Databases for G_E^n
Draft 0.2

Robert J. Feuerbach
Jefferson Lab

July 12, 2006

Abstract

To track the quality of and analyze the data, there is a need for at least two additional collections of information. First is a "run-conditions" database, which tracks the state of the experiment during data-taking (beam energy, detector angles, target polarization, half-wave plate setting, etc) to be used to construct asymmetries from each run and combine them to extract the physics asymmetry. The second is a "calibration" database, that contains information needed to process the raw detector responses; this database contains information such as detector geometries, calibration constants, optics parameters, and channel maps. We propose to use the CLAS CalDB database-model and their tools to handle the "calibration" database portion of the problem.

This document should grow to include a more complete list of the variables, and instructions for how to access them from the databases.

1 General features

To understand and track the quality of the data, there is a need for at least two additional collections of information. First is a "run-conditions" database, which tracks the state of the experiment during data-taking (beam energy, detector angles, target polarization, half-wave plate setting, etc). This database can be used to construct asymmetries from each run and combine them to extract the physics asymmetry. Brandon Craver is working on this database, borrowing from Jaideep Singh's experience.

The second is a "calibration" database, that contains information needed to process the raw detector responses; this database contains information such as detector geometries, calibration constants, optics, and channel maps. The present method of Podd (the analyzer) of tracking these changes through "date-stamped" directories is not satisfactory due to the frequent mid-day changes of the cabling. Even if this approach was fixed via a finer-granularity to the files or directories, it cannot separate fast from slowly changing quantities, forcing a duplication of information throughout files in the directory structures. We have found this to be error-prone, and wish to replace it with an approach that naturally provides these separations. We propose to use the CLAS CalDB database-model and their tools.

Both databases are implemented using the 'mySQL' SQL (Standard Querying Language) framework, and will be hosted on `yerphi.jlab.org`.

2 The Run Conditions database

2.1 CODA-file based information

Brandon Craver is working on creating the run-conditions database. The primary key in this database is a timestamp for when the information was collected. The concept here is that for each run, the conditions at the start and end of each coda file (eg: EPICS, trigger rates) would be stored. Each special event type (eg: HV, EPICS, threshold) that we are keeping would be in a separate table. The tables would be automatically generated according to the list of stored variables; this keeps the approach general and limits the start-up overhead. If a new column for a given table is needed, the table will be expanded to include that column with default values filled in for the missing entries (a feature of mySQL). Nerses Gevorgyan is also working partially on this project. The main-purpose of this database is to permit a fast way to check the stability of conditions while the run was being collected. Each table should contain a column for "comments".

For example, the EPICS table, while containing many more columns, will be similar to:

Table Name: EvtType131				
timestamp	IPM1H04A.XPOS	FB_A:use_RF	...	Comments
04-May-2006 01:07:03	-0.680766	RF On	...	evt 15 in e02013_4423.dat.0
04-May-2006 01:23:21	-0.668447	RF On	...	evt 123004 in e02013_4423.dat.0

A separate table, `RunTimes`, keyed on run-number will be used to store timestamps for the start and end times of each run.

2.2 Beam/Target polarization database

This database will also eventually include information from the target and beam, including polarization information. However, this will need to be brought-over from the Compton, Moller, and target logbooks and databases. Here it is important that the time of the measurement is kept, as well as their uncertainties. Each item (target polarization, beam polarization, etc.) would have its own table.

3 Calibration database

This database will hold the calibration constants, detector maps, and other information required to analyze a coda file.

We will use an implementation of the CalDB system created and used by the CLAS collaboration since 2001. It features:

- fine granularity in both time and item lookup;
- multiple-versions of constant-sets such that calibrations can evolve;
- a "run-index" containing links to the constant-sets to be used by the analysis;
- the ability for users to use and modify private copies of the run-index for testing purposes;
- an automatic change log for the run-index and constants, including author, date and time, and comment information.

Documentation about the CLAS CalDB system can be found at:
`http://clasweb.jlab.org/caldb/caldb`.

For GEn, the database is named “caldb”, to be accessed by the user “gen” on the host “yerphi.jlab.org”.

3.1 Concept

The layout of the primary tables of the CalDB database are shown in Figure 1. The top row in the figure shows how the key of the individual “items”, the actual data in the database, are assembled.

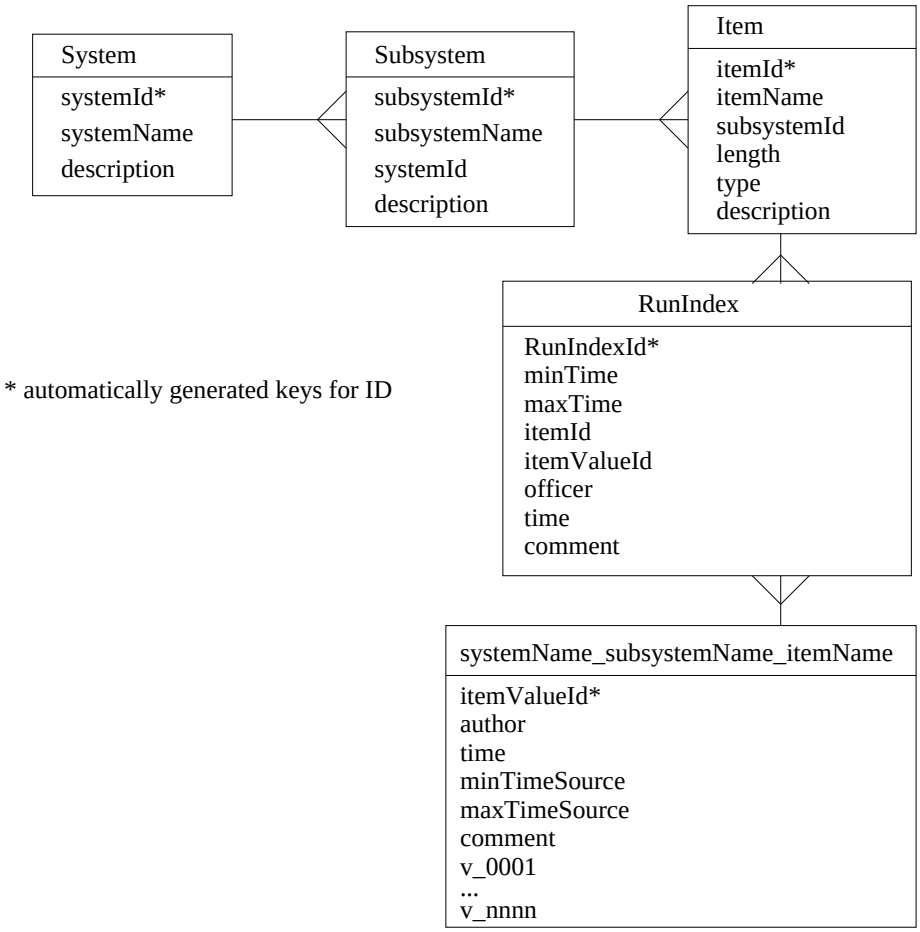

Figure 1: The general layout of the CalDB database [1]. The “crows-feet” show a “one-to-many” relation, eg: one system can contain several sub-systems, a sub-system contains many items.

The nomenclature of CalDB calls for each constant (or set of constants) to be addressed by *system*, *subsystem*, and *item*. For our purposes, a *system* is an apparatus or detector that has information to store. The *subsystem* label is optional (default value is an empty string) and can be used for a sub-detector. Finally, *item* is a specific label for the desired constant or set of constants to be stored in the database. The complete name of a calibration constant (or set) is then *system* - *subsystem* - *item*.

The bottom row shows the structure of the stored item, the data required for the analysis, which can be a quoted string, floats, or integers. Each actual constant includes information about the submission of that constant (author, date and time of submission, expected range of validity, and a mandatory comment) as well as the values themselves. This is the equivalent of a file containing a single version of a set of constants.

The middle row shows the columns of the "RunIndex" table. The RunIndex is a table of links, at least one for each item, associating an entry in the item table with a specific time-range of validity. The RunIndex table, then, is the equivalent of a list of files stating the appropriate set of constants to use. Additional copies of the RunIndex table (given different names) can be used to apply trial-sets of constants, but there will be one authoritative RunIndex table.

Different from the CalDB database, the valid ranges will be given not as run numbers, but as timestamps.

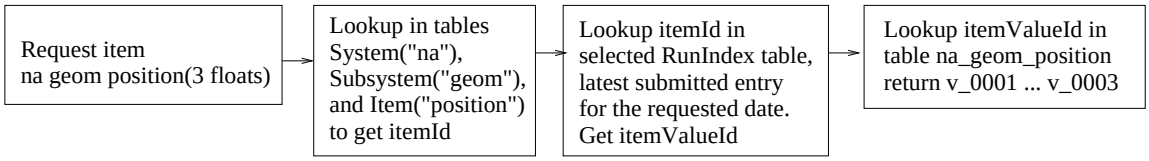

Figure 2: Example of how constants are looked up in the database.

3.2 Implementation for classes

There is some re-writing of the detectors classes required to access the database. The first task to be done is to identify the constants to be saved and generate meaningful names. This can be done most simply by looking at what is in the database files now. Common entries for a detector system would be:

detmap.detmap detector map (as a string)

geom.origin geometric placement within parent reference frame

geom.angles Euler angle description of the orientation of the detector

geom.dx, geom.dy, geom.dz half-size of the detector or element in the three body-centered directions.

3.3 Specific list of items

TO BE FILLED IN: a list of all constants required from BB and the Neutron Detector.

References

[1] "The CLAS Calibration Database", CLAS-NOTE 2001-003.
http://clasweb.jlab.org/caldb/caldb