB method), 330
- cr (DybPython.dbicnf.DbiCnf attribute), 416
- Create() (DybDbi.GCalibFeeSpec class method), 383
- Create() (DybDbi.GCalibPmtSpec class method), 379
- Create() (DybDbi.GDaqCalibRunInfo class method), 395
- Create() (DybDbi.GDaqRawDataFileInfo class method), 400
- Create() (DybDbi.GDaqRunInfo class method), 391
- Create() (DybDbi.GDbiLogEntry class method), 404
- Create() (DybDbi.GDcsAdTemp class method), 407
- Create() (DybDbi.GDcsPmtHv class method), 411
- Create() (DybDbi.GFeeCableMap class method), 387
- Create() (DybDbi.GPhysAd class method), 372
- Create() (DybDbi.GSimPmtSpec class method), 375
- CreateTableRow (DybDbi.GCalibFeeSpec attribute), 383
- CreateTableRow (DybDbi.GCalibPmtSpec attribute), 379
- CreateTableRow (DybDbi.GDaqCalibRunInfo attribute), 395
- CreateTableRow (DybDbi.GDaqRawDataFileInfo attribute), 401
- CreateTableRow (DybDbi.GDaqRunInfo attribute), 391
- CreateTableRow (DybDbi.GDbiLogEntry attribute), 404
- CreateTableRow (DybDbi.GDcsAdTemp attribute), 407
- CreateTableRow (DybDbi.GDcsPmtHv attribute), 411
- CreateTableRow (DybDbi.GFeeCableMap attribute), 387
- CreateTableRow (DybDbi.GPhysAd attribute), 372

- CreateTableRow (DybDbi.GSimPmtSpec attribute), 375  
 CSV (class in DybDbi), 352  
 csv\_check() (DybDbi.GCalibFeeSpec class method), 385  
 csv\_check() (DybDbi.GCalibPmtSpec class method), 381  
 csv\_check() (DybDbi.GDaqCalibRunInfo class method), 398  
 csv\_check() (DybDbi.GDaqRawDataFileInfo class method), 402  
 csv\_check() (DybDbi.GDaqRunInfo class method), 393  
 csv\_check() (DybDbi.GDbiLogEntry class method), 405  
 csv\_check() (DybDbi.GDcsAdTemp class method), 409  
 csv\_check() (DybDbi.GDcsPmtHv class method), 412  
 csv\_check() (DybDbi.GFeeCableMap class method), 389  
 csv\_check() (DybDbi.GPhysAd class method), 373  
 csv\_check() (DybDbi.GSimPmtSpec class method), 377  
 csv\_compare() (DybDbi.GCalibFeeSpec class method), 386  
 csv\_compare() (DybDbi.GCalibPmtSpec class method), 381  
 csv\_compare() (DybDbi.GDaqCalibRunInfo class method), 398  
 csv\_compare() (DybDbi.GDaqRawDataFileInfo class method), 403  
 csv\_compare() (DybDbi.GDaqRunInfo class method), 393  
 csv\_compare() (DybDbi.GDbiLogEntry class method), 405  
 csv\_compare() (DybDbi.GDcsAdTemp class method), 409  
 csv\_compare() (DybDbi.GDcsPmtHv class method), 412  
 csv\_compare() (DybDbi.GFeeCableMap class method), 389  
 csv\_compare() (DybDbi.GPhysAd class method), 373  
 csv\_compare() (DybDbi.GSimPmtSpec class method), 377  
 csv\_export() (DybDbi.GCalibFeeSpec class method), 386  
 csv\_export() (DybDbi.GCalibPmtSpec class method), 381  
 csv\_export() (DybDbi.GDaqCalibRunInfo class method), 398  
 csv\_export() (DybDbi.GDaqRawDataFileInfo class method), 403  
 csv\_export() (DybDbi.GDaqRunInfo class method), 393  
 csv\_export() (DybDbi.GDbiLogEntry class method), 405  
 csv\_export() (DybDbi.GDcsAdTemp class method), 409  
 csv\_export() (DybDbi.GDcsPmtHv class method), 412  
 csv\_export() (DybDbi.GFeeCableMap class method), 390  
 csv\_export() (DybDbi.GPhysAd class method), 373  
 csv\_export() (DybDbi.GSimPmtSpec class method), 377  
 csv\_import() (DybDbi.GCalibFeeSpec class method), 386  
 csv\_import() (DybDbi.GCalibPmtSpec class method), 382  
 csv\_import() (DybDbi.GDaqCalibRunInfo class method), 398  
 csv\_import() (DybDbi.GDaqRawDataFileInfo class method), 403  
 csv\_import() (DybDbi.GDaqRunInfo class method), 393  
 csv\_import() (DybDbi.GDbiLogEntry class method), 405  
 csv\_import() (DybDbi.GDcsAdTemp class method), 409  
 csv\_import() (DybDbi.GDcsPmtHv class method), 413  
 csv\_import() (DybDbi.GFeeCableMap class method), 390  
 csv\_import() (DybDbi.GPhysAd class method), 374  
 csv\_import() (DybDbi.GSimPmtSpec class method), 377  
 Ctx (class in DybDbi), 354  
 ctx\_count() (in module DybDbi.vld.vsmry), 360  
 CurrentTimeGate (DybDbi.GCalibFeeSpec attribute), 383  
 CurrentTimeGate (DybDbi.GCalibPmtSpec attribute), 379  
 CurrentTimeGate (DybDbi.GDaqCalibRunInfo attribute), 395  
 CurrentTimeGate (DybDbi.GDaqRawDataFileInfo attribute), 401  
 CurrentTimeGate (DybDbi.GDaqRunInfo attribute), 391  
 CurrentTimeGate (DybDbi.GDcsAdTemp attribute), 407  
 CurrentTimeGate (DybDbi.GDcsPmtHv attribute), 411  
 CurrentTimeGate (DybDbi.GFeeCableMap attribute), 388  
 CurrentTimeGate (DybDbi.GPhysAd attribute), 372  
 CurrentTimeGate (DybDbi.GSimPmtSpec attribute), 375
- ## D
- darkrate (DybDbi.GCalibPmtSpec attribute), 382  
 darkrate (DybDbi.GSimPmtSpec attribute), 378  
 databaselayout (DybDbi.GCalibFeeSpec attribute), 386  
 databaselayout (DybDbi.GCalibPmtSpec attribute), 382  
 databaselayout (DybDbi.GDaqCalibRunInfo attribute), 399  
 databaselayout (DybDbi.GDaqRawDataFileInfo attribute), 403  
 databaselayout (DybDbi.GDaqRunInfo attribute), 394  
 databaselayout (DybDbi.GDbiLogEntry attribute), 406  
 databaselayout (DybDbi.GDcsAdTemp attribute), 409  
 databaselayout (DybDbi.GDcsPmtHv attribute), 413  
 databaselayout (DybDbi.GFeeCableMap attribute), 390  
 databaselayout (DybDbi.GPhysAd attribute), 374  
 databaselayout (DybDbi.GSimPmtSpec attribute), 378  
 dataversion (DybDbi.GDaqRunInfo attribute), 394  
 date (DybDbi.TimeStamp attribute), 367  
 DB (class in DybPython.db), 329  
 DBCas (class in DybPython.dbcas), 343  
 DBCon (class in DybPython.dbcas), 343  
 DBCONF, 206, 210, 224, 242, 342  
 DBConf (class in DybPython.dbconf), 341  
 DBCONF\_PATH, 342, 343

- Dbi (class in DybDbi), 370
  - DbiCnf (class in DybPython.dbicnf), 415
  - DbiDataSvc (module), 417
  - DBIValidate (class in DybPython.dbsvn), 347
  - DCS (class in Scraper.base), 426
  - DD (class in DybPython.dbcas), 344
  - define\_\_repr\_\_() (DybDbi.Wrap method), 350
  - define\_create() (DybDbi.Wrap method), 350
  - define\_csv() (DybDbi.Wrap method), 351
  - define\_listlike() (DybDbi.Wrap method), 351
  - define\_properties() (DybDbi.Wrap method), 351
  - define\_update() (DybDbi.Wrap method), 351
  - desc() (DybPython.db.DB method), 330
  - descline() (DybDbi.Source method), 353
  - describ (DybDbi.GCalibPmtSpec attribute), 382
  - describ (DybDbi.GSimPmtSpec attribute), 378
  - describe() (DybPython.db.DB method), 330
  - Detector (in module DybDbi), 369
  - DetectorId (class in DybDbi), 369
  - detectorid (DybDbi.GDaqCalibRunInfo attribute), 399
  - detectormask (DybDbi.GDaqRunInfo attribute), 394
  - DetectorSensor (class in DybDbi), 369
  - detid (DybDbi.Context attribute), 364
  - digest (DybDbi.GCalibFeeSpec attribute), 386
  - digest (DybDbi.GCalibPmtSpec attribute), 382
  - digest (DybDbi.GDaqCalibRunInfo attribute), 399
  - digest (DybDbi.GDaqRawDataFileInfo attribute), 403
  - digest (DybDbi.GDaqRunInfo attribute), 394
  - digest (DybDbi.GDbiLogEntry attribute), 406
  - digest (DybDbi.GDcsAdTemp attribute), 409
  - digest (DybDbi.GDcsPmtHv attribute), 413
  - digest (DybDbi.GFeeCableMap attribute), 390
  - digest (DybDbi.GPhysAd attribute), 374
  - digest (DybDbi.GSimPmtSpec attribute), 378
  - dj\_init\_() (in module NonDbi), 420
  - docs() (DybPython.db.DB class method), 330
  - DoubleValueForKey (DybDbi.GCalibFeeSpec attribute), 383
  - DoubleValueForKey (DybDbi.GCalibPmtSpec attribute), 379
  - DoubleValueForKey (DybDbi.GDaqCalibRunInfo attribute), 395
  - DoubleValueForKey (DybDbi.GDaqRawDataFileInfo attribute), 401
  - DoubleValueForKey (DybDbi.GDaqRunInfo attribute), 391
  - DoubleValueForKey (DybDbi.GDbiLogEntry attribute), 404
  - DoubleValueForKey (DybDbi.GDcsAdTemp attribute), 407
  - DoubleValueForKey (DybDbi.GDcsPmtHv attribute), 411
  - DoubleValueForKey (DybDbi.GFeeCableMap attribute), 388
  - DoubleValueForKey (DybDbi.GPhysAd attribute), 372
  - DoubleValueForKey (DybDbi.GSimPmtSpec attribute), 375
  - dump\_() (DybPython.db.DB method), 330
  - dump\_ctxsmry() (in module DybDbi.vld.vsmry), 360
  - dump\_difctx() (in module DybDbi.vld.vsmry), 361
  - dump\_diff() (DybPython.dbsvn.DBIValidate method), 348
  - DumpTMStruct (DybDbi.TimeStamp attribute), 366
  - duration (DybDbi.GDaqCalibRunInfo attribute), 399
  - DYB\_DB\_PWSD, 342
  - DYB\_DB\_URL, 342
  - DYB\_DB\_USER, 342
  - DybDbi (module), 349
  - DybDbi.vld.versiondate (module), 356
  - DybDbi.vld.vlut (module), 357
  - DybDbi.vld.vsmry (module), 359
  - DybDbiPre (module), 348
  - DybPython (module), 414
  - DybPython.Control (module), 414
  - DybPython.db (module), 327
  - DybPython.dbaux (module), 337
  - DybPython.dbcas (module), 343
  - DybPython.dbconf (module), 340
  - DybPython.dbicnf (module), 415
  - DybPython.dbsvn (module), 344
  - dybtest (module), 431
  - dybtest.capture (module), 433
  - dybtest.cfroot (module), 432
  - dybtest.histref (module), 431
- ## E
- efficiency (DybDbi.GCalibPmtSpec attribute), 382
  - efficiency (DybDbi.GSimPmtSpec attribute), 378
  - engine\_() (in module NonDbi), 420
  - ENV\_TSQL\_FIX, 342, 343
  - ENV\_TSQL\_PSWD, 342
  - ENV\_TSQL\_URL, 342
  - ENV\_TSQL\_USER, 342
  - environment variable
    - BUILD\_REVISION, 190
    - DBCONF, 206, 210, 224, 242, 342
    - DBCONF\_DB, 342
    - DBCONF\_FIX, 342
    - DBCONF\_FIXPASS, 342
    - DBCONF\_HOST, 342
    - DBCONF\_PATH, 342, 343
    - DBCONF\_PWSD, 342
    - DBCONF\_RESTRICT, 342
    - DBCONF\_URL, 342
    - DBCONF\_USER, 342
    - DYB\_DB\_PWSD, 342
    - DYB\_DB\_URL, 342
    - DYB\_DB\_USER, 342

- ENV\_TSQL\_FIX, 342, 343
  - ENV\_TSQL\_PSWD, 342
  - ENV\_TSQL\_URL, 342
  - ENV\_TSQL\_USER, 342
  - LOCAL\_NODE, 323
  - NODE\_TAG, 321
  - SCM\_FOLD, 323
  - SCRAPER\_CFG, 275, 282, 431
  - eot (DybDbi.TimeStamp attribute), 367
  - Export() (DybPython.dbconf.DBConf class method), 342
  - export\_() (DybPython.dbconf.DBConf method), 342
  - extracondition (DybDbi.GCalibFeeSpec attribute), 386
  - extracondition (DybDbi.GCalibPmtSpec attribute), 382
  - extracondition (DybDbi.GDAQCalibRunInfo attribute), 399
  - extracondition (DybDbi.GDAQRawDataFileInfo attribute), 403
  - extracondition (DybDbi.GDAQRunInfo attribute), 394
  - extracondition (DybDbi.GDbiLogEntry attribute), 406
  - extracondition (DybDbi.GDcsAdTemp attribute), 409
  - extracondition (DybDbi.GDcsPmtHv attribute), 413
  - extracondition (DybDbi.GFeeCableMap attribute), 390
  - extracondition (DybDbi.GPhysAd attribute), 374
  - extracondition (DybDbi.GSimPmtSpec attribute), 378
- ## F
- fabseqno (DybPython.db.DB attribute), 330
  - fake() (Scraper.adtemp.AdTempFaker method), 424
  - fake() (Scraper.pmtHv.PmtHvFaker method), 423
  - Faker (class in Scraper.base), 428
  - feechanneldesc (DybDbi.GFeeCableMap attribute), 390
  - FeeChannelId (class in DybDbi), 370
  - feechannelid (DybDbi.GFeeCableMap attribute), 390
  - FeeHardwareId (class in DybDbi), 370
  - feehardwareid (DybDbi.GFeeCableMap attribute), 390
  - fieldnames (DybDbi.CSV attribute), 352
  - fields (DybDbi.GCalibFeeSpec attribute), 386
  - fields (DybDbi.GCalibPmtSpec attribute), 382
  - fields (DybDbi.GDAQCalibRunInfo attribute), 399
  - fields (DybDbi.GDAQRawDataFileInfo attribute), 403
  - fields (DybDbi.GDAQRunInfo attribute), 394
  - fields (DybDbi.GDbiLogEntry attribute), 406
  - fields (DybDbi.GDcsAdTemp attribute), 410
  - fields (DybDbi.GDcsPmtHv attribute), 413
  - fields (DybDbi.GFeeCableMap attribute), 390
  - fields (DybDbi.GPhysAd attribute), 374
  - fields (DybDbi.GSimPmtSpec attribute), 378
  - filename (DybDbi.GDAQRawDataFileInfo attribute), 403
  - fileno (DybDbi.GDAQRawDataFileInfo attribute), 403
  - filesize (DybDbi.GDAQRawDataFileInfo attribute), 403
  - filestate (DybDbi.GDAQRawDataFileInfo attribute), 403
  - Fill (DybDbi.GCalibFeeSpec attribute), 383
  - Fill (DybDbi.GCalibPmtSpec attribute), 379
  - Fill (DybDbi.GDAQCalibRunInfo attribute), 395
  - Fill (DybDbi.GDAQRawDataFileInfo attribute), 401
  - Fill (DybDbi.GDAQRunInfo attribute), 391
  - Fill (DybDbi.GDcsAdTemp attribute), 407
  - Fill (DybDbi.GDcsPmtHv attribute), 411
  - Fill (DybDbi.GFeeCableMap attribute), 388
  - Fill (DybDbi.GPhysAd attribute), 372
  - Fill (DybDbi.GSimPmtSpec attribute), 375
  - FloatValueForKey (DybDbi.GCalibFeeSpec attribute), 384
  - FloatValueForKey (DybDbi.GCalibPmtSpec attribute), 379
  - FloatValueForKey (DybDbi.GDAQCalibRunInfo attribute), 395
  - FloatValueForKey (DybDbi.GDAQRawDataFileInfo attribute), 401
  - FloatValueForKey (DybDbi.GDAQRunInfo attribute), 391
  - FloatValueForKey (DybDbi.GDbiLogEntry attribute), 404
  - FloatValueForKey (DybDbi.GDcsAdTemp attribute), 407
  - FloatValueForKey (DybDbi.GDcsPmtHv attribute), 411
  - FloatValueForKey (DybDbi.GFeeCableMap attribute), 388
  - FloatValueForKey (DybDbi.GPhysAd attribute), 372
  - FloatValueForKey (DybDbi.GSimPmtSpec attribute), 375
  - forced\_reloadcat\_() (DybPython.db.DB method), 331
  - fresh\_db() (DybPython.dbaux.Aux method), 340
  - from\_env() (DybPython.dbconf.DBConf class method), 342
  - FromIndex (DybDbi.Ctx attribute), 354
  - FromString (DybDbi.Ctx attribute), 354
  - FromString (DybDbi.DetectorId attribute), 369
  - FromString (DybDbi.SimFlag attribute), 369
  - FromString (DybDbi.Site attribute), 368
  - FromString0 (DybDbi.DetectorId attribute), 369
  - FullMask (DybDbi.Ctx attribute), 354
  - FullMask (DybDbi.SimFlag attribute), 369
  - FullMask (DybDbi.Site attribute), 368
- ## G
- gain (DybDbi.GSimPmtSpec attribute), 378
  - GCalibFeeSpec (class in DybDbi), 383
  - GCalibPmtSpec (class in DybDbi), 379
  - GDAQCalibRunInfo (class in DybDbi), 395
  - GDAQRawDataFileInfo (class in DybDbi), 400
  - GDAQRunInfo (class in DybDbi), 391
  - GDbiLogEntry (class in DybDbi), 404
  - GDcsAdTemp (class in DybDbi), 407
  - GDcsPmtHv (class in DybDbi), 410
  - get\_allseqno() (DybPython.db.DB method), 331
  - get\_attfn() (DybDbi.Wrap method), 351
  - get\_fabseqno() (DybPython.db.DB method), 331
  - get\_prep() (DybPython.dbcas.DD method), 344
  - get\_seqno() (DybPython.db.DB method), 331

- GetAdcPedestalHigh (DybDbi.GCalibFeeSpec attribute), 384
- GetAdcPedestalHighSigma (DybDbi.GCalibFeeSpec attribute), 384
- GetAdcPedestalLow (DybDbi.GCalibFeeSpec attribute), 384
- GetAdcPedestalLowSigma (DybDbi.GCalibFeeSpec attribute), 384
- GetAdcThresholdHigh (DybDbi.GCalibFeeSpec attribute), 384
- GetAdcThresholdLow (DybDbi.GCalibFeeSpec attribute), 384
- GetAdNo (DybDbi.GDaqCalibRunInfo attribute), 395
- GetAfterPulseProb (DybDbi.GCalibPmtSpec attribute), 379
- GetAfterPulseProb (DybDbi.GSimPmtSpec attribute), 375
- GetBaseVersion (DybDbi.GDaqRunInfo attribute), 391
- GetBOT (DybDbi.TimeStamp attribute), 366
- GetChanHrdwDesc (DybDbi.GFeeCableMap attribute), 388
- GetChannelId (DybDbi.GCalibFeeSpec attribute), 384
- GetChecksum (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetColumn (DybDbi.GDcsPmtHv attribute), 411
- GetDarkRate (DybDbi.GCalibPmtSpec attribute), 379
- GetDarkRate (DybDbi.GSimPmtSpec attribute), 375
- GetDatabaseLayout (DybDbi.GCalibFeeSpec attribute), 384
- GetDatabaseLayout (DybDbi.GCalibPmtSpec attribute), 379
- GetDatabaseLayout (DybDbi.GDaqCalibRunInfo attribute), 395
- GetDatabaseLayout (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetDatabaseLayout (DybDbi.GDaqRunInfo attribute), 392
- GetDatabaseLayout (DybDbi.GDcsAdTemp attribute), 408
- GetDatabaseLayout (DybDbi.GDcsPmtHv attribute), 411
- GetDatabaseLayout (DybDbi.GFeeCableMap attribute), 388
- GetDatabaseLayout (DybDbi.GPhysAd attribute), 372
- GetDatabaseLayout (DybDbi.GSimPmtSpec attribute), 375
- GetDataVersion (DybDbi.GDaqRunInfo attribute), 392
- GetDate (DybDbi.TimeStamp attribute), 366
- GetDescrib (DybDbi.GCalibPmtSpec attribute), 379
- GetDescrib (DybDbi.GSimPmtSpec attribute), 375
- GetDetectorId (DybDbi.GDaqCalibRunInfo attribute), 395
- GetDetectorMask (DybDbi.GDaqRunInfo attribute), 392
- GetDetId (DybDbi.Context attribute), 364
- GetDigest (DybDbi.GCalibFeeSpec attribute), 384
- GetDigest (DybDbi.GCalibPmtSpec attribute), 379
- GetDigest (DybDbi.GDaqCalibRunInfo attribute), 395
- GetDigest (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetDigest (DybDbi.GDaqRunInfo attribute), 392
- GetDigest (DybDbi.GDbiLogEntry attribute), 404
- GetDigest (DybDbi.GDcsAdTemp attribute), 408
- GetDigest (DybDbi.GDcsPmtHv attribute), 411
- GetDigest (DybDbi.GFeeCableMap attribute), 388
- GetDigest (DybDbi.GPhysAd attribute), 372
- GetDigest (DybDbi.GSimPmtSpec attribute), 375
- GetDuration (DybDbi.GDaqCalibRunInfo attribute), 396
- GetEfficiency (DybDbi.GCalibPmtSpec attribute), 379
- GetEfficiency (DybDbi.GSimPmtSpec attribute), 375
- GetEOT (DybDbi.TimeStamp attribute), 366
- GetFeeChannelDesc (DybDbi.GFeeCableMap attribute), 388
- GetFeeChannelId (DybDbi.GFeeCableMap attribute), 388
- GetFeeHardwareId (DybDbi.GFeeCableMap attribute), 388
- GetFields (DybDbi.GCalibFeeSpec attribute), 384
- GetFields (DybDbi.GCalibPmtSpec attribute), 380
- GetFields (DybDbi.GDaqCalibRunInfo attribute), 396
- GetFields (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetFields (DybDbi.GDaqRunInfo attribute), 392
- GetFields (DybDbi.GDbiLogEntry attribute), 404
- GetFields (DybDbi.GDcsAdTemp attribute), 408
- GetFields (DybDbi.GDcsPmtHv attribute), 411
- GetFields (DybDbi.GFeeCableMap attribute), 388
- GetFields (DybDbi.GPhysAd attribute), 372
- GetFields (DybDbi.GSimPmtSpec attribute), 375
- GetFileName (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetFileNo (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetFileSize (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetFileState (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetGain (DybDbi.GSimPmtSpec attribute), 375
- GetHomeA (DybDbi.GDaqCalibRunInfo attribute), 396
- GetHomeB (DybDbi.GDaqCalibRunInfo attribute), 396
- GetHomeC (DybDbi.GDaqCalibRunInfo attribute), 396
- GetLadder (DybDbi.GDcsPmtHv attribute), 411
- GetLedFreq (DybDbi.GDaqCalibRunInfo attribute), 396
- GetLedNumber1 (DybDbi.GDaqCalibRunInfo attribute), 396
- GetLedNumber2 (DybDbi.GDaqCalibRunInfo attribute), 396
- GetLedPulseSep (DybDbi.GDaqCalibRunInfo attribute), 396

- GetLedVoltage1 (DybDbi.GDaqCalibRunInfo attribute), 396
- GetLedVoltage2 (DybDbi.GDaqCalibRunInfo attribute), 396
- GetLtbMode (DybDbi.GDaqCalibRunInfo attribute), 396
- GetNanoSec (DybDbi.TimeStamp attribute), 366
- GetNBOT (DybDbi.TimeStamp attribute), 366
- GetPartitionName (DybDbi.GDaqRunInfo attribute), 392
- GetPhysAdId (DybDbi.GPhysAd attribute), 372
- GetPmtHardwareId (DybDbi.GFeeCableMap attribute), 388
- GetPmtHrdwDesc (DybDbi.GFeeCableMap attribute), 388
- GetPmtId (DybDbi.GCalibPmtSpec attribute), 380
- GetPmtId (DybDbi.GSimPmtSpec attribute), 376
- GetPrePulseProb (DybDbi.GCalibPmtSpec attribute), 380
- GetPrePulseProb (DybDbi.GSimPmtSpec attribute), 376
- GetPw (DybDbi.GDcsPmtHv attribute), 411
- GetRing (DybDbi.GDcsPmtHv attribute), 411
- GetRunNo (DybDbi.GDaqCalibRunInfo attribute), 396
- GetRunNo (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetRunNo (DybDbi.GDaqRunInfo attribute), 392
- GetRunType (DybDbi.GDaqRunInfo attribute), 392
- GetSchemaVersion (DybDbi.GDaqRunInfo attribute), 392
- GetSec (DybDbi.TimeStamp attribute), 366
- GetSeconds (DybDbi.TimeStamp attribute), 366
- GetSensorDesc (DybDbi.GFeeCableMap attribute), 388
- GetSensorId (DybDbi.GFeeCableMap attribute), 388
- GetSigmaGain (DybDbi.GSimPmtSpec attribute), 376
- GetSigmaSpeHigh (DybDbi.GCalibPmtSpec attribute), 380
- GetSimFlag (DybDbi.Context attribute), 364
- GetSimMask (DybDbi.ContextRange attribute), 364
- GetSite (DybDbi.Context attribute), 364
- GetSiteMask (DybDbi.ContextRange attribute), 365
- GetSourceIdA (DybDbi.GDaqCalibRunInfo attribute), 396
- GetSourceIdB (DybDbi.GDaqCalibRunInfo attribute), 396
- GetSourceIdC (DybDbi.GDaqCalibRunInfo attribute), 396
- GetSpeHigh (DybDbi.GCalibPmtSpec attribute), 380
- GetSpeLow (DybDbi.GCalibPmtSpec attribute), 380
- GetStatus (DybDbi.GCalibFeeSpec attribute), 384
- GetStatus (DybDbi.GCalibPmtSpec attribute), 380
- GetStream (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetStreamType (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetTableDescr (DybDbi.GCalibFeeSpec attribute), 384
- GetTableDescr (DybDbi.GCalibPmtSpec attribute), 380
- GetTableDescr (DybDbi.GDaqCalibRunInfo attribute), 396
- GetTableDescr (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetTableDescr (DybDbi.GDaqRunInfo attribute), 392
- GetTableDescr (DybDbi.GDcsAdTemp attribute), 408
- GetTableDescr (DybDbi.GDcsPmtHv attribute), 411
- GetTableDescr (DybDbi.GFeeCableMap attribute), 388
- GetTableDescr (DybDbi.GPhysAd attribute), 373
- GetTableDescr (DybDbi.GSimPmtSpec attribute), 376
- GetTableProxy (DybDbi.GCalibFeeSpec attribute), 384
- GetTableProxy (DybDbi.GCalibPmtSpec attribute), 380
- GetTableProxy (DybDbi.GDaqCalibRunInfo attribute), 396
- GetTableProxy (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetTableProxy (DybDbi.GDaqRunInfo attribute), 392
- GetTableProxy (DybDbi.GDbiLogEntry attribute), 404
- GetTableProxy (DybDbi.GDcsAdTemp attribute), 408
- GetTableProxy (DybDbi.GDcsPmtHv attribute), 411
- GetTableProxy (DybDbi.GFeeCableMap attribute), 388
- GetTableProxy (DybDbi.GPhysAd attribute), 373
- GetTableProxy (DybDbi.GSimPmtSpec attribute), 376
- GetTemp1 (DybDbi.GDcsAdTemp attribute), 408
- GetTemp2 (DybDbi.GDcsAdTemp attribute), 408
- GetTemp3 (DybDbi.GDcsAdTemp attribute), 408
- GetTemp4 (DybDbi.GDcsAdTemp attribute), 408
- GetTime (DybDbi.TimeStamp attribute), 366
- GetTimeEnd (DybDbi.ContextRange attribute), 365
- GetTimeGate (DybDbi.Dbi attribute), 370
- GetTimeOffset (DybDbi.GCalibPmtSpec attribute), 380
- GetTimeOffset (DybDbi.GSimPmtSpec attribute), 376
- GetTimeSpec (DybDbi.TimeStamp attribute), 366
- GetTimeSpread (DybDbi.GCalibPmtSpec attribute), 380
- GetTimeSpread (DybDbi.GSimPmtSpec attribute), 376
- GetTimeStamp (DybDbi.Context attribute), 364
- GetTimeStart (DybDbi.ContextRange attribute), 365
- GetTransferState (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetTriggerType (DybDbi.GDaqRunInfo attribute), 392
- GetValues (DybDbi.GCalibFeeSpec attribute), 384
- GetValues (DybDbi.GCalibPmtSpec attribute), 380
- GetValues (DybDbi.GDaqCalibRunInfo attribute), 396
- GetValues (DybDbi.GDaqRawDataFileInfo attribute), 401
- GetValues (DybDbi.GDaqRunInfo attribute), 392
- GetValues (DybDbi.GDbiLogEntry attribute), 404
- GetValues (DybDbi.GDcsAdTemp attribute), 408
- GetValues (DybDbi.GDcsPmtHv attribute), 411
- GetValues (DybDbi.GFeeCableMap attribute), 388
- GetValues (DybDbi.GPhysAd attribute), 373
- GetValues (DybDbi.GSimPmtSpec attribute), 376
- GetVldDescr (DybDbi.Dbi attribute), 370
- GetVoltage (DybDbi.GDcsPmtHv attribute), 411

GetZoneOffset (DybDbi.TimeStamp attribute), 367  
 GetZPositionA (DybDbi.GDaqCalibRunInfo attribute), 396  
 GetZPositionB (DybDbi.GDaqCalibRunInfo attribute), 396  
 GetZPositionC (DybDbi.GDaqCalibRunInfo attribute), 396  
 GFeeCableMap (class in DybDbi), 387  
 GPhysAd (class in DybDbi), 371  
 grow\_cf() (in module DybDbi.vld.vsmry), 361  
 GSimPmtSpec (class in DybDbi), 374

## H

has\_config() (DybPython.dbconf.DBConf class method), 343  
 has\_table() (DybPython.db.DB method), 332  
 homea (DybDbi.GDaqCalibRunInfo attribute), 399  
 homeb (DybDbi.GDaqCalibRunInfo attribute), 399  
 homec (DybDbi.GDaqCalibRunInfo attribute), 399  
 hostname (DybDbi.GDbiLogEntry attribute), 406

## I

iinst() (Scraper.base.sourcevector.SourceVector method), 429  
 ILookup (class in DybDbi), 361  
 info (DybPython.dbaux.Aux attribute), 340  
 initialize() (Scraper.base.Regime method), 426  
 instance() (Scraper.base.Target method), 427  
 IntValueForKey (DybDbi.GCalibFeeSpec attribute), 384  
 IntValueForKey (DybDbi.GCalibPmtSpec attribute), 380  
 IntValueForKey (DybDbi.GDaqCalibRunInfo attribute), 397  
 IntValueForKey (DybDbi.GDaqRawDataFileInfo attribute), 401  
 IntValueForKey (DybDbi.GDaqRunInfo attribute), 392  
 IntValueForKey (DybDbi.GDbiLogEntry attribute), 405  
 IntValueForKey (DybDbi.GDcsAdTemp attribute), 408  
 IntValueForKey (DybDbi.GDcsPmtHv attribute), 411  
 IntValueForKey (DybDbi.GFeeCableMap attribute), 388  
 IntValueForKey (DybDbi.GPhysAd attribute), 373  
 IntValueForKey (DybDbi.GSimPmtSpec attribute), 376  
 IRunLookup (class in DybDbi), 361  
 is\_descline() (DybDbi.Source method), 353  
 IsA (DybDbi.Context attribute), 364  
 IsA (DybDbi.ContextRange attribute), 365  
 IsA (DybDbi.GCalibFeeSpec attribute), 384  
 IsA (DybDbi.GCalibPmtSpec attribute), 380  
 IsA (DybDbi.GDaqCalibRunInfo attribute), 397  
 IsA (DybDbi.GDaqRawDataFileInfo attribute), 402  
 IsA (DybDbi.GDaqRunInfo attribute), 392  
 IsA (DybDbi.GDbiLogEntry attribute), 405  
 IsA (DybDbi.GDcsAdTemp attribute), 408  
 IsA (DybDbi.GDcsPmtHv attribute), 411  
 IsA (DybDbi.GFeeCableMap attribute), 388

IsA (DybDbi.GPhysAd attribute), 373  
 IsA (DybDbi.GSimPmtSpec attribute), 376  
 IsA (DybDbi.ServiceMode attribute), 368  
 IsA (DybDbi.TimeStamp attribute), 367  
 isAD (DybDbi.DetectorId attribute), 369  
 IsCompatible (DybDbi.ContextRange attribute), 365  
 IsLeapYear (DybDbi.TimeStamp attribute), 367  
 IsNull (DybDbi.TimeStamp attribute), 367  
 isRPC (DybDbi.DetectorId attribute), 369  
 IsValid (DybDbi.Context attribute), 364  
 isWaterShield (DybDbi.DetectorId attribute), 369

## K

KeyList (class in dybtest.cfroot), 432  
 kls (DybDbi.AdLogicalPhysical attribute), 363  
 kls() (Scraper.base.sa.SA method), 431  
 kNow() (DybDbi.TimeStamp class method), 367  
 known\_input\_type() (DybPython.Control.NuWa method), 415

## L

ladder (DybDbi.GDcsPmtHv attribute), 413  
 lag() (Scraper.base.sourcevector.SourceVector method), 429  
 lastentry() (Scraper.base.sourcevector.SourceVector method), 429  
 lastresult\_ (Scraper.base.sourcevector.SourceVector attribute), 429  
 lastvld() (Scraper.base.Target method), 428  
 ledfreq (DybDbi.GDaqCalibRunInfo attribute), 399  
 lednumber1 (DybDbi.GDaqCalibRunInfo attribute), 399  
 lednumber2 (DybDbi.GDaqCalibRunInfo attribute), 399  
 ledpulsesep (DybDbi.GDaqCalibRunInfo attribute), 399  
 ledvoltage1 (DybDbi.GDaqCalibRunInfo attribute), 399  
 ledvoltage2 (DybDbi.GDaqCalibRunInfo attribute), 399  
 Length (DybDbi.Ctx attribute), 354  
 load\_() (DybPython.db.DB method), 332  
 loadcsv() (DybPython.db.DB method), 332  
 LOCAL\_NODE, 323  
 logging\_() (DybPython.dbicnf.DbiCnf method), 416  
 lognumseqno (DybDbi.GDbiLogEntry attribute), 406  
 logseqnomax (DybDbi.GDbiLogEntry attribute), 406  
 logseqnomin (DybDbi.GDbiLogEntry attribute), 406  
 logtablename (DybDbi.GDbiLogEntry attribute), 406  
 lookup\_logical2physical() (DybDbi.AdLogicalPhysical class method), 363  
 ls\_() (DybPython.dbaux.Aux method), 340  
 ltbmode (DybDbi.GDaqCalibRunInfo attribute), 399

## M

main() (in module Scraper.base), 425  
 make\_\_repr\_\_() (DybDbi.Wrap method), 351  
 MakeDateTimeString (DybDbi.Dbi attribute), 370  
 MakeTimeStamp (DybDbi.Dbi attribute), 370

Mapper (class in DybDbi), 354  
 MaskFromString (DybDbi.Ctx attribute), 354  
 MaskFromString (DybDbi.Site attribute), 368  
 MaxBits (DybDbi.Ctx attribute), 354  
 MetaDB (class in NonDbi), 419  
 MktimeFromUTC (DybDbi.TimeStamp attribute), 367  
 mysql() (DybPython.db.DB method), 332  
 mysqldb\_parameters() (DybPython.dbconf.DBConf method), 343

## N

name (DybDbi.GCalibFeeSpec attribute), 386  
 name (DybDbi.GCalibPmtSpec attribute), 382  
 name (DybDbi.GDAQCalibRunInfo attribute), 399  
 name (DybDbi.GDAQRawDataFileInfo attribute), 404  
 name (DybDbi.GDAQRunInfo attribute), 394  
 name (DybDbi.GDbiLogEntry attribute), 406  
 name (DybDbi.GDcsAdTemp attribute), 410  
 name (DybDbi.GDcsPmtHv attribute), 413  
 name (DybDbi.GFeeCableMap attribute), 390  
 name (DybDbi.GPhysAd attribute), 374  
 name (DybDbi.GSimPmtSpec attribute), 378  
 nanosec (DybDbi.TimeStamp attribute), 367  
 nbot (DybDbi.TimeStamp attribute), 367  
 next() (DybDbi.Source method), 353  
 NODE\_TAG, 321  
 NonDbi (module), 417  
 noop\_() (DybPython.db.DB method), 332  
 NotGlobalSeqNo (DybDbi.Dbi attribute), 370  
 NuWa (class in DybPython.Control), 414

## O

optables (DybPython.db.DB attribute), 332  
 outfile() (DybPython.db.DB method), 333

## P

parse\_path() (DybPython.dbicnf.DbiCnf method), 416  
 Parser (class in Scraper.base.parser), 430  
 partitionname (DybDbi.GDAQRunInfo attribute), 394  
 paytables (DybPython.db.DB attribute), 333  
 physadid (DybDbi.GPhysAd attribute), 374  
 PmtHardwareId (class in DybDbi), 370  
 pmthardwareid (DybDbi.GFeeCableMap attribute), 390  
 pmthrdwdesc (DybDbi.GFeeCableMap attribute), 390  
 PmtHv (class in Scraper.pmtHV), 421  
 PmtHvFaker (class in Scraper.pmtHV), 423  
 PmtHvScraper (class in Scraper.pmtHV), 422  
 PmtHvSource (class in Scraper.pmtHV), 422  
 pmtid (DybDbi.GCalibPmtSpec attribute), 382  
 pmtid (DybDbi.GSimPmtSpec attribute), 378  
 predump() (DybPython.db.DB method), 333  
 prep (DybPython.dbconf.DD attribute), 344  
 prepulseprob (DybDbi.GCalibPmtSpec attribute), 382  
 prepulseprob (DybDbi.GSimPmtSpec attribute), 378

present\_smry() (in module DybDbi.vld.vsmry), 361  
 prime\_parser() (DybPython.dbconf.DBConf class method), 343  
 Print (DybDbi.TimeStamp attribute), 367  
 process() (DybPython.dbconf.DBConf method), 343  
 processname (DybDbi.GDbiLogEntry attribute), 406  
 propagate() (Scraper.adtemp.AdTempScraper method), 424  
 propagate() (Scraper.base.Scraper method), 427  
 propagate() (Scraper.pmtHV.PmtHvScraper method), 422  
 pw (DybDbi.GDcsPmtHv attribute), 413

## Q

qafter() (Scraper.base.DCS method), 426  
 qbefore() (Scraper.base.DCS method), 426

## R

rcmpcat\_() (DybPython.db.DB method), 333  
 rcmpcat\_() (DybPython.dbaux.Aux method), 340  
 rdumppcat\_() (DybPython.db.DB method), 333  
 read\_cfg() (DybPython.dbconf.DBConf class method), 343  
 read\_desc() (DybPython.db.DB method), 334  
 read\_seqno() (DybPython.db.DB method), 334  
 reason (DybDbi.GDbiLogEntry attribute), 406  
 reflect() (Scraper.base.sa.SA method), 431  
 Regime (class in Scraper.base), 426  
 ring (DybDbi.GDcsPmtHv attribute), 413  
 rloadcat\_() (DybPython.db.DB method), 334  
 rloadcat\_() (DybPython.dbaux.Aux method), 340  
 Rpt (DybDbi.GCalibFeeSpec attribute), 384  
 Rpt (DybDbi.GCalibPmtSpec attribute), 380  
 Rpt (DybDbi.GDAQCalibRunInfo attribute), 397  
 Rpt (DybDbi.GDAQRawDataFileInfo attribute), 402  
 Rpt (DybDbi.GDAQRunInfo attribute), 392  
 Rpt (DybDbi.GDbiLogEntry attribute), 405  
 Rpt (DybDbi.GDcsAdTemp attribute), 408  
 Rpt (DybDbi.GDcsPmtHv attribute), 412  
 Rpt (DybDbi.GFeeCableMap attribute), 388  
 Rpt (DybDbi.GPhysAd attribute), 373  
 Rpt (DybDbi.GSimPmtSpec attribute), 376  
 run\_post\_user() (DybPython.Control.NuWa method), 415  
 runno (DybDbi.GDAQCalibRunInfo attribute), 400  
 runno (DybDbi.GDAQRawDataFileInfo attribute), 404  
 runno (DybDbi.GDAQRunInfo attribute), 394  
 runtime (DybDbi.GDAQRunInfo attribute), 394

## S

SA (class in Scraper.base.sa), 431  
 Save (DybDbi.GCalibFeeSpec attribute), 384  
 Save (DybDbi.GCalibPmtSpec attribute), 380  
 Save (DybDbi.GDAQCalibRunInfo attribute), 397  
 Save (DybDbi.GDAQRawDataFileInfo attribute), 402

- Save (DybDbi.GDaqRunInfo attribute), 392
- Save (DybDbi.GDbiLogEntry attribute), 405
- Save (DybDbi.GDcsAdTemp attribute), 408
- Save (DybDbi.GDcsPmtHv attribute), 412
- Save (DybDbi.GFeeCableMap attribute), 389
- Save (DybDbi.GPhysAd attribute), 373
- Save (DybDbi.GSimPmtSpec attribute), 376
- schemaversion (DybDbi.GDaqRunInfo attribute), 394
- SCM\_FOLD, 323
- Scraper (class in Scraper.base), 426
- Scraper (module), 421
- Scraper.adlidsensor (module), 424
- Scraper.adtemp (module), 423
- Scraper.dcs (module), 425
- Scraper.pmthv (module), 421
- SCRAPER\_CFG, 275, 282, 431
- sec (DybDbi.TimeStamp attribute), 367
- seconds (DybDbi.TimeStamp attribute), 367
- seed() (Scraper.base.Target method), 428
- seed() (Scraper.pmthv.PmtHvScraper method), 422
- sensordesc (DybDbi.GFeeCableMap attribute), 390
- sensorid (DybDbi.GFeeCableMap attribute), 391
- seqno (DybPython.db.DB attribute), 335
- server (DybPython.dbcas.DBCon attribute), 344
- servername (DybDbi.GDbiLogEntry attribute), 406
- ServiceMode (class in DybDbi), 368
- session() (NonDbi.MetaDB method), 420
- session\_() (in module NonDbi), 420
- set\_cursor() (Scraper.base.sourcevector.SourceVector method), 429
- SetAdcPedestalHigh (DybDbi.GCalibFeeSpec attribute), 384
- SetAdcPedestalHighSigma (DybDbi.GCalibFeeSpec attribute), 384
- SetAdcPedestalLow (DybDbi.GCalibFeeSpec attribute), 385
- SetAdcPedestalLowSigma (DybDbi.GCalibFeeSpec attribute), 385
- SetAdcThresholdHigh (DybDbi.GCalibFeeSpec attribute), 385
- SetAdcThresholdLow (DybDbi.GCalibFeeSpec attribute), 385
- SetAdNo (DybDbi.GDaqCalibRunInfo attribute), 397
- SetAfterPulseProb (DybDbi.GCalibPmtSpec attribute), 380
- SetAfterPulseProb (DybDbi.GSimPmtSpec attribute), 376
- SetBaseVersion (DybDbi.GDaqRunInfo attribute), 392
- SetChanHrdwDesc (DybDbi.GFeeCableMap attribute), 389
- SetChannelId (DybDbi.GCalibFeeSpec attribute), 385
- SetChecksum (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetColumn (DybDbi.GDcsPmtHv attribute), 412
- SetDarkRate (DybDbi.GCalibPmtSpec attribute), 380
- SetDarkRate (DybDbi.GSimPmtSpec attribute), 376
- SetDataVersion (DybDbi.GDaqRunInfo attribute), 392
- SetDescrib (DybDbi.GCalibPmtSpec attribute), 380
- SetDescrib (DybDbi.GSimPmtSpec attribute), 376
- SetDetectorId (DybDbi.GDaqCalibRunInfo attribute), 397
- SetDetectorMask (DybDbi.GDaqRunInfo attribute), 392
- SetDetId (DybDbi.Context attribute), 364
- SetDuration (DybDbi.GDaqCalibRunInfo attribute), 397
- SetEfficiency (DybDbi.GCalibPmtSpec attribute), 380
- SetEfficiency (DybDbi.GSimPmtSpec attribute), 376
- SetFeeChannelDesc (DybDbi.GFeeCableMap attribute), 389
- SetFeeChannelId (DybDbi.GFeeCableMap attribute), 389
- SetFeeHardwareId (DybDbi.GFeeCableMap attribute), 389
- SetFileName (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetFileNo (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetFileSize (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetFileState (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetGain (DybDbi.GSimPmtSpec attribute), 376
- SetHomeA (DybDbi.GDaqCalibRunInfo attribute), 397
- SetHomeB (DybDbi.GDaqCalibRunInfo attribute), 397
- SetHomeC (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLadder (DybDbi.GDcsPmtHv attribute), 412
- SetLedFreq (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLedNumber1 (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLedNumber2 (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLedPulseSep (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLedVoltage1 (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLedVoltage2 (DybDbi.GDaqCalibRunInfo attribute), 397
- SetLtbMode (DybDbi.GDaqCalibRunInfo attribute), 397
- SetPartitionName (DybDbi.GDaqRunInfo attribute), 392
- SetPhysAdId (DybDbi.GPhysAd attribute), 373
- SetPmtHardwareId (DybDbi.GFeeCableMap attribute), 389
- SetPmtHrdwDesc (DybDbi.GFeeCableMap attribute), 389
- SetPmtId (DybDbi.GCalibPmtSpec attribute), 380
- SetPmtId (DybDbi.GSimPmtSpec attribute), 376
- SetPrePulseProb (DybDbi.GCalibPmtSpec attribute), 381
- SetPrePulseProb (DybDbi.GSimPmtSpec attribute), 376
- SetPw (DybDbi.GDcsPmtHv attribute), 412

- SetRing (DybDbi.GDcsPmtHv attribute), 412
- SetRunNo (DybDbi.GDaqCalibRunInfo attribute), 397
- SetRunNo (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetRunNo (DybDbi.GDaqRunInfo attribute), 393
- SetRunType (DybDbi.GDaqRunInfo attribute), 393
- SetSchemaVersion (DybDbi.GDaqRunInfo attribute), 393
- SetSensorDesc (DybDbi.GFeeCableMap attribute), 389
- SetSensorId (DybDbi.GFeeCableMap attribute), 389
- SetSigmaGain (DybDbi.GSimPmtSpec attribute), 376
- SetSigmaSpeHigh (DybDbi.GCalibPmtSpec attribute), 381
- setsignals() (Scraper.base.Regime method), 426
- SetSimFlag (DybDbi.Context attribute), 364
- SetSimMask (DybDbi.ContextRange attribute), 365
- SetSite (DybDbi.Context attribute), 364
- SetSiteMask (DybDbi.ContextRange attribute), 365
- SetSourceIdA (DybDbi.GDaqCalibRunInfo attribute), 397
- SetSourceIdB (DybDbi.GDaqCalibRunInfo attribute), 397
- SetSourceIdC (DybDbi.GDaqCalibRunInfo attribute), 397
- SetSpeHigh (DybDbi.GCalibPmtSpec attribute), 381
- SetSpeLow (DybDbi.GCalibPmtSpec attribute), 381
- SetStatus (DybDbi.GCalibFeeSpec attribute), 385
- SetStatus (DybDbi.GCalibPmtSpec attribute), 381
- SetStream (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetStreamType (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetTemp1 (DybDbi.GDcsAdTemp attribute), 408
- SetTemp2 (DybDbi.GDcsAdTemp attribute), 408
- SetTemp3 (DybDbi.GDcsAdTemp attribute), 408
- SetTemp4 (DybDbi.GDcsAdTemp attribute), 408
- SetTimeEnd (DybDbi.ContextRange attribute), 365
- SetTimeGate (DybDbi.Dbi attribute), 370
- SetTimeOffset (DybDbi.GCalibPmtSpec attribute), 381
- SetTimeOffset (DybDbi.GSimPmtSpec attribute), 376
- SetTimeSpread (DybDbi.GCalibPmtSpec attribute), 381
- SetTimeSpread (DybDbi.GSimPmtSpec attribute), 377
- SetTimeStamp (DybDbi.Context attribute), 364
- SetTimeStart (DybDbi.ContextRange attribute), 365
- SetTransferState (DybDbi.GDaqRawDataFileInfo attribute), 402
- SetTriggerType (DybDbi.GDaqRunInfo attribute), 393
- setup() (in module DybDbi.vld.versiondate), 357
- SetVoltage (DybDbi.GDcsPmtHv attribute), 412
- SetZPositionA (DybDbi.GDaqCalibRunInfo attribute), 398
- SetZPositionB (DybDbi.GDaqCalibRunInfo attribute), 398
- SetZPositionC (DybDbi.GDaqCalibRunInfo attribute), 398
- ShowMembers (DybDbi.Context attribute), 364
- ShowMembers (DybDbi.ContextRange attribute), 365
- ShowMembers (DybDbi.GCalibFeeSpec attribute), 385
- ShowMembers (DybDbi.GCalibPmtSpec attribute), 381
- ShowMembers (DybDbi.GDaqCalibRunInfo attribute), 398
- ShowMembers (DybDbi.GDaqRawDataFileInfo attribute), 402
- ShowMembers (DybDbi.GDaqRunInfo attribute), 393
- ShowMembers (DybDbi.GDbiLogEntry attribute), 405
- ShowMembers (DybDbi.GDcsAdTemp attribute), 408
- ShowMembers (DybDbi.GDcsPmtHv attribute), 412
- ShowMembers (DybDbi.GFeeCableMap attribute), 389
- ShowMembers (DybDbi.GPhysAd attribute), 373
- ShowMembers (DybDbi.GSimPmtSpec attribute), 377
- ShowMembers (DybDbi.ServiceMode attribute), 368
- ShowMembers (DybDbi.TimeStamp attribute), 367
- showpaytables (DybPython.db.DB attribute), 335
- showtables (DybPython.db.DB attribute), 335
- sigmagain (DybDbi.GSimPmtSpec attribute), 378
- sigmaspehigh (DybDbi.GCalibPmtSpec attribute), 382
- SimFlag (class in DybDbi), 369
- simflag (DybDbi.Context attribute), 364
- simflag (DybPython.dbicnf.DbiCnf attribute), 417
- simmask (DybDbi.ContextRange attribute), 365
- simmask (DybDbi.GDbiLogEntry attribute), 406
- simmask (DybPython.dbicnf.DbiCnf attribute), 417
- Site (class in DybDbi), 368
- site (DybDbi.Context attribute), 364
- site (DybPython.dbicnf.DbiCnf attribute), 417
- sitemask (DybDbi.ContextRange attribute), 365
- sitemask (DybDbi.GDbiLogEntry attribute), 406
- sitemask (DybPython.dbicnf.DbiCnf attribute), 417
- smry() (Scraper.base.sourcevector.SourceVector method), 429
- Source (class in DybDbi), 352
- sourceida (DybDbi.GDaqCalibRunInfo attribute), 400
- sourceidb (DybDbi.GDaqCalibRunInfo attribute), 400
- sourceidc (DybDbi.GDaqCalibRunInfo attribute), 400
- SourceVector (class in Scraper.base.sourcevector), 429
- spawn() (DybPython.dbcas.DBCas method), 343
- spawn() (DybPython.dbcas.DBCon method), 344
- SpecKeys (DybDbi.GCalibFeeSpec attribute), 385
- SpecKeys (DybDbi.GCalibPmtSpec attribute), 381
- SpecKeys (DybDbi.GDaqCalibRunInfo attribute), 398
- SpecKeys (DybDbi.GDaqRawDataFileInfo attribute), 402
- SpecKeys (DybDbi.GDaqRunInfo attribute), 393
- SpecKeys (DybDbi.GDcsAdTemp attribute), 408
- SpecKeys (DybDbi.GDcsPmtHv attribute), 412
- SpecKeys (DybDbi.GFeeCableMap attribute), 389
- SpecKeys (DybDbi.GPhysAd attribute), 373

SpecKeys (DybDbi.GSimPmtSpec attribute), 377  
 SpecList (DybDbi.GCalibFeeSpec attribute), 385  
 SpecList (DybDbi.GCalibPmtSpec attribute), 381  
 SpecList (DybDbi.GDaqCalibRunInfo attribute), 398  
 SpecList (DybDbi.GDaqRawDataFileInfo attribute), 402  
 SpecList (DybDbi.GDaqRunInfo attribute), 393  
 SpecList (DybDbi.GDcsAdTemp attribute), 408  
 SpecList (DybDbi.GDcsPmtHv attribute), 412  
 SpecList (DybDbi.GFeeCableMap attribute), 389  
 SpecList (DybDbi.GPhysAd attribute), 373  
 SpecList (DybDbi.GSimPmtSpec attribute), 377  
 SpecMap (DybDbi.GCalibFeeSpec attribute), 385  
 SpecMap (DybDbi.GCalibPmtSpec attribute), 381  
 SpecMap (DybDbi.GDaqCalibRunInfo attribute), 398  
 SpecMap (DybDbi.GDaqRawDataFileInfo attribute), 402  
 SpecMap (DybDbi.GDaqRunInfo attribute), 393  
 SpecMap (DybDbi.GDcsAdTemp attribute), 409  
 SpecMap (DybDbi.GDcsPmtHv attribute), 412  
 SpecMap (DybDbi.GFeeCableMap attribute), 389  
 SpecMap (DybDbi.GPhysAd attribute), 373  
 SpecMap (DybDbi.GSimPmtSpec attribute), 377  
 spehigh (DybDbi.GCalibPmtSpec attribute), 382  
 spelow (DybDbi.GCalibPmtSpec attribute), 382  
 squeeze\_tab() (in module DybDbi.vld.vsmry), 361  
 stat (DybPython.dbaux.Aux attribute), 340  
 status (DybDbi.GCalibFeeSpec attribute), 386  
 status (DybDbi.GCalibPmtSpec attribute), 382  
 status (Scraper.base.sourcevector.SourceVector attribute), 430  
 status\_ (Scraper.base.sourcevector.SourceVector attribute), 430  
 Store (DybDbi.GCalibFeeSpec attribute), 385  
 Store (DybDbi.GCalibPmtSpec attribute), 381  
 Store (DybDbi.GDaqCalibRunInfo attribute), 398  
 Store (DybDbi.GDaqRawDataFileInfo attribute), 402  
 Store (DybDbi.GDaqRunInfo attribute), 393  
 Store (DybDbi.GDcsAdTemp attribute), 409  
 Store (DybDbi.GDcsPmtHv attribute), 412  
 Store (DybDbi.GFeeCableMap attribute), 389  
 Store (DybDbi.GPhysAd attribute), 373  
 Store (DybDbi.GSimPmtSpec attribute), 377  
 stream (DybDbi.GDaqRawDataFileInfo attribute), 404  
 streamtype (DybDbi.GDaqRawDataFileInfo attribute), 404  
 StringForIndex (DybDbi.Ctx attribute), 354  
 StringFromMask (DybDbi.Ctx attribute), 354  
 StringFromMask (DybDbi.SimFlag attribute), 369  
 StringFromMask (DybDbi.Site attribute), 368  
 subbase() (Scraper.base.DCS method), 426  
 subsite (DybDbi.GDbiLogEntry attribute), 406  
 subsite (DybPython.dbicnf.DbiCnf attribute), 417  
 Subtract (DybDbi.TimeStamp attribute), 367  
 svnup\_() (DybPython.dbaux.Aux method), 340

## T

Tab (class in DybDbiPre), 348  
 tab() (DybPython.db.DB method), 336  
 tabfile() (DybPython.db.DB method), 336  
 table() (Scraper.base.sa.SA method), 431  
 tabledescr (DybDbi.GCalibFeeSpec attribute), 386  
 tabledescr (DybDbi.GCalibPmtSpec attribute), 383  
 tabledescr (DybDbi.GDaqCalibRunInfo attribute), 400  
 tabledescr (DybDbi.GDaqRawDataFileInfo attribute), 404  
 tabledescr (DybDbi.GDaqRunInfo attribute), 394  
 tabledescr (DybDbi.GDcsAdTemp attribute), 410  
 tabledescr (DybDbi.GDcsPmtHv attribute), 413  
 tabledescr (DybDbi.GFeeCableMap attribute), 391  
 tabledescr (DybDbi.GPhysAd attribute), 374  
 tabledescr (DybDbi.GSimPmtSpec attribute), 378  
 tableproxy (DybDbi.GCalibFeeSpec attribute), 386  
 tableproxy (DybDbi.GCalibPmtSpec attribute), 383  
 tableproxy (DybDbi.GDaqCalibRunInfo attribute), 400  
 tableproxy (DybDbi.GDaqRawDataFileInfo attribute), 404  
 tableproxy (DybDbi.GDaqRunInfo attribute), 394  
 tableproxy (DybDbi.GDbiLogEntry attribute), 406  
 tableproxy (DybDbi.GDcsAdTemp attribute), 410  
 tableproxy (DybDbi.GDcsPmtHv attribute), 413  
 tableproxy (DybDbi.GFeeCableMap attribute), 391  
 tableproxy (DybDbi.GPhysAd attribute), 374  
 tableproxy (DybDbi.GSimPmtSpec attribute), 378  
 tables (DybPython.db.DB attribute), 336  
 Target (class in Scraper.base), 427  
 task (DybDbi.GDbiLogEntry attribute), 406  
 task (DybDbi.ServiceMode attribute), 368  
 tcursor (Scraper.base.sourcevector.SourceVector attribute), 430  
 temp1 (DybDbi.GDcsAdTemp attribute), 410  
 temp2 (DybDbi.GDcsAdTemp attribute), 410  
 temp3 (DybDbi.GDcsAdTemp attribute), 410  
 temp4 (DybDbi.GDcsAdTemp attribute), 410  
 time (DybDbi.TimeStamp attribute), 368  
 TimeAction (class in DybPython.dbicnf), 417  
 timeend (DybDbi.ContextRange attribute), 365  
 timeend (DybPython.dbicnf.DbiCnf attribute), 417  
 timegate (DybDbi.Dbi attribute), 370  
 timeoffset (DybDbi.GCalibPmtSpec attribute), 383  
 timeoffset (DybDbi.GSimPmtSpec attribute), 378  
 timespec (DybDbi.TimeStamp attribute), 368  
 timespread (DybDbi.GCalibPmtSpec attribute), 383  
 timespread (DybDbi.GSimPmtSpec attribute), 378  
 TimeStamp (class in DybDbi), 365  
 timestamp (DybDbi.Context attribute), 364  
 timestart (DybDbi.ContextRange attribute), 365  
 timestart (DybPython.dbicnf.DbiCnf attribute), 417  
 TKey\_GetCoords() (in module dybtest.cfoot), 433  
 TKey\_GetIdentity() (in module dybtest.cfoot), 433

tmpdir (DybPython.db.DB attribute), 336  
 tmpfold (DybPython.db.DB attribute), 336  
 transferstate (DybDbi.GDaqRawDataFileInfo attribute), 404  
 transfix() (in module DybDbi.vld.versiondate), 357  
 transfix\_tab() (in module DybDbi.vld.versiondate), 357  
 traverse\_vlut() (in module DybDbi.vld.vlut), 359  
 triggerstype (DybDbi.GDaqRunInfo attribute), 394  
 TrimTo (DybDbi.ContextRange attribute), 365  
 tunesleep() (Scraper.base.Scraper method), 427

## U

updatetime (DybDbi.GDbiLogEntry attribute), 406  
 username (DybDbi.GDbiLogEntry attribute), 406  
 UsernameFromEnvironment (DybDbi.Dbi attribute), 370  
 UTCtoDatetime (DybDbi.TimeStamp attribute), 367  
 UTCtoNaiveLocalDatetime (DybDbi.TimeStamp attribute), 367

## V

validate\_hunk() (DybPython.dbsvn.DBValidate method), 348  
 validate\_update() (DybPython.dbsvn.DBValidate method), 348  
 validate\_validity() (DybPython.dbsvn.DBValidate method), 348  
 values (DybDbi.GCalibFeeSpec attribute), 387  
 values (DybDbi.GCalibPmtSpec attribute), 383  
 values (DybDbi.GDaqCalibRunInfo attribute), 400  
 values (DybDbi.GDaqRawDataFileInfo attribute), 404  
 values (DybDbi.GDaqRunInfo attribute), 394  
 values (DybDbi.GDbiLogEntry attribute), 406  
 values (DybDbi.GDcsAdTemp attribute), 410  
 values (DybDbi.GDcsPmtHv attribute), 413  
 values (DybDbi.GFeeCableMap attribute), 391  
 values (DybDbi.GPhysAd attribute), 374  
 values (DybDbi.GSimPmtSpec attribute), 378  
 vdupe() (DybPython.db.DB method), 336  
 vdupe\_() (DybPython.db.DB method), 336  
 VFS (class in DybDbi.vld.versiondate), 356  
 vlldescr (DybDbi.Dbi attribute), 370  
 voltage (DybDbi.GDcsPmtHv attribute), 413  
 vsssta() (DybPython.db.DB method), 336

## W

wipe\_cache() (DybPython.db.DB method), 336  
 Wrap (class in DybDbi), 350  
 write() (DybDbi.AdLogicalPhysical method), 363  
 write() (DybDbi.CSV method), 352  
 writer() (DybPython.dbicnf.DbiCnf method), 417  
 writer() (Scraper.base.Target method), 428  
 Wrt (DybDbi.GCalibFeeSpec attribute), 385  
 Wrt (DybDbi.GCalibPmtSpec attribute), 381  
 Wrt (DybDbi.GDaqCalibRunInfo attribute), 398

Wrt (DybDbi.GDaqRawDataFileInfo attribute), 402  
 Wrt (DybDbi.GDaqRunInfo attribute), 393  
 Wrt (DybDbi.GDbiLogEntry attribute), 405  
 Wrt (DybDbi.GDcsAdTemp attribute), 409  
 Wrt (DybDbi.GDcsPmtHv attribute), 412  
 Wrt (DybDbi.GFeeCableMap attribute), 389  
 Wrt (DybDbi.GPhysAd attribute), 373  
 Wrt (DybDbi.GSimPmtSpec attribute), 377

## Z

zoneoffset (DybDbi.TimeStamp attribute), 368  
 zpositiona (DybDbi.GDaqCalibRunInfo attribute), 400  
 zpositionb (DybDbi.GDaqCalibRunInfo attribute), 400  
 zpositionc (DybDbi.GDaqCalibRunInfo attribute), 400