A Program to Estimate Resolution for Charged Particles in GlueX

GlueX-doc-1015-v2

Mark M. Ito, Dmitry A. Romanov
Thomas Jefferson National Accelerator Facility
12000 Jefferson Avenue
Newport News, VA 23606

June 15, 2012

Abstract

This note contains a documentation of the CCDB package. A package for storing and managing calibration constants database.

Contents

1 Introduction

Calibration Constants Database (CCDB) aims the next goals:

- Storing calibration constants.
- Managing calibrations.
- API for JANA, plain C++, Python.
- Additional Logging, import, export data.

CCDB stores data as tables with columns and rows. As a data storage CCDB supports:

- **Naming**. Each table is identified by path-name;
- **Versioning**. Each table may has many versions of data;

- **Branching**. So called "variations" allows to use branches of data;

As a management tool and as a data provider CCDB allows:

- C++ User API. Allows an easy access to CCDB data from C++.
- JANA API. An integration to JANA framework.
- Python API. Allows accessing and managing CCDB from python language.
- Command line tools. Tools to manage CCDB data from the shell.
- Web interface.

2 Basic concepts

CCDB basic usage concepts and ccdb console tool for managing CCDB contents

3 CCDB command line tutorial

This section is a tutorial of using CCDB command line tools.

3.1 Getting started

CCDB provides console tool for introspection and managing constants database. One can run it by typing 'ccdb' command. Command line tools could be used as interactive shell or as a shell command.

Usage from command line is:

```
ccdb <ccdb arguents> command <command arguments>
```

Usage as interactive shell:

```
ccdb <ccdb arguments> -i
> command1
> command2
> ...
> q
```

Lets see a real example. Example 1. Command line:

```
           (1)                              (2)  (3)
ccdb -c "mysql://john@localhost:999" ls /TOF/params
```

1. -c mysql://john@localhost - sets the ccdb connection string. If no -c flag is given, ccdb will try CCDB_CONNECTION environment variable, then use default connection string. The connection strings are described in the connections chapter
2. ls - is a ccdb command which returns a list of directories and tables that belongs to directory '/TOF/-params'
3. /TOF/params - is the argument of ls command. Like a posix shell ls.

Example 2. Interactive:

```
ccdbcmd -i -c "mysql://john@localhost:999"              (1)
> ls /TOF/params                                       (2)
> help                                                 (3)
> cd /TOF                                              (4)
> cd params
> ls
> pwd                                                  (5)
> q                                                    (6)
```

1. flag '-i' will start ccdb in interactive mode.
2. 'ls /TOF/params' - the result of the is exactly the same as in Example 1. One stays in the interactive shell after the execution.
3. 'help' command provides list of commands and how to use each of them
4. executing next commands will reproduce Example 1 step by step.
5. The same as in posix shell, ccdb interactive mode have the current working directory, with relative and absolute pathes. pwd command will show the current working directory.
6. to exit interactive mode enter 'q', 'quit' or press ctrl+D

Since ccdb objects have /name/paths and many other things that looks like POSIX file system, the commands are very posix-shell-like.

3.2 Connection

CCDB uses so called "connection strings" to specify a data source. The generic format of a connection string is:

```
<protocol>://<datasource specified string>
```

MySQL connection string:

```
mysql://<username>:<password>@<server_address>:<port> <database>
```

One may omit any part except "mysql://" and "¡server_address¿". The default values will be used.

CCDB MySQL connection defaults:

- username - ccdb_user
- password - no password
- port - default MySQL port (now is 3306)
- database - ccdb

Here is the order of how ccdb gets the connection string:

1. The default connection string is **"mysql://ccdb_user@localhost ccdb"**
2. if **CCDB_CONNECTION** environment variable is set it is used overwriting the default connection string
3. if -c or –connection flag is given in command promt it is used overwriting all other.

Example 3. Connection string 1:

```
"mysql://john@localhost:999"
```

- MySQL server on 'localhost' using port 999
- user is 'john' with no password

- the database is 'ccdb' by default

Example 4. Simple connection string:

```
"mysql://localhost"
```

- MySQL server on localhost using port 3306 (default)
- user is 'ccdb_user' with no password (default)
- the database is 'ccdb' (default)

Example 5. Full connection string:

```
"mysql://smith:hHjD83f@192.168.1.3:4444 ccdb_database"
```

It tells ccdbcmd to connect to:

- MySQL server on '192.168.1.3' using port 4444
- user is 'smith' with password 'hHjD83f'
- the database is 'ccdb_database'

3.3 Help system

The ccdb is designed to be a self descriptive. By using 'help' 'usage' and 'example' commands one could get all the commands and how to use them.

By using 'howto' command one could get tutorials for typical situations.

3.4 Commands

3.4.1 Commands consistency

Command keys are consistent. This means that some flags and argument formats are the same across all commands. There are unified flags to identify objects for all commands:

- **-v** - Variation
- **-t** - Data table
- **-r** - Run or run-range
- **-d** - Directory

For example 'info' command may be executed against directory, table or variation. Example 6. Info command:

```
    ccdb -i
    > info -v default                                    (1)
  > info -r all                                          (2)
  > info -d /TOF                                         (3)
  > info -t /TOF/params                                  (4)
  > info /TOF/params                                     (5)
```

1. Get information about "default" variation 2. Get information about "all" runrange. "all" runrange is [0, infinite_run] 3. Get information about "/TOF" directory. 4. Get information about "/TOF/params" type table 5. By default '*info*' treat non flag argument as a name of a table.

info	Info	Prints extended information about an object
vers	Versions	Show versions of data for the specified table
run	CurrentRun	Gets or sets current working run
dump	Dump	Dumps data table to a file
show	Show	Shows type table data
mkdir	MakeDirectory	Create directory
pwd	PrintWorkDir	Prints working directory
cd	ChangeDir	Change current directory
add	AddData	Add data constants
mktbl	MakeTable	Create constants type table
cat	Cat	Show assignment data by ID
ls	List	List objects in a given directory

Table 1: List of ccdb commands

3.4.2 Commands overview

This table is printed if one executes "ccdb help"

Assuming that user is in interactive mode, one may categorize the commands:

To navigate directories pwd - prints curent directory cd - switch to specified directory ls - list objects in the directory (wildcards are allowed) mkdir - creates directory

Example 7. Directory commands overview:

```
    > pwd
    /
> cd /TOF
> ls
    table1  table2
> mkdir constants
> ls con*
    constants
```

Get information about objects

- **info** - gets information about objects (use -v -r -d flags), see example 6.
- **vers** - gets all versions of the table
- **cat** - displays values
- **dump** - same as cat but dumps files to disk
- **logs** - see logs information

Manage objects

- **mkdir** - creates directory
- **mktbl** - creates data table
- **add** - adds data from text file to table (variation and runranges are created automatically by add command)

3.5 Data requests

There are two problems related to data access which CCDB tries to sove: 1. Getting constants should be as easy as to say "Give this constants for June 2011" 2. There should be one way to give an unique key for

every set of data.

The first thing that comes to mind when one hears "unique key" might be using incremental indexes. But CCDB is database independent. Indexes which are good to use with databases become uneasy for standalone ACSII files. Moreover, moving indexes from one database to another might be a problem. And, last but not least, indexes are good for machines. If an operator has an index 1114211 it tells nothing to him and could be easily mistaken with 1142111 which belong to absolutely another data set.

CCDB uses so called "Requests" to solve both of the problems. The full form of the request is an "unique composite key" for the particular data values.

Full form of the request is

```
</path/to/data>:<run>:<variation>:<time>
```

To get the data user can specify only a part of the request. The minimal request to get the data is just /path/to/data One may omit any part of the request except for name-path.

Lets look at examples:

- /path/to/data - just path to data, no run, variation or timestamp is specified
- /path/to/data::mc - no run specified, variation is "mc", no date is specified
- /path/to/data:::2029 - only the path and the date(year) are specified

As shown in the examples above, to specify a path and a variation but to use default run one skips the run number and leave its place like "::":

So

```
                +-- variation
                |
                |
/path/to/data::mc
             ^
             |
             +-- place where run number should be
```

And the request '/path/to/data:::2029' means that we specify a path and a date but leave a run number and a variation to be set by default. What does word 'default' means? We will discuss it in "DEFAULT VALUES" chapter.

The time is parsed as:

YYYY:MM:DD-hh:mm:ss

Any non digit character may be used as separator instead of ':' and '-'

so all these data lines are the same

```
2029/06/17-22:03:05
2029-06-17-22-03-05
2029/06/17:22/03/05
2029a06b17c22d03e05
```

One can omit any part of the time string starting from the right, this the latest date for this part will be returned.

Examples:

"2011" - (this means the year 2011), it will be interpreted as 2011/12/31-23:59:59 timestamp so the latest constants for year 2011 will be returned.

"2012/05/21" - it will be interpreted as 2012/05/21-23:59:59 meaning to be the latest constants for 21 May 2012

CCDB searches the closest constants before or equal to timestamp provided.

3.6 Default values and priorities

There are two general cases of using the requests: 1. In physics software to read out constants 2. When one manages constants (with ccdb console, python or other).

In the first case most probably the software will provide the run number being processed. Also, most probably, the software should allow to set the variation to prefer for the analysis.

So, CCDB defaults and priorities (1 - highest)

Run number: 1. Run number specified in a request (if you use "/path/to/data:100" request, constants for run 100 will be returned dependless of the run being processed) 2. Software set global default run number. (if 10200 run is being processed and you use "/path/to/data" data for run # 10200 will be returned) 3. 0 - (means run number 0).

Variation: 1. Variation specified in a request (if you use "/path/to/data::mc" request, constants for variation mc will be used) 2. Global preferred variation set by software. (...) 3. the "default" variation.

Timestamp: 1. Request specified time will be used 2. Current time

When one uses ccdb console tool in interactive mode, one can set the default run number by running 'run' command

Example:

```
> run 100
> cat /path/to/data     # all commands will get constants for run 100
> run                   # you can check what run is set by default
100
```

C++ API section will overview how to set defult run

References