CLAS Calibration Database Specification

L. Dennis, A. Freyberger, G. Gavalian, M. Holtrop, M. Ito, G. Riccardi, R. Suleiman, D. Weygand

November 30, 2000

1 Introduction

This document is a programming specification for the CLAS calibration database. The database contains all the necessary calibration constants needed to analyze CLAS data. As such, its primary function is to respond to queries for these constants. The values returned will in general depend on the run number of the data being analyzed.

The database is implemented in MySQL[1]. There are user-interface functions for both analysis programs and detector calibrators.

The Mapmanager[2] is the system we are currently using to hold and retrieve the calibration constants. The calibration database enhances the functionality of the Mapmanager.

2 Features

- Run indexed lookup of constants by analysis jobs.
- Ability for users to use and modify private copies of the run index without copying the constants themselves.
- Keep a change log: user, date, time, comment.
- Backward compatibility with Mapmanager.
- Optimization for speed of access by analysis jobs, not by database updaters.

3 Interface Routines for Analysis Jobs

In the initial stages, we continue to use the unmodified analysis code by dynamically generating the Map files from the database. The next step will be to write Map-like wrappers for direct DB access, so that the intermediate step of a disk file could be eliminated. Eventually, the code should evolve to invoke native DB-oriented routines to directly exploit the capabilities of the database.

To facilitate this development path we wrote both a utility to generate the database from the Map files, and one to do the reverse, generate Map files from the database.

```
runmap.pl runmin= 18495 runmax= 18502
```

4 Interface Routines for Detector Calibrators

There will be a set of application programmer interface (API) routines to access and modify the constants. We list some prototype functions that will be provided. At a minimum, we will need routines for perl and C, and probably C++ eventually.

4.1 Read Routines

Read Constants: Dumps constants for a particular item from a specified version of the database as of a specified date.

Inputs	**Output**
run number	constant values
index table name	
date	
system	
subsystem	
item	

Show Constant Sets: Shows the constant set id's for all items that are relevant for a particular run from a specified version of the database as of a specified date.

Inputs	**Outputs (per item)**
run number	system
index table name	subsystem
date	item
	item value id
	starting run
	ending run
	author
	creation date
	comment

Show Run Ranges: Shows all run ranges for a particular item from a specified version of the database as of a specified date.

Inputs	**Outputs (per run range)**
system	item value id
subsystem	starting run
item	ending run
index table name	author
date	creation date
	comment

Show Item History: Shows all constant sets that were ever valid for a specified item and run for a specified version of the database.

Inputs	Outputs (per constant set)
system	item value id
subsystem	date of validity
item	starting run
run number	ending run
	author
	creation date
	comment

4.2 Write Routines

Write Constants Set: Makes a new set of constants (item value). Does not link the constants to any run number. Author and creation date are entered automatically.

Input	Output
constant set	item value id
source starting run	
source ending run	
comment	

Link Constants To Run Range: Makes the correspondence between a particular set of constants and a particular run range for a specified item and version of the database.

Input	Output
item value id	none
starting run	
ending run	
system	
subsystem	
item	
index table name	

5 Database Tables

The database tables are shown schematically in Fig. 1 and are listed explicitly in Tables 1-2. Notes on this structure:

Intermediate level of referencing. Run numbers are not kept in the same table as the constants. Different versions of the calibration constants are simply different instances of the run index table.

"Freeze" by saving a version of the run index table. There is no need to duplicate the constants themselves. Having a run index table is a form of documentation; one knows exactly which constants are referenced in the frozen version.

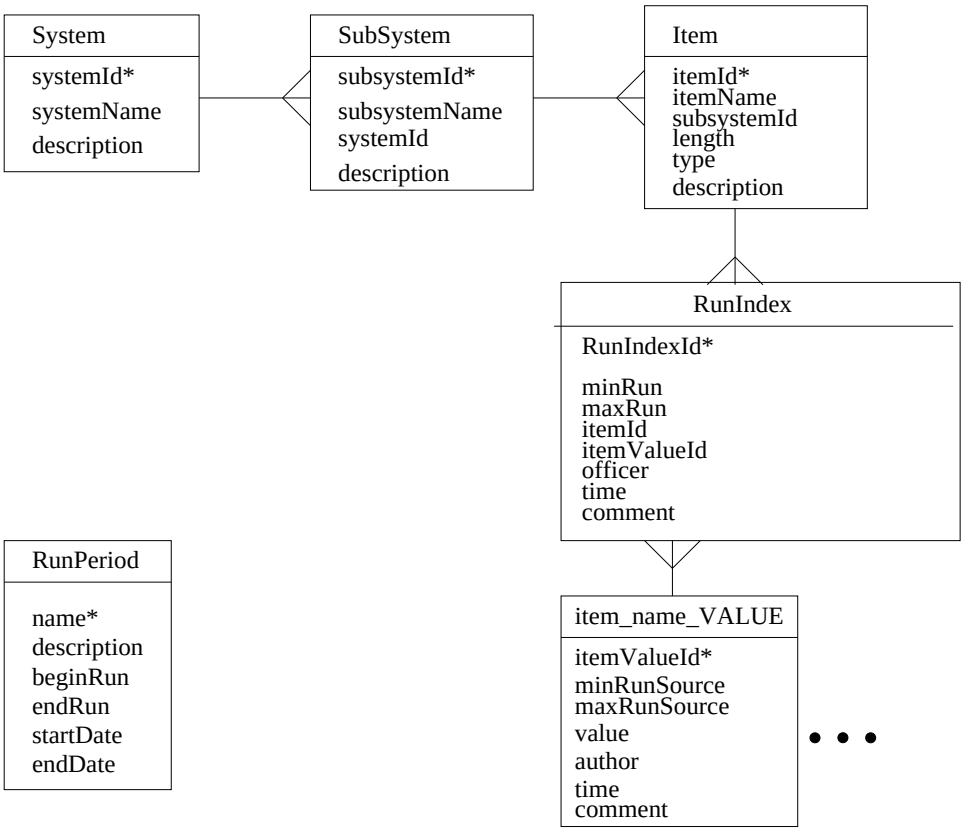

Figure 1: Schematic drawing of the database tables.

RunPeriod			
column name	type	example	comment
name	varchar	"e1a"	Primary key
description	text		
beginRun	int	7374	
endRun	int	8110	
startDate	date	1997-12-09	
endDate	date	1997-12-24	

System			
column name	type	example	comment
systemId	int	1	Primary key auto increment
systemName	varchar	"CALL_CALIB"	
description	text		

SubSystem			
column name	type	example	comment
subsystemId	int	1	Primary key auto increment
subsystemName	varchar	"RFoffset"	
systemId	int	1	Foreign key reference System
description	text		

Item			
column name	type	example	comment
itemId	int	1	Primary key auto increment
itemName	varchar	"rf2rf1Offset"	
subsystemId	int	1	Foreign key reference SubSystem
length	int	1	Number of elements
type	varchar	"float"	
description	text		

Table 1: Informational tables.

RunIndex			
column name	type	example	comment
RunIndexId	int	1	Primary key auto increment
minRun	int	10865	
maxRun	int	100000	
itemId	int	1	Foreign key reference Item
itemValueId	int	1	Foreign key reference item_name_VALUE
officer	varchar	"dbmanager"	
time	timestamp	20000502165717	
comment	text		

CALL_CALIB_RFoffset_rf2rf1Offset_VALUE			
column name	type	example	comment
itemValueId	int	1	Primary key auto increment
minRunSource	int	10865	
maxRunSource	int	10870	
value_1	"float"	1.82	
author	varchar	"smith"	
time	timestamp	20000502165717	
comment	text		

Table 2: Functional tables.

Constants are "never" deleted from the database. By keeping old constants, any frozen version will remain viable. The default would be to keep all constants for all time. The only deletions would be for obvious mistakes.

Private/custom versions of constants are easy to make. Individual users can copy and modify a run index table for private use. This is especially useful when doing code development where one wants a stable set of constants. Also developers of new calibration schemes can modify their private tables to use prototype sets of constants without affecting the rest of the collaboration.

6 Private and Frozen Versions of Constants

There are many cases where independent versions of the calibration constants are desirable.

Freeze constants for large production runs. This allows us to reproduce results at a later time.

Private code development. Changes to test results can be limited to changes induced by the code. The calibration constants will be stable.

Private calibration constants development. Changes to test results can be limited to only those induced by the constants under study. All of the other calibration constants can be kept stable. In addition, changes made to the constants during testing will not affect other users.

There are also several features that would be convenient to have when creating an independent version of the constants. Among these are an option to select certain run ranges for the version, so that information for irrelevant runs can be dropped, and the ability to modify and correct each independent version, independently of the others.

One solution that we considered and rejected as too unwieldy was to add "tag" columns to the run index table. Instead, we propose creation of independent versions by making new tables which are copies of selected rows of the run index table. These copies can be used in place of the run index to look up appropriate calibration constants. The identity of each copy is managed via the name of the table. This method automatically gives us the desired features mentioned above. The modification history of an independent copy can be tracked in exactly the same way as for the main run index table without any additional programming.

7 Access Privileges

There are several different levels of access to the database:

1. database manager (1 or 2 people)
2. experts (1 or 2 per calibration system)

user table			
Host	User	Password	Privileges
jlabs1	dbmanager	non-NULL	none
jlabs1	expert_1	non-NULL	none
jlabs1	expert_2	non-NULL	none
...	...	...	...
jlab%	NULL	NULL	none
claspc%	NULL	NULL	none
...	...	...	...
einstein.sr.unh.edu	NULL	NULL	none

Table 3: MySQL user table. Any user from any CLAS designated host can connect to the database. Passwords must be set for users that need to be identified.

3. users

Further, there are several features which are built into the access system:

- Individuals can, without approval
  - create their own item_name_VALUE tables
  - insert constants to the db
  - create private run index tables
  - modify (*i. e.*, insert, update, delete) private run index tables
  - drop or modify run index tables of others
  - read anything in the database
- Individuals cannot
  - delete constants from the public item_name_VALUE tables
  - drop or modify the public run index table
- Designated experts can, in addition
  - delete constants from the public item_name_VALUE tables
  - modify the public run index table
- Designated experts cannot
  - drop item_name_VALUE tables from the db
  - drop the public run index
- Database manager can do anything to the database

The MySQL access tables shown in Tables 3-5 accomplish these goals.

db table			
Host	Db	User	Privileges
%	calib	dbmanager	all
%	calib	%	select

Table 4: MySQL db table. The dbmanager has all privileges in the calib database. Any user that can connect can read any of the tables in the calib database.

tables_priv table				
Host	Db	User	Table_name	Table_priv
%	calib	expert_1	RunIndex	select,insert,update,delete
%	calib	expert_1	item_name_VALUE	select,insert,update,delete
%	calib	%	RunIndexUser%	all
%	calib	%	item_name_VALUE	select,insert
%	calib	%	item_name_VALUEUser%	all

Table 5: MySQL tables_priv table. An expert can modify the RunIndex table. An expert can modify an item_name_VALUE table. Any user can create a private RunIndex Any user can read or add to an item_name_VALUE table. Any user can create and modify private item_name_VALUE tables.

8 Database Deployment

There will be one authoritative version of the database. All write operations must be performed on this JLab-resident copy. If remote copies are necessary, they should be downloaded from JLab.

References

[1] http://www.mysql.org

[2] http://www.jlab.org/~manak/packages/Map/mapmanager.html

`$Id: cal_db_spec.tex,v 2.15 2000/12/04 20:50:18 marki Exp $`