CCDB File format

CCDB File format - is a file system convention of storing the CCDB constants locally.

Abstract. CCDB File format is aimed to hold each assignment data set in separate file. The simplest form of such file is just a ASCII text with columns and rows of values in it. But also such files may contain comments, data type info and some meta data. It is possible to load just text files with columns and rows, JANA calibration dumps or full featured CCDB files as CCDB constant sets with the same parser API.

File format description.

Brief. CCDB Files are text files with UTF-8 (?) encoding. Constants are stored as columns and rows. Data containing a space characters (strings) must be taken into quotes ‘ “ ‘. All text after # is a comment. #& - comment indicates that the line holds column names. #meta comment indicates meta data key-value pairs that holds information such as variation, run range etc. according to “DB design”. If data have only one row it also may be stored as column of data fields and columns of column_names (see examples below)

General file format:

- CCDB files are text files stored in UTF-8 (?) encoding.
- Space characters* in the beginning of a line are not taken into account.
- Empty strings or strings containing only space characters are not taken into account.

* “Space characters” are: " ", "\n", "\t", "\v", "\r", "\f". So space, two spaces, space-tab-space, etc. - all combinations of above symbols are interpreted as “space characters”.

Data layout:

- Data is stored as columns and rows.
- Columns are separated by space characters (any combination of them). The combination of space characters is irrelevant from row to row, only columns order(count) matters.
- Rows are lines that contain columns.

Example.

Left and right examples are equivalent. It is 2 columns by 3 rows layout.

value1 value2 value3 value4 value5 value6	value1 value2 value3 value4 value5 value6

Strings:

- string fields* should be taken into quotes ‘ “ ‘ (“string field” means data field which contains or just may contain space characters, like if user stores text in such fields)
- if there is no spaces inside a data field it is irrelevant ether it taken into “ quotes or not. See example below.

- quotes in user strings must be shielded as ‘ \” ’

There is no possibility to store multi-line text in “multi-lines”. Thus it suggested for user to replace “new line” symbol to store multi-line texts.
- “new line” symbol (like \n in linux) must be encoded by user to be stored in files. And decoded back by user after file readout.
- Recommended encode symbol for “new line” is &nl; since this symbol is used to encode new lines by CCDB C++ API.

Example.

record in file	readout value	comments
3.14	3.14	
“Jhon Smith”	Jhon Smith	
Jhon Smith		will be interpreted as two columns ‘Jhon’ and ‘Smith’ probably causing an error
“3.14”	3.14	value is same as above
“3.14 “	‘3.14 ‘	value with two spaces after!
“my \”quotes\””	my “quotes”	example of shielded quotes

Comments:

- character indicates one line comment. All data from this character to the end of the line is interpreted as comments or meta data (see below)

Example.

it is my comment

Metas:

Combination of #meta indicates that meta-data information is stored in this comment line. Meta data keeps information of ccdb source, variation, run ranges etc. What is "#meta"? On the on hand it is just a comment (probably automatically generated), that could help user to figure out the origins of this file and other information. The difference of #meta and simple # comment comes out when file is transferred to DB: simple comments will be collected together and saved as record comments. #meta comments will not be saved as record comments as record comments, but might be used to provide some data. See conversion to DB chapter.

- Only one #meta is allowed per one file line.
- All symbols after #meta (except any first spaces and tabs) and before first ':' are interpreted as "key"
- All symbols after first ':' are interpreted as "value"
- Thus meta data is stored in form of :

#meta key : value

- Reserved (used by ccdb file process) meta keys

run range, event range, variation, path, time stamp... something else

Example.

#meta variation : default

#meta file specification url : http://wiki.gluex.org

Layouts and column names:

- According to CCDB specification, columns may have names. Text containing column names starts with #& combination.
- According to CCDB specification, column names may not contain space

characters.
-There are two possible layouts of data and column names.

- names and data values may be placed as:

  Example.

```
#& column1_name   column2_name …   columnN_name
   <column 1>     <column 2>   …   <column N>
```

- If data have only one row, it may be placed as column of data and column of names in form of:

  Example.

```
<value 1> #& name1
<value 2> #& name2
...
<value N> #& nameN
```

Order andcontent layout:

Recommended order of content layout in files are:

1) all meta data separated by emty line

2) all lines of comment

3) column names if it is multy rows data set

4) all data

Example.

```
#meta ….
#meta …. all meta datas
#meta ….
# …
# ... all lines of comments
# …
#& names of columns
< data columns and rows. >
```

Handling inconsistencies and errors while readout

parse time:

string as “string”.

- **No ending quote** . If no ending “ is found, string value will be taken until the end of line.
- **Comment inside a string.** Comment symbol inside the line is ignored. So if you have a record in the file “info #4” it will be read just as “info #4” string
- **Sticked string.** In case of there is no spaces between symbols and an quotes, all will be merged as one string. I.e.:
  Jhon” Smith” will be parsed as one value: “Jhon Smith”
  Jhon“ ”Smith will be parsed as one value: “Jhon Smith”
  but be carefull(!) not to forget to do a spaces between columns
  5.14”Smith” will be parsed as one value “5.14Smith” that probably will lead to errors if it were two different columns
- If data contains string fields they are taken into “...” characters. All “ inside string should be saved by \” symbol. All words and symbols inside “...” will be interpreted as string entity.

Examples of files:

1) Simplest CCDB file may contain just columns and rows of data:

```
1    2    3
4    5    6
```

2) Like JANA calibration dump:

… will write full example in future

File system layout.

To allow easy using of JANA and C++ API, directories and file names, in witch the files are placed, arranged so that file paths (relative to some user defined $PARENT_DIR) corresponds to CCDB constants type table paths.

Directories:

- Directories structure in file system corresponds to CCDB directory structure of constants in database.
- “Parent directory” is a file system directory where all nested files and directories are located. Speaking of “Parent directory” as some path in file system we will reference it as $PARENT_DIR here and below.

Files:

- Each constants set is stored in one file.
- File name begins with corresponding constants type table name in database and ends with extension.
- File extenstion is recomended to be .ccdb.

Example.

If CCDB constant have a name:

/CDC/alingments/adc

it should be stored in file system as

$PARENT_DIR/CDC/alingments/adc.ccdb

Parsing API.

API for parsing of files consist of Parser and FileDOM class .

Parser:

Parser class have a static method :

```
FileDOM*  Parse(const string& fileName);
```

Which returns not NULL pointer to FileDOM object if parsing was done with no errors.

FileDOM:

FileDOM (DOM- document object model) presents contents of CCDB file by its properties. It holds information as:

```
map<string, string> meta;    // All meta data as key value pairs
string comments;             //all lines of comments
vector<string> columnNames; //names of columns
vector<vector<string>> data;//all data tokens as strings
```

Importing files as CCDB database.

This section describes the process of adding files as assignments. Import

from files to CCDB database. This is basics. More advanced things should be implemented in the future.

- All type table information is provided by user.
- All comments except #& and #meta are saved as assignment comments.
- #& and #meta ignored for this iteration of development.
- inconsistency of rows and columns number of file and database is treated as error.

ITERATION 2

To be done in future development

Importing files as CCDB database.

This section describes the process of adding files as assignments. Import from files to CCDB database

Priorities.

To add assignment to database the user should specify at least run range and variation. The both may be found in file name or in #meta inside the file. Thus to import file to db one needs to set priority if this meta data differs.

When using CCDB to get constants from files, file names have higher priority.

When importing files to database, meta data stored in files have higher priority. (It is ugly! Needs discussion)

When importing to DB.

1) First priority (certainly) is user will.
2) Meta data stored in files (as #meta)
3) Meta data retrieved from file names

In other words.If user sets run range or variation, assignment will be added with them independently of meta data in file or file name. If user didn't specified it #meta in files wold have more priority under variation or run range from file name.

Default values.

1) Run ranges. If run range is unspecified, 1 to infinity run range will be used.

2) Variation. If no variation is specified in #meta it should be specified by user.

1) If there is inconsistencies in column names. Column names from DB will be used, user will be warned.

2) Unknown metas will be ignored. User will be warned.

3) If there is inconsistencies in column number or rows number. Error will be raised.

4) Inconsistencies in values format (means that in i.e. if value is set as double but cannot be read as double). Error will be raised.

5) Unknown characters and artifacts (in data rows or before # comments) will lead to 3) and 4) errors.

6) what else?

File system layout.

- File names may include meta data. Full file name specification includes:
  <name>.v<variation>.r<run_range>.d<date_time>.ccdb
  where:
  <name> - name of constants type table in db
  <variation> - assignment variation
  <run_tange> in form of min-max - minimum and maximum run range wounds

<date_time> - in form of yyyy-mm-ss_hh-mm-ss date and time according to time stamp of assignment in db.

if variation is not set, file treated as “default” variation

if run range is not set, file treated as 0 - inf

if date_time is not set, file treated as most old file.

For more information see “Priorities and error situations”.

- It is recommended for files to have creation time corresponded to assignment timestamp, thus allowing users to order file with assignments by creation time.

It is recommended for files exported from DBto have creation time corresponded to assignment timestamp, thus allowing users to order file with assignments by creation time.

USING JANA and CCDB C++ user API

If all files is holded in each directory it is easy to use it by JANA or CCDB C++ API.

Default values.

1) Run ranges. If run range is unspecified, 1 to infinity run range will be used.

2) Variation. If no variation is specified, file will processed as “default” variation

ITERATION 3

To be done