The CLAS Calibration Database [a]

Harut Avagyan, Mark Ito, Greg Riccardi, Riad Suleiman

May 28, 2003

Abstract

This note describes the design and use of the CLAS calibration database (CalDB). 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. This note supersedes CLAS-NOTE 2000-008[1].

[a] This note is available in a web version at `http://clasweb.jlab.org/caldb/caldb/` .

Contents

1 Introduction

The database is implemented in MySQL[2]. There are user-interface functions for both analysis programs and detector calibrators.

The Mapmanager[3] is the system we used before development of the database to hold and retrieve the calibration constants. In fact the initial version of the database is a copy of the constants and run correlations held in the Map at the time of conversion on January 29, 2001.

The design has the following 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 the Mapmanager.

Some broad technical details:

- The default database server is `clasdb.jlab.org`.
- Database servers should run MySQL version 3.23.6 or greater. Older versions do not have true floating point storage of FLOAT values.
- The default location for all tables is the `calib` database.

2 Database Tables

The database tables are shown schematically in Fig. 1 and are listed explicitly in Tables 1 and 2. This structure for the database supports all of the features mentioned in Section 1. There are several remarks to make on the database structure:

Unique item key. Each item in the database has a unique key (`itemId`) even though it may not have a unique item name. Given a particular triplet of system name, subsystem name and item name, the key can be retrieved from a join of the `System`, `Subsystem` and `Item` tables.

Intermediate level of referencing. Run numbers are not kept in the same table as the constants. To get constants for a particular item and run one queries the run index table (`RunIndex`) for the corresponding item value key (`itemValueId`). One then gets the constants from the appropriate constant set table (*e. g.*, `DC_DOCA_t_max_Sector3`), looking them up with the item value key.

This correspondence between a range of run numbers and a constant set is called a link. Each row of a run index table is thus a link. Each link is specific to a particular item.

New constants are entered as a new entry to an item value table. A new set never overwrites an old set, even if it is meant to supersede the old one. Rather, the change is made in the run index table; the same old runs are made to point to the new constant set.

"Freeze the constants" by saving a version of the run index table. To get the frozen version of constants, one uses the frozen version of the run index table. Since old constant sets are not deleted, the frozen version remains viable even after subsequent "changes" to the constants. Recall that these "changes" are really only additions of new constant sets and editing of the non-frozen run index table. There is no need to duplicate the constants themselves when freezing.

History is kept by time stamping run index changes. Like the constants themselves, entries in the run index table are never discarded. As such, a particular item/run combination may have several constant sets associated with it. To get the "latest" version, one chooses the entry with the most recent time stamp. To get the version as of a particular date, one chooses the entry that was the most recent as of that date.

Overlapping run ranges: last one wins. Since entries in the run index table are never discarded and new entries are not in anyway bound to match the minimum and maximum run specifications of previous entries, it is possible, in fact likely, that for a given item there will be run index entries that have partially overlapping run range specifications. For example assume that there is an item with the following (partially listed) entries in the run index table:

runMin	runMax	itemValueId	time
1000	6000	234	2001-01-29 14:15:16
2000	4000	235	2001-02-02 02:03:04
3000	5000	236	2001-03-15 08:09:10

The second line says "At around two in the morning on February 2, 2001, we were instructed that we should use the constant set with ID number 235 for all runs between 2000 and 4000 inclusive for this item." Note that this instruction conflicts with that on the first line. And that the third line conflicts partially with that on the second line. The conflicts are always resolved by using the last instruction given, where "last" in this context means the entry with the greatest value of `time`. So if we are analyzing run 3100, the instruction of March 15 is followed and constant set 236 is used. For run 1800, only the January 29 instruction applies so constant set 234 will be used. Note that this behavior is not intrinsic to MySQL; it has been coded into all of the access routines.

Effective run ranges vs. specified run ranges. In the example of the previous section, three sets of **specified run ranges** from the run index table are displayed: 1000-6000, 2000-4000 and 3000-5000. Given the conflict-resolution scheme described there, we get the following **effective run ranges**:

System			
column name	type	example	comment
systemId	int	4	primary key, auto-increment
systemName	varchar	"DC_DOCA"	
description	text	"from the Map"	

Subsystem			
column name	type	example	comment
subsystemId	int	17	primary key, auto-increment
subsystemName	varchar	"t_max"	
systemId	int	4	reference to System
description	text	"from the DC_DOCA Map"	

Item			
column name	type	example	comment
itemId	int	44	primary key, auto-increment
itemName	varchar	"Sector2"	
subsystemId	int	17	reference to Subsystem
length	int	36	number of elements
type	varchar	"float"	
description	text	"from the DC_DOCA Map," subsystem t_max"	

Table 1: Informational tables.

Effective run min.	**Effective run max.**	`itemValueId`
1000	1999	234
2000	2999	235
3000	5000	236
5001	6000	234

Note that the beginning and end runs of effective run ranges may not appear in any of the specified run ranges, *i. e.*, they may not appear in the run index table. They are a consequence of the conflict-resolution algorithm and describe the effective behavior of the CalDB.

Private versions of constants are easy to make. Individual users can copy and modify a run index table for private use. This is especially useful for a developer of a new calibration scheme. He can modify his private tables to use prototype sets of constants without affecting the rest of the collaboration. These private constants will live in the public item value tables, but no one besides the developer will be pointing to them. To promote a private version to the public version, he need only insert a new entry in the public run index table.

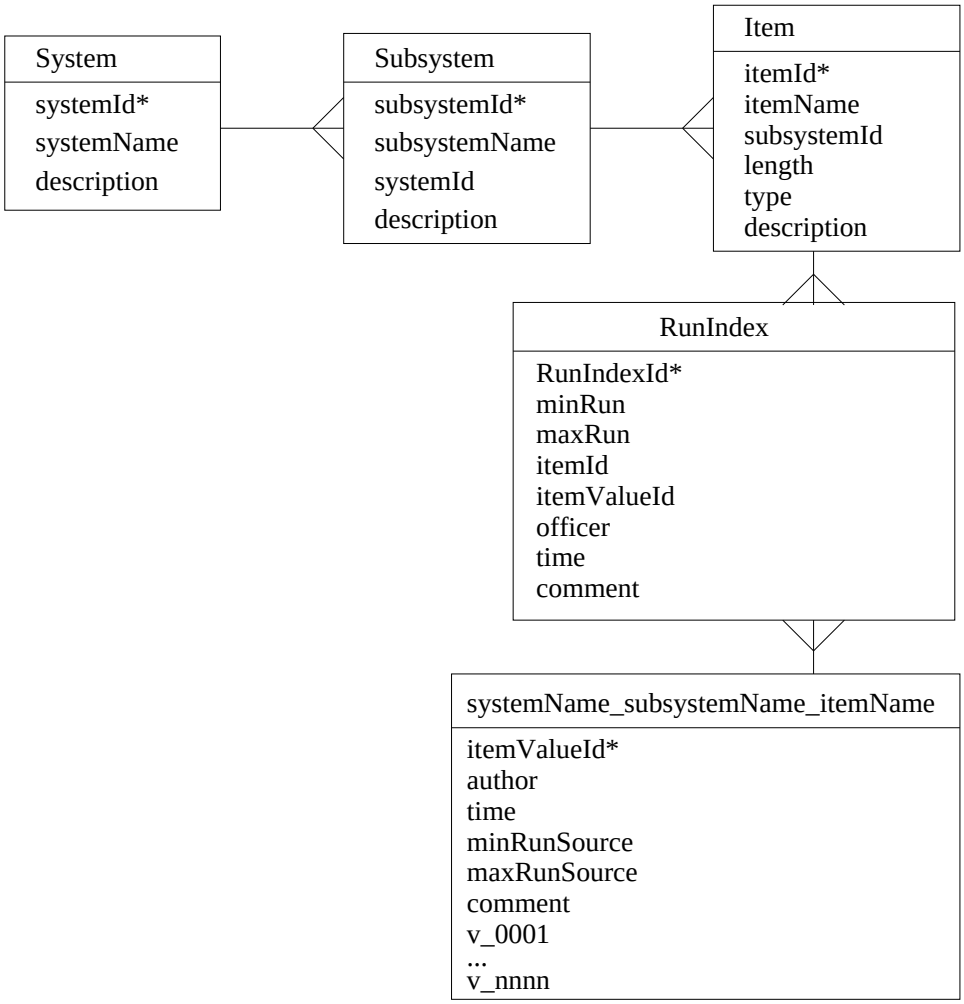

Figure 1: Schematic drawing of the database tables. Crow's feet indicate the "one" side and "many" side of one-to-many relationships. Primary keys are indicated with asterisks. There are actually many instances of the `systemName_subsystemName_itemName` table; there is one for each item in the database. The constant values themselves are the columns named `v_0001`, `v_0002`,....

RunIndex			
column name	type	example	comment
RunIndexId	int	682	primary key, auto-increment
minRun	int	20020	
maxRun	int	20062	
itemId	int	44	reference to Item
itemValueId	int	50	reference to item value table
officer	varchar	"dbmanager"	
time	timestamp	20010117120303	
comment	text	"copied from Map"	

DC_DOCA_t_max_Sector2			
column name	type	example	comment
itemValueId	int	50	primary key, auto-increment
author	varchar	"davidl"	
time	timestamp	20000502165717	
minRunSource	int	20030	
maxRunSource	int	20040	
comment	text	"dc3 with new t_max def'n"	
v_0001	float	192.58	
...	...	...	
v_0036	float	1347.67	

Table 2: Functional tables. The DC_DOCA_t_max_Sector2 table is an example of an item value table. Item value tables are the ones that actually hold the constants. There is an item value table in the database for each item.

3 Interface Routines for Analysis Jobs and Detector Calibrators

3.1 The Application Programmer Interface

There is a set of application programmer interface (API) routines to access and modify the constants. At this writing, only a Perl API is available. We will need routines in C, and probably C++ eventually.[b] We list the functions that are available. Arguments are specified in a language-independent form here.

3.1.1 Initialization Routine

Connect to Server: Establishes a connection to the MySQL database server.

Inputs	**Output**
hostname	connection handle/structure
user	
password	

3.1.2 Read Routines

Show Systems: Lists all systems. If subsystem and/or item are provided, only list systems correlated with the specified subsystem/item.

Inputs	**Output (per system)**
subsystem	systemId
item	systemName
	description

Show Subsystems: Lists all subsystems. If system and/or item are provided, only list subsystems correlated with the specified system/item.

Inputs	**Output (per subsystem)**
system	subsystemId
item	subsystemName
	systemId
	description

Show Items: Lists all items. If system and/or subsystem are provided, only list items correlated with the specified system/subsystem.

[b] There exists a working C library to do Map emulation. See Section 4. In this section we are only concerned with the functions designed with the CalDB structure in mind; the native-CalDB routines have different arguments than for the native-Map routines.

Inputs	Output (per item)
system	itemId
subsystem	itemName
	subsystemId
	length
	type
	description

Show Constants By Run: 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 Constants By ID: Dumps constants for a particular item from a specified version of the database as of a specified date. Look up is done by item value ID rather than run number, date and run index table name.

Inputs	Output
itemValueId	constant values
system	
subsystem	
item	

Show Constant Sets By Run: Shows the item value 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
	itemValueId
	starting run
	ending run
	officer
	link date
	comment

Show Constant Sets by Item: Shows all item value ID's for a particular item including those not linked to runs.

Inputs	Outputs (per item)
systemName	itemValueId
subsystemName	author
itemName	date
	minRunSource
	maxRunSource
	comment

Show Run Ranges By Item: Shows run ranges for a particular item from a specified version of the database as of a specified date and in specified run range.

Inputs	Outputs (per run range)
system	itemValueId
subsystem	starting run
item	ending run
index table name	officer
date	link date
minRun	comment
maxRun	

Show Item Value History: Shows all item value ID's that were ever valid for a specified item and run for a specified version of the database.

Inputs	Outputs (per constant set)
system	itemValueId
subsystem	link date
item	starting run
run number	ending run
index table name	officer
	comment

3.1.3 Write Routines

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

Input	Output
system	itemValueId
subsystem	
item	
constant set	
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
itemValueId	RunIndexId
starting run	
ending run	
system	
subsystem	
item	
index table name	
comment	

3.2 Controlling CalDB from the Command Line

Perl scripts to give command line control of the CalDB using the Perl API are available.

Arguments are specified on the command line in the form `argname=argvalue`. No white-space is allowed before or after the `=` sign. For example, for a mythical script called `caldb_script.pl` that takes three arguments `x`, `xlabel` and `comment`:

```
caldb_script.pl x=7.9 xlabel=distance comment='just an example'
```

sets the value of `x` to 7.9, the value of `xlabel` to "distance" and the value of `comment` to the string "just an example". Here the single quotes are used to pass the blanks in `comment` to the script from the Unix shell.

There are some command arguments that appear in more than one script. They are described here in detail. When they appear in the subsections on the individual commands, their descriptions will be brief and refer back to this section.

`help` Most of the commands will type out a usage message if the `help` argument is set to a non-zero value, *e. g.*,

```
caldb_show_systems.pl help=1
```

`inputFile` For the commands that write constants into the CalDB, this argument specifies the name of the input file. The internal format of the file for float's and int's should have one constant per line, with each line ending in a newline character. For char's, the entire contents of the file will be written as a character string, including newline characters. Because of this, the file can contain multiple lines.

`runIndexTable` If this argument is present, then all database queries will use the specified run index table rather than the default, `calib.RunIndex`. This argument allows access to frozen or private versions of constants via use of a private run index. Run indices can be specified as `runIndexTable=tablename` if the table is in the `calib` database. If not, then the specification takes the form `runIndexTable=databasename.tablename`. For example:

Specification	Result
`runIndexTable=RunIndex`	Use the `RunIndex` table of the `calib` database. This is a restatement of the default.
`runIndexTable=RunIndexE1`	Use the `RunIndexE1` table of the `calib` database.
`runIndexTable=calib_user.RunIndexKjoo`	Use the `RunIndexKjoo` table of the `calib_user` database.

`time` This argument specifies the historical time to use when making all queries. Changes to the CalDB that came later than this time will be ignored. Supplying this argument invokes the history mechanism of the CalDB. Since entries that post-date the specified time are ignored, the results are equivalent to those that would have been obtained by asking for the latest version *at the specified time*. More simply, you get the answer to the question "What did the constants look like back then?". Time can be specified using a variety of formats. See the MySQL manual[2] for details, but all of the following are valid time specifications:

```
time='2001-06-12 15:25:59'
time=01/06/12
time=20010612152559
time=010612
```

`hostname` This names the database server to use. If supplied, the script will connect to the MySQL server running on the specified node for all database queries. If not supplied, the server defaults to `clasdb.jlab.org`. This allows non-Lab users and/or CalDB developers to use local database mirrors.

`user` This specifies the username to use when connecting to the MySQL server. It does not affect the value of the `author` value in the constant set tables or of the `officer` value in the run index tables. Those are derived from the user's environment. It is only relevant for write operations. For read operations the MySQL username `clasuser` is used. By default, if the write is to the `calib_user` database (see Section 6), `calib_user` is used. Otherwise the value of the `USER` shell environment variable is used.

3.2.1 caldb_show_systems.pl

Shows systems in the CalDB. With no arguments, all systems are displayed. If subsystem and/or item are specified, then only those systems which have matching subsystems and items are displayed.

The usage message is:

```
usage:
caldb_show_systems.pl [subsystem=<subsystem>] [item=<item>] \
    [hostname=<hostname of db server>] [oneline=<non-zero>] \
    [help=<non-zero>]
```

```
example:
caldb_show_systems.pl
```

The arguments are:

`subsystem` (optional) If this is supplied, systems are restricted to those that contain the specified subsystem. The restriction is and'ed with the `item` restriction if that is present.

`item` (optional) If this is supplied, systems are restricted to those that contain the specified item. The restriction is and'ed with the `subsystem` restriction if that is present.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`oneline` (optional) If set to a non-zero value, the systems will be displayed on one line, separated by spaces.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

3.2.2 caldb_show_subsystems.pl

Shows subsystems in the CalDB. With no arguments, all subsystems are displayed. If system and/or item are specified, then only those subsystems which have matching systems and items are displayed.

The usage message is:

```
usage:
caldb_show_subsystem.pl [system|s=<subsystem>] [item|i=<item>] \
    [hostname=<hostname of db server>] [oneline=<non-zero>] \
    [help=<non-zero>]

example:
caldb_show_subsystems.pl
```

The arguments are:

`system` or `s` (optional) If this is supplied, subsystems are restricted to those that are contained in the specified system. The restriction is and'ed with the `item` restriction if that is present.

`item` or `i` (optional) If this is supplied, subsystems are restricted to those that contain the specified item. The restriction is and'ed with the `system` restriction if that is present.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`oneline` (optional) If set to a non-zero value, the subsystems will be displayed on one line, separated by spaces.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

3.2.3 caldb_show_items.pl

Shows items in the CalDB. With no arguments, all items are displayed. If system and/or subsystem are specified, then only those items which have matching systems and subsystems are displayed.

The usage message is:

```
usage:
caldb_show_items.pl [system=<system>] [subsystem=<subsystem>] \
    [hostname=<hostname of db server>] [oneline=<non-zero>] \
    [help=<non-zero>]

example:
caldb_show_items.pl
```

The arguments are:

`subsystem` (optional) If this is supplied, items are restricted to those that are contained in the specified system. The restriction is and'ed with the `subsystem` restriction if that is present.

`item` (optional) If this is supplied, items are restricted to those that are contained in the specified subsystem. The restriction is and'ed with the `system` restriction if that is present.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`oneline` (optional) If set to a non-zero value, the items will be displayed on one line, separated by spaces.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

3.2.4 caldb_show_item_info.pl

Shows various properties of a particular item. The item must be specified by system, subsystem and item. The script will print out the `systemId`, `subsystemId`, `itemId`, `length`, `type` and `description` for the specified item. See Table 1 for descriptions of these values.

The usage message is:

```
usage:
caldb_show_item_info.pl system|s=<system> subsystem|ss=<subsystem> \
    item|i=<item> [hostname=<hostname of db server>] [help=<non-zero>]

example:
caldb_show_item_info.pl system=DC_DOCA subsystem=xvst_params item=SL3
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example given in the usage message above produces the following output:

```
systemId = 4
subsystemId = 23
itemId = 87
length = 24
type = float
description = from the DC_DOCA Map, subsystem xvst_params
```

3.2.5 caldb_show_ssi.pl

Show existing system/subsystem/item (SSI) combinations along with the item ID number. If the `system`, `subsystem` and/or `item` arguments are specified, then only those SSI's that are consistent with the arguments are shown.

The usage message is:

```
purpose: show system/subsystem/item combinations and corresponding item ID's
usage:
caldb_show_ssi.pl [system|s=<system>] [subsystem|ss=<subsystem>] \
    [item|i=<item>] [hostname=<hostname of db server>] [help=<non-zero>]

example:
caldb_show_ssi.pl s=DC_DOCA ss=t_max
```

The arguments are:

`system` or `s` (optional) System specification. If omitted, all systems are considered.

`subsystem` or `ss` (optional) Subystem specification. If omitted, all subsystems are considered.

`item` or `s` (optional) Item specification. If omitted, all items are considered.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
DC_DOCA t_max comment 49
DC_DOCA t_max Sector1 43
DC_DOCA t_max Sector2 44
DC_DOCA t_max Sector3 45
DC_DOCA t_max Sector4 46
DC_DOCA t_max Sector5 47
DC_DOCA t_max Sector6 48
```

3.2.6 caldb_show_sets_run.pl

Given a run number, information for all relevant constant sets are shown, spanning all valid system/subsystem/item (SSI) combinations. In other words, for the given run and any one SSI, a unique constant set is identified via the run index table. That constant set is specified by its `itemValueId`. This script displays this constant set identifier, along with other related information, and does so for every SSI in the CalDB.

For each SSI combination the following are displayed: `systemName`, `subsystemName`, `itemName`, `runMin`, `runMax`, `itemValueId`, `officer`, `time` and `comment`. See Tables 1 and 2 for descriptions of these values. Note that `time` and `comment` here are from the `RunIndex` table. Note further that `runMin` and `runMax` are also from the `RunIndex` table and as such define the relevant specified run range and not the effective run range. See Section 2 for a discussion of the difference.

The usage message is:

```
usage:
caldb_show_sets_run.pl r=<run> [it=<run index table>] \
    [t=<time of validity>] [hostname=<hostname of db server>] \
    [help=<non-zero>]

alternate flag names:
  r or run for run number
  it or runIndexTable for run index table name
  t or time for time of validity

example:
caldb_show_sets_run.pl r=20000
```

The arguments are:

`run` (required) The run specification.

`runIndexTable` or `it` (optional) The name of the run index table to use. The default value is `calib.RunIndex`. See Section 3.2 above.

`time` (optional) The historical time to use in database queries. The default value is (effectively) the current time. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
using runIndexTable = RunIndex
CALL_CALIB pedestals unc 1 1000000 1 dbmanager 2001-01-29 15:59:03
    copied from Map
CALL_CALIB pedestals value 1 1000000 1 dbmanager 2001-01-29 15:59:03
    copied from Map
CALL_CALIB RFoffset rf2rf1Offset 10865 1000000 1 dbmanager 2001-01-29 15:59:02
    copied from Map
.
.
.
TAG_SCALER PSEff value 19983 20032 10 dbmanager 2001-01-29 17:25:20
    copied from Map
TAG_SCALER RTSL value 19983 20032 10 dbmanager 2001-01-29 17:25:21
    copied from Map
TAG_SCALER TagEff value 19983 20032 10 dbmanager 2001-01-29 17:25:21
    copied from Map
```

(Many lines have been omitted for brevity. Comments have been printed on separate lines to fit on the page.) The last line tells us that for run 20000, the `TAG_SCALER`, `TagEff`, `value` item uses constant set 10 (the `itemValueId`), and that that assignment was copied from the Map by `dbmanager` on January 29, 2001.

3.2.7 caldb_show_sets_item.pl

Shows all constants sets by ID (the `itemValueId`) for a particular system/subsystem/item, including those not linked to runs. For each constants set, the values of `itemValueId`, `author`, `time`, `minRunSource`, `maxRunSource` and `comment` from the appropriate constant set table (*i. e.*, item value table) will be shown.

The usage message is:

```
Makes a list of all constant set ID's for the specified
system/subsystem/item.

Usage:
caldb_show_sets_item.pl system=<system> subsystem=<subsystem> item=<item> \
    [hostname=<hostname of db server>] [help=<non-zero>]

example:
caldb_show_sets_item.pl system=DC_DOCA subsystem=xvst_params item=SL3
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
1 dbmanager 2001-01-29 16:05:23 25190 1000000 'copied from Map'
2 dbmanager 2001-01-29 16:05:23 24088 25189 'copied from Map'
3 dbmanager 2001-01-29 16:05:23 24037 24087 'copied from Map'
.
.
.
298 lachniet 2001-05-22 11:41:53 NULL NULL 'Map-style put'
299 lachniet 2001-05-23 18:29:52 NULL NULL 'Map-style put'
300 lachniet 2001-05-25 15:39:31 NULL NULL 'Map-style put'
```

This is simply a complete list of all constant sets that have ever been entered into the CalDB for the `DC_DOCA`, `xvst_params`, `SL3` item. No information about the run numbers to which the sets should be applied is displayed. (Recall that `minRunSource`, `maxRunSource` are comment-like fields.)

3.2.8 caldb_show_run_ranges.pl

For a given system/subsystem/item (SSI) combination, this script shows all effective run ranges. See Section 2 for a discussion of effective vs. specified run ranges. For each range, the values of effective begin run and effective end run are displayed along with the corresponding values of `itemValueId`, `officer`, `time`, the constant values (optionally) and `comment` from the run index table. See Table 2 for descriptions of the run-index-resident values.

The usage message is

```
purpose: shows effective run ranges for a specific item

usage:
caldb_show_run_ranges.pl system|s=<system> subsystem|ss=<subsystem> \
    item|i=<item> [runMin|min=<run min>] [runMax|max=<run max>] \
    [const=<non-zero>] [first=<non-zero>] \
    [runIndexTable|it=<run index table>] [time|t=<time of validity>] \
    [hostname=<hostname of db server>] [help=<non-zero>]

example:
caldb_show_run_ranges.pl s=mom_corr ss=theta_func i=sector6
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`const` (optional) If set to a non-zero value, the constants themselves will be displayed, along with the other information about the run ranges.

`first` (optional) If set to a non-zero value, and if `const` is non-zero, then only the first constant of the array of constants will be displayed. If `const` is not set or set to zero, this argument has no effect.

`runIndexTable` or `it` (optional) The name of the run index table to use. The default value is `calib.RunIndex`. See Section 3.2 above.

`time` (optional) The historical time to use in database queries. The default value is (effectively) the current time. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
using runIndexTable = RunIndex
1 11799 3 dbmanager 2001-01-29 17:26:25 copied from Map
11800 11899 2 dbmanager 2001-01-29 17:26:25 copied from Map
11900 1000000 1 dbmanager 2001-01-29 17:26:25 copied from Map
```

This tells us that for the `mom_corr/theta_func/sector6` item, there are three effective run ranges defined: 1-11799, 11800-11899 and 11900-1000000. These ranges use constant sets 3, 2 and 1 respectively (these are the `itemValueId`'s). These assignments were all copied from the Map on January 29, 2001.

3.2.9 caldb_show_history.pl

For the given system/subsystem/item (SSI) combination and run, show each (historical) time that the calibration constants have changed. For each change, the `time`, `minRun`, `maxRun`, `itemValueId`, `officer` and `comment` values from the run index table are shown. See Table 2 for descriptions of these values. Note that the `itemValueId` uniquely identifies the constant set that was applied at the `time`. The information is displayed in reverse chronological order.

```
usage:
caldb_show_history.pl s=<system> ss=<subsystem> i=<item> r=<run number> \
    [it=<run index table>] [hostname=<hostname of db server>] \
    [help=<non-zero>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
```

```
  i or item for item name
  r or run for run number
  it or runIndexTable for run index table name

example:
caldb_show_history.pl s=DC_DOCA ss=t_max i=Sector3 r=23000
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`run` (required) The run specification.

`runIndexTable` or `it` (optional) The name of the run index table to use. The default value is `calib.RunIndex`. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
using runIndexTable = RunIndex
2001-03-26 15:07:13 22950 23049 271 claschef copied
2001-01-29 15:59:39 22933 23034 8 dbmanager copied from Map
```

This tells us that for `DC_DOCA/t_max/Sector3`, `claschef` changed the constants for run 23000 to use constant set 271 (the `itemValueId`). This was done on March 3, 2001 at around 3 pm. At that time, the intention was to apply constant set 271 for all runs from 22950 to 23049 inclusive. Before that we were using constant set 8 for run 23000. That assignment was copied from the Map by the `dbmanager`.

3.2.10 caldb_show_constants_run.pl

Shows the values of the calibration constants themselves for the specified system/subsystem/-item and run number. By default, the constant set ID (`itemValueId`), `author`, `time`, `runMinSource`, `runMaxSource` and `comment` from the constant set table are printed before the constant values are printed out. See Table 2 for a description of these values.

The usage message is:

```
usage:
caldb_show_constants_run.pl s=<system> ss=<subsystem> i=<item> r=<run> \
    [it=<run index table>] [t=<time of validity>] \
    [hostname=<hostname of db server>] [q=<non-zero to suppress some info>] \
    [help=<non-zero>]
```

```
alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  r or run for run number
  it or runIndexTable for run index table name
  t or time for time of validity
  q or quiet for suppression of extra information

example:
caldb_show_constants_run.pl s=DC_DOCA ss=t_max i=Sector6 r=20000
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`run` (required) The run specification.

`runIndexTable` or `it` (optional) The name of the run index table to use. The default value is `calib.RunIndex`. See Section 3.2 above.

`time` (optional) The historical time to use in database queries. The default value is (effectively) the current time. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`quiet` (optional) A non-zero value will suppress printing of column headings and other extra information.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
53 dbmanager 20010129155955 19861 20019
copied from Map
210.748
213.066
215.41
.
.
.
1524.74
1539.98
1555.38
```

This tells us that for run 20000, constant set 53 is used, it was copied from the Map on January 29, 2001 by `dbmanager`, and that constant set 53 is composed of the values shown.

3.2.11 caldb_show_constants_id.pl

Shows the values of the calibration constants themselves for the specified system/subsystem/-item and constant set ID. By default, the constant set ID (`itemValueId`), `author`, `time`, `runMinSource`, `runMaxSource` and `comment` from the constant set table are printed before the constant values are printed out. See Table 2 for a description of these values.

The usage message is:

```
usage:
caldb_show_constants_id.pl s=<system> ss=<subsystem> i=<item> \
    id=<constant set ID> [hostname=<hostname of db server>] \
    [q=<non-zero to suppress some info>] [help=<non-zero>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  id or constantSetId for constant set ID
  q or quiet for quiet mode

example:
caldb_show_constants_id.pl s=DC_DOCA ss=t_max i=Sector6 id=253
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`constantSetId` (required) Identifies the constant set whose contents are to be displayed. This is the `itemValueId` field of the constant set table.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`quiet` (optional) A non-zero value will suppress printing of column headings and other extra information.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
253 lachniet 20010216134855
Map-style put
148.789
150.425
152.08
.
```

```
.
.
1225.21
1237.46
1249.84
```

This tells us that for `DC_DOCA/t_max/Sector6`, constant set 253 was written to the CalDB using the Map-emulation libraries ("Map-style put") on February 16, 2001 by `lachniet` and is composed of the values shown.

3.2.12 caldb_show_RUN_CONTROL.pl

Shows the values of beam energy and torus and minitorus currents per run in a given run range.

The usage message is:

```
usage:
caldb_show_RUN_CONTROL.pl  min=<run min> max=<run max>  \
    it=<run index table> \
    [t=<time of validity>] \
    [hostname=<hostname of db server>]  \
    [help=<non-zero>] [q=<non-zero to suppress some info>]


alternate flag names:
  min or runMin
  max or runMax
  it or  runIndexTable for source run index table name
  t or time for time of validity
  q or quiet for suppression of extra information


example:
caldb_.show_RUN_CONTROL.pl  min=31500 max=32170 it=calib_user.RunIndexe6
```

The arguments are:

`min` (required) Minimum run number

`max` (required) Maximum run number

`it` (required) source RunIndex file

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

The example command in the usage message above gives:

```
> caldb_show_RUN_CONTROL.pl min=31520 max=31525 it=calib_user.RunIndexe6
using RunIndex = calib_user.RunIndexe6 from clasdb.jlab.org

run#    Energy(MeV)  Torus    mini-Torus
-------------------------------------
31520   5769.59       2250      6000
31521   5769.59       2250      6000
31522   5769.59       2250      6000
31523   5769.59       2250      6000
31524   5769.59       2250      6000
31525   5769.59       2250      6000
```

This shows the energy,torus and minitorus currents per run.

3.2.13 caldb_show_changes.pl

Shows changes made to a run index from a specified time in the past to the present. Optionally, the system, subsystem, item, and/or officer can be specified to modify the search for changes. If any of these is omitted, the search is conducted over all of the corresponding possibilities. Also a specific run index can be specified as the target of the search. By default, only the first three lines of any comments are displayed, for brevity, but the full comment can be displayed if desired.

The usage message is:

```
purpose: Shows changes to a run index back to a specified time for
         specified items entered by a specified officer.

usage:
caldb_show_changes.pl time|t=<time to go back to> [s|system=<system name>] \
    [subsystem|ss=<subsystem name>] [item|i=<item name> \
    [officer|o=<officer name>] [runIndexTable|it=<run index table name>]  \
    [fullComment=<non-zero>] [quiet=<non-zero>] \
    [hostname=<hostname of db server>] [help=<non-zero>]

example:
caldb_show_changes.pl t=2001/12/1 s=DC_DOCA ss=t_max i=Sector6
```

The arguments are:

`time` or `t` (required) All changes which occurred after this time will be reported.

`system` or `s` (optional) System specification. If omitted all systems will be searched.

`subsystem` or `ss` (optional) Subsystem specification. If omitted all subsystems will be searched.

`item` or `i` (optional) Item specification. If omitted all items will be searched.

`officer` or `o` (optional) Officer specification. If omitted, changes from all officers will be reported.

`runIndexTable` or `it` (optional) The name of the run index table to use. The default value is `calib.RunIndex`. See Section 3.2 above.

`fullComment` If set to a non-zero value all of the lines of the run index comment will be displayed, rather than the first three.

`quiet` (optional) If set to a non-zero value, the description of the search conditions will be omitted in the output.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
connecting to host clasdb.jlab.org
examining entries from system DC_DOCA
subsystem t_max
item Sector6
from run index table RunIndex
entered by any officer
going back to 2001/12/1
system = DC_DOCA, subsystem = t_max, item = Sector6
    runs 29808-29808, set 551, linked 2001-12-04 17:06:38 by nozarm
        comment: execution host:enigma
               host OS:LinuxRH7
               host time:Tue Dec  4 17:06:34 2001
    runs 29808-30299, set 551, linked 2001-12-04 17:37:54 by marki
        comment: Copied from run 29808 of RunIndex as of 2037-1-1.
```

3.2.14 caldb_link_constant_set.pl

Makes the correspondence between runs and constant sets. For a specified system/subsystem/item and a specified constant set ID (the `itemValueId`), the script will link the ID to the specified run range. Programs analyzing all runs in the run range will use the linked constants. When making the link, a comment is mandatory.

The usage message is:

```
usage:
caldb_link_constant_set.pl s=<system> ss=<subsystem> i=<item> min=<run min> \
    max=<run max> id=<item value id number> c=<comment> \
    it=<run index table name> [user=<MySQL user name>] \
    [hostname=<hostname of db server>] [help=<non-zero>]
```

```
alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  min or runMin
  max or runMax
  id or constantSetId
  c or comment
  it or runIndexTable for run index table name

example:
caldb_link_constant_set.pl s=mom_corr ss=theta_func i=sector6 min=100 \
    max=200 id=3 c="just a link example" it=calib_user.RunIndexJunk
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`runMin` (required) The minimum run of the run range specification.

`runMax` (required) The maximum run of the run range specification.

`constantSetId` (required) The identification number (ID) of the constant set to be associated with the specified run range. This must be an existing constant set. To see a list of existing constants sets use the `caldb_show_sets_item.pl` command, described in Section 3.2.7.

`comment` (required) A text comment. The comment should address the reasons for applying these constants to the specified run range rather than the way in which they were produced.

`runIndexTable` or `it` (required) The name of the run index table to use. See Section 3.2 above.

`user` (optional) The username to use when connecting to the MySQL server. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
runIndexId = 186189
```

The example links constant set 3 of the `mom_corr/theta_func/sector6` item to runs 100 through 200 inclusive in the run index `calib_user.RunIndexJunk`. The printout identifies the entry in that table that makes the link.

3.2.15 caldb_write_constant_set.pl

Writes a set of constants into a constant set table. The relevant system/subsystem/item and the name of the file which contains the constants must be specified. There are also two required comment fields: source run minimum and a general comment. Note that this script does not associate the constants with a run range. (See `caldb_link_constant_set.pl` or `caldb_write_and_link.pl` for a scripts which do that.)

The usage message is:

```
usage:
caldb_write_constant_set.pl s=<system> ss=<subsystem> i=<item> \
    c=<comment> [srmin=<source run min>] [srmax=<source run max>] \
    f=<file with constants> [hostname=<hostname of db server>] \
    [help=<non-zero>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  srmin or sourceRunMin
  srmax or sourceRunMax
  c or comment
  f or inputFile

example:
caldb_write_constant_set.pl s=mom_corr ss=theta_func i=sector6 \
    srmin=1000 srmax=2000 c="just an example" f=const_file.dat \
    hostname=claspc2.jlab.org
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`sourceRunMin` (optional) A comment field that records the minimum run used in the production of the constants. This value does not affect the run range to which constants are applied. It is purely informational.

`sourceRunMax` (optional) A comment field that records the maximum run used in the production of the constants. This value does not affect the run range to which constants are applied. It is purely informational.

`comment` (required) A text comment. The comment should address the production of the constants being added, rather than the run ranges to which they should be applied.

`inputFile` (required) The name of the file that contains the calibration constants themselves. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
Data from inputFile const_file.dat loaded to claspc2.jlab.org
(itemValueId = 9)
```

This tells us that the new constant set ID (`itemValueId`) in the constant set table (`mom_corr_theta_func_sector6` in this example) is 9.

3.2.16 caldb_write_and_link.pl

Does both a write of constants to the apropriate constant set table and links the resulting constant set to the specified run range. The system/subysystem/item, run range, and name of the file that contains the constants must be specified. In addition the comment for the run index and the source run#(srmin) must be supplied. If you want to write and link separately, use `caldb_write_constant_set.pl` and/or `caldb_link_constants_set.pl`.

The usage message is:

```
usage:
caldb_write_and_link.pl s=<system> ss=<subsystem> i=<item> min=<run min> \
    max=<run max> ci=<comment for run index> f=<file with constants> \
    it=<run index table name> \
    [srmin=<source run min>] [srmax=<source run max>] \
    [cc=<comment for constant set table>] \
    [user=<MySQL user name>] \
    [hostname=<hostname of db server>] [help=<non-zero>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  min or runMin
  max or runMax
  ci or commentIndex
  srmin or sourceRunMin
  srmax or sourceRunMax
  cc or commentConstants
  f or inputFile
  it or runIndexTable for run index table name

example:
```

```
caldb_write_and_link.pl s=mom_corr ss=theta_func i=sector6 min=1000 \
    max=2000 srmin=1400 srmax=1450 f=const_file.dat \
    it=calib_user.RunIndexJunk \
    ci='linking new constants' cc='creating new constants' \
    hostname=claspc2.jlab.org
```

The arguments are:

`system` (required) The system specification.

`subsystem` (required) The subsystem specification.

`item` (required) The item specification.

`runMin` (required) The minimum run of the run range specification.

`runMax` (required) The maximum run of the run range specification.

`commentIndex` (required) Text comment that will go into the run index table. This is same as the comment argument of `caldb_link_constant_set.pl` script described in Section 3.2.14. The text will serve as a default string for the `commentConstants` argument.

`inputFile` (required) The name of the file that contains the calibration constants themselves. See Section 3.2 above.

`runIndexTable` or `it` (required) The name of the run index table to use. See Section 3.2 above.

`sourceRunMin` (optional) A comment field that records the minimum run used in the production of the constants. This value does not affect the run range to which constants are applied. It is purely informational.

`sourceRunMax` (optional) A comment field that records the maximum run used in the production of the constants. This value does not affect the run range to which constants are applied. It is purely informational.

`commentConstants` (optional) Text comment that will go into the constant set table. This is same as the comment argument of `caldb_write_constant_set.pl` script described in Section 3.2.15. If not supplied, the value of the `commentIndex` argument is used.

`user` (optional) The username to use when connecting to the MySQL server. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
Data from inputFile const_file.dat loaded to claspc2.jlab.org
calib_user.RunIndexJunk (runIndexId = 194428, itemValueId = 6)
```

This confirms the hostname, `claspc2.jlab.org`, and the run index table, `calib_user.RunIndexJunk`. For the "write", the new constant set ID (`itemValueId`) in the constant set table (`mom_corr_theta_func_sector6` in this example) is 6. For the "link", the new run index ID is 194428.

3.2.17 caldb_make_run_index.pl

Makes a new run index, copying entries from the main run index. Use this command to create private run indices. A partial copy, valid for a user-selected range of runs, can be made.

Note that all relevant rows, even those that were only valid in the past, will be copied. Once a copy is made, changes to the copy can be made. The history mechanism will work on the copy transparently.

```
usage:
caldb_make_run_index.pl table=<run index table name> [db=<database name>] \
    [min=<min. run no.>] [max=<max. run no.>] \
    [hostname=<MySQL server hostname>] [user=<MySQL username>] \
    [help=<non-zero>]
```

The arguments are:

`table` (required) The MySQL table name for the new run index. Your choice. Recall that the main run index table is called `RunIndex`. Descriptive names are good.

`db` (optional) The name of the MySQL database in which the new table should be placed. Recall that the main database is called `calib`. You cannot put your new table there unless you are an officer (as described in Section 6). Make sure that you have create privilege in the database you specify. If `db` is missing, the database will default to `calib_user`. See Section 6 for a description of the `calib_user` database.[(c)]

`min` **(optional)** The lowest run for which the copy will be complete. If a run index row is relevant only for runs less than `min`, that entry will not be copied to the new run index table. If `min` is missing the default is $-\infty$.

`max` **(optional)** The highest run for which the copy will be complete. If a run index row is relevant only for runs strictly greater than `max`, that entry will not be copied to the new run index table. If the `max` is missing the default is $+\infty$.

`hostname` **(optional)** The name of the MySQL database server to use. If `hostname` is not supplied, the server will default to `clasdb.jlab.org`.

`user` **(optional)** The username to use when connecting to the MySQL server. If `user` is not supplied, the username will default to `clasuser`.

`help` **(optional)** If the `help` argument is non-zero, the usage message will be printed.

[(c)] Here we are using the technical meaning of "database" in MySQL.

3.2.18 caldb_copy_ranges.pl

Copies effective run ranges from one run index table to another. This makes the run-number/constant-set assignments for the destination run index exactly the same as that for the source run index for the run ranges and item(s) specified. In other words, analysis done with the destination run index will yield the same constants as that done with the source run index (again, for the specified run ranges and item(s)).

If a specific choice of system, subsystem and item are given then only the ranges for that item will be copied. All items in a subsystem can be copied by specifying `allitems` for the item. All items in a system can be copied by specifying `allsubsystems` for the subsystem and `alltitems` for the item.

If only a subset of the run ranges are to be copied, then a minimum and maximum run must be specified. All run ranges from the source run index which fall within these bounds will be copied. If the lowest and highest run range found in the source run index do not match the user-specified minimum and maximum runs, then the copied run ranges will be truncated to respect the user's specification. For example, if the source run ranges are 1000-1999, 2000-2999 and 3000-3999 and the user requests a copy of from run 1500 to run 3499 then the resulting copied run ranges will be 1500-1999, 2000-2999 and 3000-3499 in the destination run index. Note in this example that the constant assignments for runs 1000-1499 and runs 3500-3999 in the destination run index will remain unchanged. If all run ranges are to be copied, then the `allruns` argument must be set. In this case the minimum and maximum runs should be omitted. Use `caldb_show_run_ranges.pl`, described in Section 3.2.8, to display a list of effective run ranges.

After a summary of the requested copy transaction is printed, the user is prompted for confirmation before the operation is performed.

There is an argument, `nowrite`, that, when set to a non-zero value, will suppress all writes to the database. This can be used to "practice" copy operations; the write operations requested will be reported on the screen without actually being performed.

The usage message is:

```
purpose: Copy a set of run ranges from one run index to another. Do so
         for one item, all items in a subsystem, or all items in a system.

usage:
caldb_copy_ranges.pl s=<system name> ss=<subsystem name | 'allsubsystems'> \
    i=<item name | 'allitems'> \
    [min=<run min> max=<run max> | allruns=<non-zero>] \
    ris=<source run index table name> \
    rid=<destination run index table name> [time=<time strobe>] \
    [user=<MySQL user name>] [hostname=<hostname of db server>] \
    [nowrite=<non-zero>] [noprompt=<non-zero>] [help=<non-zero>]

note: use nowrite=1 to practice using this script

alternate flag names:
  s or system for system name
```

```
  ss or subsystem for subsystem name
  i or item for item name
  min or runMin
  max or runMax
  ris or runIndexSrc for source run index table name
  rid or runIndexDest for destination run index table name

example:
caldb_copy_ranges.pl s=DC_DOCA ss=t_max i=Sector1 ris=RunIndex \
  rid=calib_user.RunIndexMarkI min=23602 max=23682
```

The arguments are:

`system` or `s` (required) System to copy.

`subsystem` or `ss` (required) Subsystem to copy. Specify `allsubsystems` if run ranges for all subsystems in the specified system are to be copied. In this case you must specify `item=allitems`.

`item` or `i` (required) Item to copy. Specify `allitems` if run ranges for all items in the specified subsystem(s) are to be copied.

`runMin` or `min` (required if `allruns` is omitted) Minimum run to copy. The lower bound of the lowest run range copied from the source run index will modified (if necessary) to match this value.

`runMax` or `max` (required if `allruns` is omitted) Maximum run to copy. The upper bound of the highest run range copied from the source run index will be modified (if necessary) to match this value.

`allruns` (required if `runMin` and `runMax` are omitted) Copy all run ranges from the source run index to the destination run index.

`runIndexSrc` or `ris` (required) The name of the source run index. If the database is not specified, `calib` is assumed.

`runIndexDest` or `rid` (required) The name of the destination run index. The database is not specified, `calib` is assumed.

`time` (optional) Effective time of query on the source run index. Defaults to retrieving the latest version.

`user` (optional) Username to use when connecting to the MySQL server. Defaults to the value of the `$USER` environment variable.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`nowrite` (optional) A non-zero value will supress the actual writes to the database. Useful for previewing the results of a copy request without actually doing one.

`noprompt` (optional) A non-zero value will supress the prompt of the user for confirmation before a copy is done. Useful for using this script inside another script.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
note: use nowrite=1 to practice using this script
info: Copying all run ranges from 23602 to 23682
info: on host clasdb.jlab.org
info: for system DC_DOCA
info: subsystem t_max
info: item Sector1
info: from run index table calib.RunIndex
info: as of the present time
info: to run index table calib_user.RunIndexMarkI
info: writing is enabled
Are you sure you want to do this? (yes/no) yes
info: subsystem = t_max
info: item = Sector1
info: linking runs 23602-23621 to constant set 330, runIndexId = 147713
info: linking runs 23622-23624 to constant set 331, runIndexId = 147714
info: linking runs 23625-23681 to constant set 332, runIndexId = 147715
info: linking runs 23682-23682 to constant set 333, runIndexId = 147716
```

This tells us that four new specified run ranges were defined for the specified system/-subsystem/item (`DC_DOCA/t_max/Sector1` in this case) in the destination run index table (`calib_user.RunIndexMarkI`). The constant set ID's shown are those taken from the source run index table (`calib.RunIndex`). The new entries have `runIndexId`'s from 147713 to 147716 inclusive.

3.2.19 caldb_copy_run.pl

Copies constant set assignments from the source run index to the destination run index. In the source run index, the assignments corresponding a single specified run are used. In the destination run index, these assignments are copied to a specified range of runs.

Note that the source and destination run indices may be the same; this case may be useful when initializing constants for a new run period. For example you may want to use constants from run 28000 of the main run index for runs 30000 to 40000 of the main run index as a start at developing constants for the run range 30000-40000.

There is an argument, `nowrite`, that, when set to a non-zero value, will suppress all writes to the database. This can be used to "practice" copy operations; the write operations requested will be reported on the screen without actually being performed.

The usage message is:

```
function: Links constants to a range of runs (runMin to runMax). Constants
          chosen are those assigned to the specified source run number.
```

```
note: use nowrite=1 to practice using this script
usage:
caldb_copy_run.pl  min=<run min> max=<run max> r=<runSource> \
    rid=<destination run index table> \
    [s=<system>] [ss=<subsystem>] [i=<item>] \
    [ris=<source run index table>] \
    [t=<time of validity>] \
    [hostname=<hostname of db server>] [nowrite=<non-zero>] \
    [help=<non-zero>] [q=<non-zero to suppress some info>]


alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  r or runSource for reference run number
  min or runMin
  max or runMax
  ris or runIndexSource for source run index table name
  rid or runIndexDest for destination run index table name
  t or time for time of validity
  q or quiet for suppression of extra information

example:
caldb_copy_run.pl s=DC_DOCA ss=xvst_params i=SL1 ris=RunIndex \
    rid=calib_user.RunIndexJunk r=28000 min=30000 max=40000
```

The arguments are:

`runMin` (required) The minimum run of the run range specification.

`runMax` (required) The maximum run of the run range specification.

`runSource` (required) The run number of initial run.

`system` (optional) If not defined all systems will be copied.

`subsystem` (optional) If not defined all subsystems will be copied.

`item` (optional) If not defined all items will be copied.

`runIndexDest` (required) The destination runIndex table.

`runIndexSource` (optional) The source runIndex table. If not defined `calib.RunIndex` will be used.

`time` (optional) The historical time to use in database queries. The default value is (effectively) the current time. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`nowrite` (optional) If set to a non-zero value, no new links will be put into the destination run index. Use this argument to preview the results of a given set of arguments.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

`quiet` (optional) A non-zero value will suppress printing of column headings and other extra information.

The example command in the usage message above gives:

```
hostname=clasdb.jlab.org, user=clasuser
Connection successful: handle: DBI::db=HASH(0x81d7e6c)
info: in calib_user.RunIndexJunk, runs 30000-40000 for DC_DOCA, xvst_params, SL1
      now are linked with constant set ID 344, runIndexId is 186843
```

The example links constant sets used by run 28000 of `calib.RunIndex` for `DC_DOCA`, `xvst_params`, `SL1` to the run range 30000-40000 of the run index `calib_user.RunIndexJunk`.

3.2.20 caldb_export_run.pl

Exports sets of constants from the source run index to set of text files in certain directory. In the source run index, the assignments corresponding a single specified run are used. All constants are copied to the destination directory.

For example you may want to make a backup of certain set of constants from run 28000 from some host.

The usage message is:

```
function: Exports constants valid for a given run <run> to a directory
caldb_export<run>

usage:
caldb_export_run.pl   r=<runSource> \
    [s=<system>] [ss=<subsystem>] [i=<item>] \
    [it=<source run index table>] \
    [t=<time of validity>] \
    [hostname=<hostname of db server>]  \
    [help=<non-zero>] [q=<non-zero to suppress some info>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  r or runSource for reference run number
  it or runIndexTable for source run index table name
  q or quiet for suppression of extra information


example:
caldb_export_run.pl s=DC_DOCA ss=xvst_params i=SL1 it=RunIndex r=28000
```

The arguments are:

`runSource` (required) The run number of initial run.

`system` (optional) If not defined all systems will

`subsystem` (optional) If not defined all subsystems will be copied.

`item` (optional) If not defined all items will be copied.

`runIndex` (optional) The source runIndex table. If not defined `calib.RunIndex` will be used.

`time` (optional) The historical time to use in database queries. The default value is (effectively) the current time. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

`quiet` (optional) A non-zero value will suppress printing of column headings and other extra information.

The example command in the usage message above gives:

```
/group/clas/tools/caldb/caldb_export_run.pl s=DC_DOCA ss=xvst_params i=SL1
it=RunIndex r=28000
using runIndexTable = RunIndex from clasdb.jlab.org
Create new dir caldb_export28000
Now exporting DC_DOCA xvst_params SL1 of RunIndex as of 2037-1-1 to
    caldb_export28000
Create new  caldb_export28000/DC_DOCA
```

The example creates (if not available) a directory caldb_export28000 with a subdirectory DC_DOCA and a text file xvst_params_SL1 with constants for subsystem `DC_DOCA`, `xvst_params` (one directory per subsystem if not given) and item `SL1` (one file per item if not given) valid for run=28000 from the runindex table calib.RunIndex from clasdb.jlab.org (default).

3.2.21 caldb_import_set.pl

Imports sets of constants from the given directory (created by export, see 3.2.20 above) to the given destination run index. The validity range of imported set is defined by command line parameters min and max.

For example you may want to import a backup of certain set of constants from saved files (directory tree created by the export) to a different runindex and/or with a different validity range.

The usage message is:

```
function: Import constants from a given directory caldb_export<srmin>

usage:
caldb_import_set.pl   srmin=<runSource> \
    [s=<system>] [ss=<subsystem>] [i=<item>] \
    [rid=<destination run index table>] \
    [t=<time of validity>] \
    [hostname=<hostname of db server>]  \
    [help=<non-zero>] [q=<non-zero to suppress some info>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  r or runSource for reference run number
  rid or runIndexDest destination run index table name
  q or quiet for suppression of extra information
  srmin or sourceRunMin source run (defines the directory for input)


example:
caldb_import_set.pl s=DC_DOCA ss=xvst_params i=SL1 rid=calib_user.RunIndextestmap
srmin=28000 min=40000 max=40001 ci='test'
```

The arguments are:

`srmin` (required) The source run number defining the directory name.

`runIndexDest` (required) The destination runIndex table.

`system` (optional) If not defined all systems will

`subsystem` (optional) If not defined all subsystems will be copied.

`item` (optional) If not defined all items will be copied.

`time` (optional) The historical time to use in database queries. The default value is (effectively) the current time. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

`quiet` (optional) A non-zero value will suppress printing of column headings and other extra information.

The example command in the usage message above gives:

```
> /group/clas/tools/caldb/caldb_import_set.pl s=DC_DOCA ss=xvst_params
i=SL1 rid=calib_user.RunIndextestmap srmin=28000 min=40000 max=40001 ci='test'
Reading constants from dir ./caldb_export28000
using runIndexTable = calib_user.RunIndextestmap from clasdb.jlab.org
Now importing DC_DOCA xvst_params SL1 of calib_user.RunIndextestmap
from ./caldb_export28000

0 -1.73055 159.4 0 0.85 1.23981 -7.45891 23.0195 -27.5138 9 4.539 4.285
3.253 0 1.1 0 0 0 0 0 0 0 0 0
Data from inputFile ./caldb_export28000/DC_DOCA/xvst_params_SL1 loaded
to clasdb.jlab.org calib_user.RunIndextestmap
(runIndexId = 34711, itemValueId = 883)
```

The example imports constant set for `DC_DOCA`, `xvst_params`, `SL1` from the directory ./caldb_export28000 to runindex file `calib_user.RunIndextestmap` with the validity run range 40000-40001.

3.2.22 caldb_show_tables.pl

Shows all tables in the given database. The user can choose db server, MySQL username and password. The default is to connect to clasdb.jlab.org as clasuser with no password.

This is actually a very generic script. It can be used to query databases outside the CalDB.

The usage message is:

```
usage:
caldb_show_tables.pl database=<database name>
    [hostname=<hostname of db server>] [user=<db username>]
    [password=<db password>] [verbose=<non-zero for verbose output>]
    [help=<non-zero for usage message>]

example:
caldb_show_tables.pl database=calib_user
```

The arguments are:

`database` (required) The database specification.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`user` (optional) The username to use when connecting to the MySQL server. The default is `clasuser`.

`password` (optional) The password to use when connecting to the MySQL server. The default is the null string.

`verbose` (optional) A non-zero value enables more detailed printout.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
EG2K_DC_calib_DAVE
RunIndex
RunIndexEG1Pass1
.
.
.
privateRunIndex
private_runindex
```

This is the complete list (notwithstanding the ellipses) of tables in the `calib_user` database of the MySQL server on `clasdb.jlab.org`. As such, it also happens to be a list of all of the private run indices in the CalDB (that is only because `calib_user` was chosen as the database in this example; other choices would display different tables with different interpretations).

3.2.23 caldb_add_officer.pl

Adds a new officer by making appropriate additions to the access control tables of the MySQL server. See Section 6 for a description of officer privileges. You must be a Database Manager to run this script and it must be run locally on the database server.

The usage message is:

```
usage:
caldb_add_officer.pl new_user=<new user>

example:
caldb_add_officer.pl new_user=foo
```

The argument is:

`new_user` (required) Officer's MySQL username.

3.2.24 caldb_add_system.pl

Adds a new system.

The usage message is:

```
function: This script adds a system to CalDB.

usage:
caldb_add_system.pl s=<system> c=<description> \
      [hostname=<hostname of db server>]

alternate flag names:
  s or system for system name
  c or comment for system description
```

```
example:
caldb_add_system.pl  s=newGreatSystem c='important system'
```

The arguments are:

`system` (required) The new system name.

`comment` (required) Description of that system.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

The example command in the usage message above gives:

```
info: new system newGreatSystem created on clasdb.jlab.org server with
      systemId 51.
```

The example creates an entry newGreatSystem in the System table in the calib database.

3.2.25 caldb_add_subsystem.pl

Adds a new subsystem to an existing system.

The usage message is:

```
function: This script adds a subsystem to the CalDB. An existing system must
          be specified.

usage:
caldb_add_subsystem.pl s=<system> ss=<subsystem> c=<description> \
   [hostname=<hostname of db server>]

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  c or comment for subsystem description

example:
caldb_add_subsystem.pl  s=newGreatSystem ss=somesubs \
    c='important subsystem'
```

The arguments are:

`system` (required) The system name.

`subsystem` (required) The new subsystem name.

`comment` (required) Description of that system.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

The example command in the usage message above gives:

```
info: new subsystem somesubs created for newGreatSystem on clasdb.jlab.org
      with subsystemId 272.
```

3.2.26 caldb_add_item.pl

Adds a new item to an existing subsystem.

The usage message is:

```
function: This script adds an item to the CalDB and creates the
          corresponding constant set table

usage:
caldb_add_item.pl s=<system>  ss=<subsystem>  i=<item> c=<description> \
    type=[float|int|char|intblob] length=<length of constant array>  \
    [hostname=<hostname of db server>]

alternate flag names:
  s  or system for system name
  ss or subsystem for subsystem name
  i  or item for item name
  c or comment for system description

example:
caldb_add_item.pl s=newGreatSystem c="int_item8" ss=somesubs i=item8 \
    type=int length=10
```

The arguments are:

`system` (required) The system name.

`subsystem` (required) The subsystem name.

`item` (required) The new item name.

`type` (required) The item type.

`length` (required for int/float) The item length.

`comment` (required) Description of that item.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

The example command in the usage message above gives:

```
info: new item item8 (itemId=875) created for system newGreatSystem,
      subsystem somesubs on clasdb.jlab.org.
```

As a result item item8 is created for somesubs subsystem of newGreatSystem system.

3.2.27 caldb_delete_items.pl

Deletes an item or items. If system, subsystem and item are all specified, then only that one item will be deleted. If only system and subsystem are specified, that one subsystem will be deleted including all items that belong to it. If only system is specified, then the entire system, including its subsystems and their items, will be deleted. When an item is deleted, all entries in the main run index and all constants sets associated with that item are deleted. The script does not attempt to delete the item from copies of the run index.

This script can only be run by the dbmanager on the server of the local host.

This action works outside the history mechanism; the deleted items can only be restored from backup copies of the CalDB database.

The usage message is:

```
function: deletes systems, subsystems and items from the CalDB. Must be run
          by dbmanager on localhost.

usage:
caldb_delete_items.pl s=<system> [ss=<subsystem>] [i=<item>]

alternate flag names:
  s  or system for system name
  ss or subsystem for subsystem name
  i  or item for item name

example:
caldb_delete_items.pl s=newGreatSystem ss=badsubs i=item9
```

The arguments are:

`system` or `s` (required) System which contains the items to be deleted. If all subsystems in the system are deleted (see the `subsystem` argument), the system itself will be deleted.

`subsystem` or `ss` (optional) Subsystem which contains the items to be deleted. If all items in the subsystem are deleted (see the `item` argument), the subsystem itself will be deleted. If omitted, then all subsystems in the specified system will be deleted.

`item` or `i` (optional) Item to be deleted. If omitted, then all items in the specified subsystem will be deleted.

3.2.28 caldb_delete_changes.pl

Deletes recent changes to a run index. The user specifies a system/subsystem/item combination, an officer whose changes will be examined, and a time in the past, back to which changes will be deleted. If the item is specified as `allitems`, then all items in the subsystem will be searched for links to delete. If the subsystem is specified as `allsubsystems`, then all subsystems in the system will be searched (`allitems` is mandatory in this case).

N. b., this script, if misused, can be very dangerous to your run index. It deletes links that have been added in the past, and thus defeats the built-in history mechanism of the CalDB. Links which have been deleted by this script cannot be recovered easily. In addition, by destroying the history record of the run index, users that are using a fixed time to access the run index may get different results after links are deleted if their fixed time is after the deletion time specified in the input argument. Recall that users using a fixed time expect the results to be time-independent.

The script is provided to allow users to delete large, inadvertently-added sets of links, and if deemed necessary, should be used as soon as the mistake is realized to minimize the amount of history that gets erased. If a lot of time (on the order of days) has passed since the mistake, it might be better to use `caldb_copy_ranges.pl`, described in Section 3.2.18, to render the latest version of a set of items the same as that at a time in the past, preserving all of the history.

Because of this danger, only officers can run this script, and then only on run indices stored in the `calib_user` database. To delete links from the main run index or any other run index in the `calib` database, you must ask a database manager to do it.

The constant sets themselves are not deleted, only the links (references to them) in the selected run index. To view all existing constant sets, independent of how they may or may not be linked, use `caldb_show_sets_item.pl`, described in Section 3.2.7.

The user is prompted twice. Once to approve a recap of the chosen input parameters, once to approve deletion of the results of the search for recent links.

The usage message is:

```
purpose: Delete run index entries back to a specified time for
         specified items entered by a specified officer.

usage:
caldb_delete_changes.pl s=<system name> \
    ss=<subsystem name | 'allsubsystems'> i=<item name | 'allitems'> \
    a=<author name> t=<time to go back to> it=<run index table name> \
    [hostname=<hostname of db server>] [nodelete=<non-zero>] \
    [help=<non-zero>]

note: use nodelete=1 to practice using this script

alternate flag names:
  s or system for system name
  ss or subsystem for subsystem name
  i or item for item name
  a or author for author name
  t or time for time to go back to
  it or runIndexTable for run index table name

example:
caldb_delete_changes.pl s=SC_CALIBRATIONS ss=delta_T i=paddle2paddle \
```

```
it=calib_user.RunIndexJunk a=vipuli t=2001/10/5 hostname=claspc2
```

The arguments are:

`system` or `s` (required) System to search for links to delete.

`subsystem` or `ss` (required) Subsystem to search for links to delete. Specify `allsubsystems` if all subsystems in the specified system are to be searched. In this case you must specify `item=allitems`.

`item` or `i` (required) Item to search for links to delete. Specify `allitems` if all items in the specified subsystem(s) are to be searched.

`author` or `a` (required) Only links by this author will be considered for deletion.

`time` or `t` (required) Only links made after this time will be considered for deletion. See Section 3.2 for a description of the time format.

`runIndexTable` or `it` (required) The name of the run index table from which links will be deleted. See Section 3.2 for a description of the specification of run indices.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`nodelete` (optional) If this argument is set to a non-zero value, then no links will be deleted. The script will nonetheless performed the requested search for links and the results will be reported. Use this argument to preview the result of the deletion operation requested. In this mode, the user will not be prompted for approval of the (benign) search.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
note: use nodelete=1 to practice using this script
info: connecting to host claspc2
info: as user marki
info: deleting entries from system SC_CALIBRATIONS
info: subsystem delta_T
info: item paddle2paddle
info: from run index table calib_user.RunIndexJunk
info: entered by user vipuli
info: going back to 2001/10/5
info: deletion is enabled
Is this what you want to do? (yes/no) yes
info: searching for entries to delete
info: subsystem = delta_T
info: item = paddle2paddle
info: runs 27352-27499, set 163, linked 2001-10-11 19:13:13 by vipuli
```

```
      comment: adjustp2p
info: runs 27352-27499, set 167, linked 2001-10-29 10:01:43 by vipuli
      comment: pass0 p2p adjust
Are you sure you want to delete these entries? (yes/no) yes
info: deleting id=179757
info: deleting id=181871
```

3.2.29 caldb_drop_table.pl

Drops a table in the `calib_user` database on a given server. These tables are usually private run indices. As such they are in danger of getting old, stale and useless. This script allows the user to clean up old work.

The usage message is:

```
Purpose:
    Drops a table from the calib_user database.

usage:
    caldb_drop_table.pl table=<table name> \
        [hostname=<hostname of db server>] \
        [help=<non-zero for usage message>]

example:

caldb_drop_table.pl table=RunIndexJunk
```

The arguments are:

`table` (required) The name of the table to drop.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
INFO: Dropping table RunIndexJunk from the calib_user database
      on clasdb.jlab.org.
Is this OK? (yes/no): yes
INFO: Table dropped.
```

See Section 3.2.22 for a script to use to get a listing of tables in a given database.

3.3 Browsing CalDB on the Web

There is a CGI script that generates web pages for browsing the contents of the CalDB. It uses the Perl API exclusively for all database interactions. The source is in[(d)]

[(d)] At JLab, `CLAS_TOOLS` is `/group/clas/tools`. To get tools from CVS, the command is `cvs checkout tools`.

```
$CLAS_TOOLS/caldb/cq.pl .
```

The URL is

```
http://clasweb.jlab.org/cgi-bin/caldb/cq.pl .
```

3.4 Graphical Interface for Viewing Constants

There is a graphical interface for viewing calibration constants. It is based on the Java CalDB class library (documentation to come). You can view a particular constant from a constant set as a function of run number or view all constants in a constant set as a function of channel number. To get instructions on its use, see

```
http://www.jlab.org/~avakian/caldbJ .
```

4 Backward Compatibility with the Map

At this writing there is a large amount of CLAS analysis code that uses the Mapmanager to read and write calibration constants. There are two methods by which this legacy code can operate with the CalDB:

1. Link to Map-emulating routines that access the CalDB directly. See Section 4.1.
2. Write Map files from the CalDB and use the original Map libraries. See Section 4.2

4.1 Map-Emulation Library

A set of access routines have been written that mimic the calling arguments of the Mapmanager C subroutines. They access the database directly, performing the same functions on the database as the Map routines performed on the Map files. The files are in the CLAS CVS repository under `packages/caldb/Map`. Programs which formerly linked to `libmapmanger.a` can link to the combination of `libcaldbMap.a`, `libclasutil.a`, `libmysqlclient.a`,[e]. `libtcl.so` and `libdl.so`. On some Linux systems, you must also link to the system-supplied `libz.a`. By linking in these libraries, old Map-based code can be used without modification. The routines in utilities/maputil that are relevant for CalDB have been linked and tested with the new library. The converted routines are:

- `get_first`
- `get_map_char`
- `get_map_float`
- `get_map_int`

[e] At JLab, you can the MySQL libraries in directories under `/apps/mysql`. For more information on using MySQL see the Offline FAQ. The URL is `http://clasweb.jlab.org/offline/offline_faq/`.

- get_map_item
- put_map_char
- put_map_float
- put_map_int

What follows is a snippet of the makefile for these maputil routines:

```
ifndef MAP
  LIBNAMES = caldbMap$(ADD_DEBUG) clasutil$(ADD_DEBUG) mysqlclient
  ifeq "$(OS_NAME)" "LinuxRH6"
    LIBNAMES += z
  else
    ifeq "$(OS_NAME)" "SunOS"
      SHARED_LIBS += -lsocket -lnsl
    endif
  endif
  SHARED_LIBS += -ltcl -ldl
else
  LIBNAMES = mapmanager$(ADD_DEBUG)
endif
```

There are several environment variables that will allow the Map-emulation routines to access non-default versions of the calibration constants. They are `CLAS_CALDB_HOST`, `CLAS_CALDB_RUNINDEX` and `CLAS_CALDB_TIME`. They are described in detail in Section 5.2.2.

4.2 Translating Between the Map and the CalDB

There are scripts in the `$CLAS_TOOLS/caldb` area to translate from a set of Map files to the MySQL database and from MySQL back to the map.

4.2.1 Map to CalDB

`$CLAS_TOOLS/caldb/map2db.pl` is a Perl script that will translate a set of Map files into the MySQL database. It operates on all files with the `.map` extension in `$CLAS_PARMS/Maps`. It assumes that the server does not have a pre-existing verion of the CalDB installed. All database tables are placed in the "calib" database. This script was used to create the original version of the CalDB, and in principle, should not have to be run ever again.

`$CLAS_TOOLS/caldb/mapfile2db.pl` is intended for general use. It copies constants from a single Map file into the CalDB. Single items can be copied, or all items in a subsystem, or all items in all subsystems. In addition a run range can be specified; only Map runs within the range will have their constants copied.

A note on the constants selected for copying and their run assignments in the CalDB. The user must specify explicitly whether all of the runs are to be copied (`allruns=1`) or whether a run range should be copied (set `runMin` and `runMax`). If a run range is specified,

Map Run	Effective Map Range	CalDB Range
1000	1000-1999	not copied
2000	2000-2999	2500-2999
3000	3000-3999	3000-3999
4000	4000-4999	4000-4500
5000	5000-∞	not copied

Table 3: Example of run range assignments when copying from the Map to the CalDB using `mapfile2db.pl`. Here, the user-specified run range is 2500-4500.

then all Map constants that are relevant for runs `runMin` through `runMax` inclusive will be copied. In the CalDB, the run range assignments will match exactly those found in the original Map, with the possible exception of the first and last set of constants. It may be the case that the user-specified `runMin` is larger than the lower bound of the lowest Map run range. In that case the assignment in the CalDB will truncate the run range, using the user-specified minimum rather than the Map-resident minimum. Likewise if the upper bound of the highest Map run range is greater than the user-specified `runMax`, the assigned CalDB range will use the user-specified value. Table 3 shows an example.

The usage message is

```
Purpose: copy constants from a single Map file into the CalDB.

Note: use nowrite=1 to practice using the script.

Usage:
mapfile2db.pl mapfile|m=<name of map file> \
    subsystem|ss=<subsystem name|allsubsystems> \
    item|i=<item name|allitems> \
    [allruns=<non-zero>}] [runMin|min=<minimum run number>] \
    [runMax|max=<maximum run number>] \
    [runIndex|ri=<name of target run index>] \
    [hostname=<hostname of MySQL server>] [nowrite=<non-zero>] \
    [help-<non-zero>]

Example:
mapfile2db.pl mapfile=/group/clas/parms/Maps/DC_DOCA.map \
    subsystem=t_max item=Sector6 runIndex=calib_user.RunIndexJunk \
    min=2063 max=8099
```

The arguments are:

`mapfile` or `m` (required) The name of the Map file that should be inserted into the `CalDB`.

`subsystem` or `ss` (required) The subsystem to be copied. If this argument is set to `allsubsystems` then all subsystems will be copied.

`item` or `i` (required) The item to be copied. If this argument is set to `allitems` then all items in a subsystem will be copied.

`allruns` (optional) If this argument is set to a non-zero value, then all runs in the Map file will be copied. See note above.

`runMin` or `min` (optional) Minimum Map run to copy. See note above.

`runMax` or `max` (optional) Maximum Map run to copy. See note above.

`runIndex` or `ri` (optional) Destination run index for the constants. See Section 3.2.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`nowrite` (optional) If set to a non-zero value, no constants will be written to the CalDB. Use this argument to preview the results of a given set of arguments.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

4.2.2 CalDB to Map

`$CLAS_TOOLS/caldb/db2map.pl` translates from the MySQL database to a set of Map files. The files are created in a local directory named with the specified minimum and maximum runs (*e. g.*, if `runMin=1200` and `runMax=2700` the name of the directory is `Maps_1200-2700`). These files can in turn be used with old binaries that read the Map. The new Map files will reflect the most recent constants stored in the CalDB. The user can request that only one system be translated, resulting in a single Map file. The default behavior is to translate all systems. Note that if any Map file already exists in the output directory, no action will be taken for the corresponding system (no new map file, no change in old one).

If all runs from the CalDB are to be put into the Map file, then use `runMin=1` and `runMax=1000000`. For other choices, the run range specification is best explained by example. If in the CalDB, the effective run ranges for a particular item are

CalDB Effective Ranges

minimum	**maximum**
1	999
1000	1999
2000	2999
3000	3999
4000	4999

and the run range specified to the script is `runMin=1500` and `runMax=3500`, then the resulting Map file will have entries for runs

Map Runs

1000
2000
3000

In words, for each CalDB effective run range which overlaps the input run range, one Map entry will be made. These Map entries will be made for the minimum run of the corresponding CalDB effective run range. Note that this has the obviously desireable effect that, for all runs in the user specified range, the constants read from the produced Map will be the same as those from the CalDB.

The script uses native Map binaries (*e. g.* `put_map_float`) to create new map files and populate them. It expects to find them in `$CLAS_BIN`. The user has to make sure that `$CLAS_BIN` points to an appropriate directory and not one that has binaries linked with the Map-emulation libraries. In releases before and including release-2-5, the relevant binaries use the native Map routines and are therefore appropriate when running this script. The script will check for a good value for `$CLAS_BIN` and will exit before starting if it does not find one.

The usage message is:

```
Purpose:
    Creates Map files from the CalDB. If system is specified only one
    Map is made. If not, Map files are made for all systems.

Warning:
    Make sure CLAS_BIN points to a directory with "real" Map utilities
    (not Map utilities linked with CalDB's Map emulation).  These are
    there up to and including release-2-5.

usage:
    db2map.pl runMin=<minimum run> runMax=<maximum run> \
      [system=<system name>] \
      [skip_run_control=<non-zero to skip RUN_CONTROL system>] \
      [runIndexTable=<run index table name>] \
      [hostname=<db server hostname>] [help=<non-zero for usage message]

Example:
    db2map.pl system=SC_CALIBRATIONS runMin=1 runMax=1000000
```

The arguments are:

`runMin` (required) The minimum run of the run range specification. (See above for details.)

`runMax` (required) The maximum run of the run range specification. (See above for details.)

`system` (optional) System specification. If this argument is present, only one Map file will be produced, corresponding to the specified system. By default all systems are translated into Map files.

`skip_run_control` (optional) If this argument is non-zero, then when looping over all systems, the RUN_CONTROL system will be skipped. This option is provided since this system takes a long time to translate.

`runIndexTable` or `it` (optional) The name of the run index table to use. The default value is `calib.RunIndex`. See Section 3.2 above.

`hostname` (optional) The hostname of the MySQL server to use. See Section 3.2 above.

`help` (optional) A non-zero value prints the usage message. See Section 3.2 above.

The example command in the usage message above gives:

```
SC_CALIBRATIONS
  atten_length
    left
      length=288, type=float
    right
      length=288, type=float
  atten_u
    left
      length=288, type=float
    right
      length=288, type=float
  .
  .
  .
  Yoffset
    value
      length=288, type=float
```

5 Alternate Versions of Constants

There are some situations where independent versions of the calibration constants are desirable.

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

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

3. **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.

In the first case, it is sufficient to choose a particular date and use that as the "as of" date in all calls to the API. Then the constants retrieved will not be sensitive to the latest contributions of detector calibrators.

5.1 Creating an Alternate Version

In all three cases listed above, the user can make a private copy of the run index table. The script

```
$CLAS_TOOLS/caldb/caldb_make_run_index.pl
```

will do this. See Section 3.2.17 for details of its use.

5.2 Using Alternate Versions

There are several ways to specify alternate versions of constants. The exact method depends on the type of program being run.

5.2.1 Argument to Command-Line Scripts

All of the command-line scripts described in Section 3.2 have an optional argument called `runIndexTable` (whenever appropriate). Using this argument, the user can use the run index table of choice, for both read and write operations.

5.2.2 Environment Variables for Map-emulation Routines

If the Map-emulation routines are used to read and write constants, then there are shell environment variables that enable use of an alternate versions of constants for all operations. In particular, these environment variables can be used to direct analysis programs to use a specific version of the constants other than the latest one contained in the main run index on the default server.

`CLAS_CALDB_HOST` If this variable is set, it will be used as the MySQL server. This will be useful for those using a mirror site (see Section 7.2). For example,

```
setenv CLAS_CALDB_HOST einstein.sr.unh.edu
```

If this variable is not set, the server will default to `clasdb.jlab.org`.

`CLAS_CALDB_RUNINDEX` If this variable is set, its value will be used as the run index table for all queries. For example,

```
setenv CLAS_CALDB_RUNINDEX calib_user.RunIndexMine
```

will access the `RunIndexMine` table of the `calib_user` database. If this variable is not set, the run index defaults to the main one, `RunIndex` of the `calib` database.

`CLAS_CALDB_TIME` If this variable is set, then all read operations will get the version of constants as of the specified time. The format is the same as that described in Section 3.2. For example,

```
setenv CLAS_CALDB_TIME 2001/6/12
```

will give the constants as they existed at midnight, June 12, 2001. If this variable is not set, then the latest set of constants will be retrieved.

5.2.3 Tcl Variables in RECSIS

RECSIS users can change the name of the run index table by a line in their recsis tcl initialization script. If the name of your index is `RunIndexMine` in the `calib_user` database, the syntax is:

```
set RunIndex_table calib_user.RunIndexMine
```

That will override any definition you have in the environment for `CLAS_CALDB_RUNINDEX`. Note that one uses the `set` command of vanilla Tcl, not the `setc` command of RECSIS Tcl.

The other environment variables have not yet been implemented as Tcl variables.

6 Access Control

There are several different levels of access to the database:

1. Database Manager (1 or 2 people)
2. Officers (1 or 2 per calibration system)
3. Users

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

- Users can
  - insert constants to the database
  - create private run index tables
  - modify (*i. e.*, insert, update, delete) private run index tables
  - read anything in the database
- Users cannot
  - delete constants from the public item value tables
  - drop or modify the public run index table
- Officers can, in addition
  - delete constants from the public item value tables
  - modify the public run index table
- Officers cannot
  - drop item value tables from the database
  - drop the public run index
- Database Managers can do anything to the database

user table			
Host	User	Password	Privileges
clasdb.jlab.org	dbmanager	non-NULL	all
%.jlab.org	officer_1	NULL	none
%.jlab.org	officer_2	NULL	none
...	...	...	...
NULL	clasuser	NULL	none

Table 4: MySQL user table. `dbmanager` must connect from `clasdb.jlab.org`. Officers must connect from a client in the `jlab.org` domain. `clasuser` must connect from a host named in the host table (see Table 5). The `dbmanager` must be password authenticated. Officers and `clasuser` do not need passwords. The `dbmanager` can do anything on the database server.

host table		
Host	Db	Privileges
%.jlab.org	calib%	Select
einstein.sr.unh.edu	calib%	Select
...	...	...

Table 5: MySQL host table. This is the list of hosts that we trust. `clasuser` can only connect from one of these hosts.

The MySQL access tables shown in Tables 4-7 accomplish these goals.

The restrictions described above only apply to tables in the `calib` database. In order to provide complete freedom for users to develop new calibrations schemes, another database `calib_user` is provided. All users are allowed to write to the `calib_user` database. Tables can be added and dropped, and rows in run index tables can be inserted, updated, and deleted.

db table			
Host	Db	User	Privileges
NULL	calib	officer_1	select, insert, create
NULL	calib	officer_2	select, insert, create
...	...	...	...

Table 6: MySQL db table. Officers can read and insert new rows to any table in the `calib` database. They can create new tables. They cannot delete rows or drop tables.

tables_priv table				
Host	Db	User	Table_name	Table_priv
NULL	calib	clasuser	item value table 1	select, insert
NULL	calib	clasuser	item value table 2	select, insert
...	...	...	...	...

Table 7: MySQL tables_priv table. `clasuser` can insert rows into any item value table.

7 Database Deployment

7.1 Authoritative Version

There will be one authoritative version of the database. All write operations must be performed on this JLab-resident copy. The access control scheme encourages this.

7.2 Replication: Maintaining a Mirror Site

Remote sites will want to replicate the database, to avoid having to read constants over the network from JLab (see Section 5.2.2). Quoting from the MySQL document[2]:

> Starting in Version 3.23.15, MySQL supports one-way replication internally. One server acts as the master, while the other acts as the slave. Note that one server could play the roles of master in one pair and slave in the other. The master server keeps a binary log of updates and an index file to binary logs to keep track of log rotation. The slave, upon connecting, informs the master where it left off since the last successfully propagated update, catches up on the updates, and then blocks and waits for the master to notify it of the new updates.
>
> Note that if you are replicating a database, all updates to this database should be done through the master!

To get replication going:

1. Look over the section "Replication in MySQL" of the "MySQL Database Administration" chapter of the MySQL manual to get a feeling for the concepts.

2. Make sure the MySQL server you are running is at version 3.23.29 or higher so that you avoid conflicts with older versions of the replication code.

3. Get a unique server ID number and the replication password from the CalDB manager (Mark Ito). You can view the currently assigned ID's at

   `clasweb.jlab.org/offline/server_id_hallb.html`.

4. Put the following lines in your `my.cnf` file[f] under the `[mysqld]` section:

[f] Not all installations of MySQL will have a `my.cnf` file. If none exists, you will have to create one. For more information, see the section "my.cnf Option Files" of the chapter "MySQL Database Administration" of the MySQL manual.

```
server-id=<your-server-id>
master-host=clasdb.jlab.org
master-user=repl
master-password=<the-password>
master-port=3306
```

5. Install the snapshot[g] of the CLAS database server in your MySQL data directory. The current correct snapshot is

   `/group/clas/mysql_data/mysql_snapshot.tar.gz`

   on the JLab CUE. Note that this is actually a symbolic link to the correct snapshot. Copying, gunzip'ing and un-tar'ing this file makes a copy of the CLAS databases in the current working directory, therefore this should be done in your servers MySQL data directory. For example, if your data directory is `/var/lib/mysql`, and you copied the snapshot to say

   `/scratch/mysql_snapshot.tar.gz`,

   then as root you type:

   ```
   cd /var/lib/mysql
   tar zxvf /scratch/mysql_snapshot.tar.gz
   ```

   Now the local server has a complete (though out-of-date) copy of all of the CLAS databases (starting replication will bring them up-to-date).

6. Stop and restart the MySQL server. On most Linux systems this can be done as root if you type:

   ```
   # /etc/rc.d/init.d/mysqld stop
   # /etc/rc.d/init.d/mysqld start
   ```

   This forces a re-read of the my.cnf file. Until this is done, the local server will not act like a slave.

Replication should start on your local server. You should check this in your MySQL error log. For vanilla rpm installation the log file is `/var/lib/mysql/your_host_name.err`. You should see a message like

```
020403 15:43:36  Slave: connected to master 'repl@clasdb.jlab.org:3306',
replication started in log 'FIRST' at position 4
```

[g] What is a snapshot exactly? MySQL keeps the totality of data for a given database in a single directory. For example, if the data directory is `/var/lib/mysql` and the database in question is `test`, then all tables and indices are stored in files in the directory `/var/lib/mysql/test`. For our purposes, a snapshot is a tarball of one of these database-holding directories.

Another good way to check is to give the command

```
show slave status;
```

to the mysql program. In particular, "Slave_Running" should be marked "Yes".

There is one important thing to note about this mechanism. The replication of the databases is meant as a one-way street. All changes to constants should be made on the central server, `clasdb.jlab.org`, the "master". The replication mechanism will then distribute them to the mirror sites, the "slaves." There is no completely straightforward way to propagate changes made on slaves back to the main server. Even if there were, the possible confusion due to parallel changes would make it a bad idea. In fact, if you make changes to your replicated databases, replication may hang due to duplicate values of "primary keys" when independent changes are made to the same table on the master and on the slave. In other words, treat a replicated database as read-only.

It is not absolutely necessary to use replication to get a local version of the database. You can simply upload one of the backups (see Section 7.3). Replication has the advantage of continuously and automatically updating your local version.

7.3 Backups of the CalDB

Every night a backup of the `calib` database on `clasdb.jlab.org` is done. The technique used is to run `mysqldump` on the entire `calib` database. This saves all of the contents of all of the tables in `calib` in an ascii text file that has the form of MySQL commands. If a user inputs this file to the `mysql` command line tool on standard input, the entire contents of `calib` will be reconstituted on the chosen server. See the MySQL manual for details.

These backups can be used for a variety of purposes:

- In case of a disk failure we can reconstitute the database from one of these backups. The JLab Computer Center is also doing a nightly backup of the group disks, so our backup is redundant. On the other hand having direct access to the backups may be convenient.

- As an alternative to replication (see Section 7.2) MySQL servers other than `clasdb.jlab.org` can use these backups to install local versions of the `calib` database, either as a one-time test, an occasional refresh, or as part of a periodic update procedure.

- The backups provide a crude time history. Detailed time histories are built into the database, but in the case of major user/officer/manager/software error, it may be convenient to have a set of copies, marked by date, to study the problem and possibly fall back on.

The backup runs at midnight every night. The files are stored on the CLAS work disk in `/work/clas/disk3/claslib/calib_backup`. The files have names like `calib_dump_2001_08_22.sql.gz` (in this example the dump was done on August 22, 2001). There is a symbolic link, `calib_dump.sql.gz`, that points to the latest version.

Once a week we write the latest backup to the silo. The silo directory is `/mss/clas/caldb`. You can retrieve any backup from tape via the JLab Computer Center's `jget` command.

8 Online Constants: Updating the RUN_CONTROL System

The RUN_CONTROL system[h] contains quantities that are generated during data-taking, such as beam energy, magnet currents and critical scaler values. Among other places, the online system stores all of them and more in an INGRES database. There is a cron job that runs every six hours that checks for new entries in the online INGRES database and transfers them to the CalDB.[i] The script is `$CLAS_TOOLS/caldb/online_update.pl` and is run as user `clascron` on `clon10.jlab.org`.

Appendix A CALDB C-API functions

A set of C subroutines (C-API) have been written to perform read and write functions on the database. The C-API files are in the CLAS CVS repository under packages/caldb/C.

ConnectToServer()

function prototype:
MYSQL *ConnectToServer(char *host,char *dbname,char *user,char *user_password)

Description

Attempts to establish a connection to a MySQL database engine running on **host**(by default clasdb) and access the database **dbname** (by default calib) . The **password** parameter contains the password for the **user** .

return value

MySQL handler (if no error), needed by all other functions to access the dbase.

NULL in case of error.

example mysql=ConnectToServer(host,dbasename,user,password);

DisconnectFromServer()

function prototype: void DisconnectFromServer(MYSQL *conn)

Description

Closes a previously opened connection.

[h] Our entire approach to handling these online constants should be reviewed. The CalDB, as currently configured is designed to deal with constants that apply to a range of runs. If there is a one-to-one correspondence between run number and constant, a different structure is appropriate. We also need to deal with the question of authority: which is right, online or offline database?

[i] Although in principle, the online programs could update the CalDB in real-time, that method requires an extra-counting-room network connection that would have the potential to interfere with data-taking.

`WriteConstantSet()`

function prototype:
int WriteConstantSet(MYSQL *conn, char *systemname, char *subsystemname, char *itemname, int minrunsource, int maxrunsource, char *calib_comment, char *value, itemvalue *itemvalueid)

Description Writes a set of calibration constants in the calib database.

input:
1. systemName, subsystemName, itemName to fill a valueID table systemName_subsystemName_itemName (EC_GAIN_outer_u).
2. value array of N constants for corresponding item (calibration constants "v_0001...v_0NNN")
3. author
4. time
5. the range of runs used in extraction (minRunSource,maxRunSource)
6. comment (calib_comment) including some details on the procedure (version of software ....)
7. mysql connection handler.

return values

itemValueId a variable containing the number of entry in systemName_subsystemName_itemName table.

istat a status of the transaction (0 if OK)

example:

istat=WriteConstantSet(mysql, systemName, subsystemName, itemName, minRun, maxRun, calib_comment, value, &itemvaluid)

`LinkConstantSet()`

function prototype:
int LinkConstantSet(MYSQL *conn, char *systemname, char *subsystemname, char *itemname, char *RunIndexTable, int minrun, int maxrun, char *runindex_comment, itemvalue itemvalueid, commentstruc runcomm, itemvalue *runindexid)

Description Link a set of calibration constants fro value table to RunIndex table. input:

1.systemname,subsystemname,itemname (to get the tablename=EC_GAIN_outer_u and the itemId=283 (see item table))

2.RunIndexTable Name="RunIndex" (also a working copy of RunIndex table could be used)

3.minrun, maxrun minimum and maximum run# validity range for this set of constants (could be different from minrunsource,maxrunsource)

4.runindex_comment,author comment and author filled in a runcomm structure(see later) (could be different from valuId table values)

5.itemvalueid the number of entry in the valueId table (EC_GAIN_outer_u).

6.mysql connection handler

return values

runindexid ID# in RunIndex for this entry. structure commentstruc int minRunSource; int maxRunSource; char *author; char *time; char *comment; (defined in the calib_connect.h) gives details on constants for record=itemvalueid of systemName_subsystemName_itemName table.

istat a status of the transaction (0 if OK)

example:

istat=LinkConstantSet(mysql, systemName, subsystemName, itemName, RunIndexTable, minRun, maxRun, calib_comment, itemvalueid, runcomm, &runindexid)

ReadConstants()

function prototype:

int ReadConstants(MYSQL *conn, char *systemname, char *subsystemname, char *itemname, char *RunIndexTable, int runno, char *date, itemvalue *itemvaluid, commentstruc *runcomm, valuestruc *tlvalue)

Description Reads a set of calibration constants from a valueId table

input:

1. systemname, subsystemname, itemname (to get the tablename=EC_GAIN_outer_u and the itemId=283 (see item table))

2.RunIndexTable Name="RunIndex" (also a working copy of RunIndex table could be used)

3.runno run#

4.date (consider inputs before validity date and take the latest set)

return values

itemvalueid - the number of entry in the valueId table

structure commentstruc int minRunSource; int maxRunSource; char *author; char *time; char *comment;

structure valuestruc int length; char *type; char **item; (defined in the calib_connect.h) gives length,type and value string with constants ("v_0001...v_0NNN")

istat a status of the transaction (0 if OK)

example:

istat=ReadConstants(mysql, systemName, subsystemName, itemName, RunIndexTable, runno, date, &itemvaluid, &runcomm, &tlvalue);

WriteandLinkConstantSet()

function prototype:

int WriteAndLinkConstantSet(MYSQL *conn, char *systemname, char *subsystemname, char *itemname, int minrunsource, int maxrunsource, char *calib_comment, char *value, itemvalue *itemvalueid, char *RunIndexTable, int minrun, int maxrun,
char *runindex_comment, itemvalue *runindexid)

Description Write and link a set of calibration constants in the calib database. All input and output variables are the same as for WriteConstantSet() and LinkConstantSet() functions.

caldb_getItemId()

function prototype:
int caldb_getItemId(MYSQL *conn, char *systemname,char *subsystemname,char *itemname,int *length,char **type, char **itemid)

Description
Return the item number, variable type and length.

caldb_getItemValueId()

function prototype:
int caldb_getItemValueId(MYSQL *conn, char *tablename, itemvalue *itemvalueid ,commentstruc *runcomm)

Description
Return the id of entry from value table.

References

[1] L. Dennis, A. Freyberger, G. Gavalian, M. Holtrop, M. Ito, G. Riccardi, R. Suleiman and D. Weygand, CLAS Calibration Database Specification, CLAS-NOTE 2000-008, http://clasweb.jlab.org/caldb/cal_db_spec/.

[2] http://www.mysql.com

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

$Id: caldb.tex,v 2.112 2003/05/28 20:39:43 marki Exp $