Table of Contents

LHCb Conditions Database How-To (for COOL)

This is an old version of the documentation. The latest version is CondDBHowTo.

Table of Contents

Overview

The aim of this page is to collect examples and instructions to allow the users of the LHCb Conditions Database (CondDB) to accomplish common CondDB related tasks.

The tasks are grouped in two areas:

- test and development
- production

Test and Development Tasks

Set the Content of a Condition Bypassing the CondDB

Problem

We need to know how the result of an application would change if we replace one or few conditions in the database with different static (not changing between two events) values.

Example

The position of the Velo halves are fixed in the "close" position. To reconstruct using the "open Velo" configuration, we need to modify the `VeloLeft` and `VeloRight` alignment conditions in order to move the right half of -30mm on X and the left one of 30mm on X.

Solution

We need to add to the python option file of our application the following lines

```
from Configurables import UpdateManagerSvc
UpdateManagerSvc().ConditionsOverride += [
  "Conditions/Alignment/Velo/VeloRight :=  double_v dPosXYZ = -30 0 0;",
  "Conditions/Alignment/Velo/VeloLeft  :=  double_v dPosXYZ =  30 0 0;"
]
```

Explanation

The `UpdateManagerSvc` is the manager of the used conditions. When somebody needs the overridden condition, the `UpdateManagerSvc` will replace it with the one provided via job options.

To define a condition in the job options file, you have to add to the option `UpdateManagerSvc.ConditionsOverride` (array of strings) a string containing the path to the condition in the transient store (e.g. `Conditions/Alignment/Velo/VeloRight`) followed by `':='` and by a `';'` separated list of parameters. Each parameter in the form of `type = value` where `type` is any of `int`, `double`, `string`, `int_v`, `double_v` or `string_v` (the `'_v'` means *vector of*) and where `value` is the *value* you want to use (a space-separated list in case of vectors).
Note: For LHCb < v22r8, if the condition contains more than on parameter, you also have to copy all the

values you do not want to override.

Create and Use a Local Copy of the CondDB

Problem

If we want to test or validate a dynamic set of conditions (that change between events), we need to have a local copy of the conditions database.

Solution

Here I shall show how to create a copy of the database containing the conditions (former `XmlConditions`). To create a copy of the Detector Description Database (former `XmlDDDB`), replace in the following recipes `LHCBCOND` with `DDDB`, `LHCBCOND` or `ONLINE`.

Create the Copy

There are two equivalent ways to create a copy of the database, either via the CondDB Browser or via python commands that can be put in a script.

Using the Graphical User Interface

- First you have to start the CondDB Browser:

```
# lb-run LHCb bash --norc
# CondDBBrowser LHCBCOND &
```

- Select from the menu **DataBase** the action **Create slice**.
- Chose a name for the SQLite file you want to create and write it in the filed **SQLite file name:** (you can also use the **...** button that will open a file selection dialog). I will chose the file **myLHCBCOND.db** in my home directory.
- Chose a string to use as **Database name:** (it can contain up to 8 upper case characters or "_", it is needed because one SQLite file may contain more independent databases). I suggest to keep the original one: **LHCBCOND**.
- In the group **Selection Object Creation**, you can chose which nodes (foldersets or folders), for which period (since, until) and for which tag to copy. A copy of a whole version (namely **DC06**) of the database is achieved by using **Node: /**, **Since: 1970-01-01 01:00:00**, **Until: +inf** and selecting the tag in the **Tag Name:** list (**DC06**). If you want, you can select multiple tags.
- In the group **Selection Objects List**, click on **Add** to schedule the selection we prepared for the copy. You can add many selections to the list.
- Now you can click on **OK** to create the new database. You can also add the selected data to an already existing database using the button **Append**.

The database is ready.

Using the Python commands

- Open a python shell in the correct environment (LHCb >= v24r1):

```
# lb-run LHCb bash --norc
# CondDBAdmin_MakeSnapshot.py -T DC06 LHCBCOND sqlite_file:$HOME/myLHCBCOND.db/LHCBCOND
```

The database is ready.

Use the Copy in Your Application

There are two possibilities:

1. you created a local copy of the whole LHCBCOND database
2. you only copied a few nodes

Complete Copy

To use the local full copy of the database in your application, you have to override the default job options. Add to your options a file like:

```
from Gaudi.Configuration import *
from Configurables import CondDB
cdb = CondDB()
cdb.PartitionConnectionString["LHCBCOND"] = "sqlite_file:$HOME/myLHCBCOND.db/LHCBCOND"
```

If you need to use a tag different from the one defined in the standard options (e.g. `"my_tag"`), add also the line:

```
cdb.Tags["LHCBCOND"] = "my_tag"
```

or

```
cdb.Tags["LHCBCOND"] = ""
```

if you want to use the HEAD version of your local copy. Note that a complete copy can be used as a partial copy with everything inside

Partial Copy

This case is much simpler:

```
from Gaudi.Configuration import *
from Configurables import CondDB, CondDBAccessSvc
CondDB(). addLayer(
    CondDBAccessSvc("myCond",
        ConnectionString = "sqlite_file:$HOME/myLHCBCOND.db/LHCBCOND",
        DefaultTAG = "my_tag"))
```

Explanation

TODO

Generate XML Files from the Content of the CondDB

Problem

The CondDB is not particularly easy to browse and modify. In some cases, it may be useful to edit many XML files at the same time and, possibly, with editors that are validating the XML.

Solution

Since `Tools/CondDBUI v2r4`, a tool is available to extract a usable snapshot of the conditions database as XML files: `dump_db_to_files.py`. The usage is simple:

```
# lb-run LHCb bash --norc
# dump_db_to_files.py -c <connection string> -T <tag> -t <time> -d <destination directory>
```

A practical example to extract the current version of the detector description is:

```
# lb-run LHCb bash --norc
# set tag = "DC06-20080407"
# dump_db_to_files.py -c sqlite_file:$SQLITEDBPATH/DDDB.db/DDDB -T $tag  -t `date +"%s000000000"`
# dump_db_to_files.py -c sqlite_file:$SQLITEDBPATH/LHCBCOND.db/LHCBCOND -T $tag  -t `date +"%s000
```

(Note: you need to use a tag compatible with the version of the software you are using).
To use these XML files, you need to add to the default options (in Python):

```
from Gaudi.Configuration import *
from Configurables import DDDBConf
DDDBConf(DbRoot = "/tmp/myDDDB/lhcb.xml")
```

Because of some implementation details, the ONLINE partition cannot be used outside a database, so you need to use an hybrid configuration where some data is taken from files and others are taken from the DB. To achieve that, you have to modify the file /tmp/myDDDB/Conditions/MainCatalog.xml replacing the line

```
<catalogref href="Online"/>
```

with

```
<catalogref href="conddb:/Conditions/Online"/>
```

Another practical example, to extract the current version of the particle properties table:

```
# dump_db_to_files.py -c sqlite_file:$SQLITEDBPATH/DDDB.db/DDDB -s /param/ParticleTable.txt
```

Use a Local Tag of the database

When changes in the XML need to be paired with changes in the C++ code, it is important to be able to get part of the content of the DB from one tag and part from another.

Let's assume we want to use the global tag `"head-20080225"` as a base and the tag `"muon-20080407"` (only in `DDDB`) to test some new Muon code.

The options to do that are (python options):

```
from Gaudi.Configuration import *
from Configurables import CondDB, LHCbApp

# Define the new tags
base_tag = "head-20080225"
muon_tag = "muon-20080407"

# Set the default tags
LHCbApp(DDDBtag = base_tag, CondDBtag = base_tag)

# Get the instance of the CondDB configurable
CondDB().LocalTags["DDDB"] = [ muon_tag ]
```

Override part of a global tag using another global tag

Sometimes it is useful to compare the the effect of the changes in some conditions between two global tags.

If the latest global tag in `LHCBCOND` is `head-20100119` and we want to compare the effect of the changes in the Rich conditions with respect to the tag `head-20091211`, we have essentially two possibilities.

Reproduce the global tag from the local tags

From the CondDB Release Notes, we can see that (in `LHCBCOND`) the global tag `head-20100119` is based on the global tag `head-20091211` plus the local tags `rich-20100119`, `prs-20100117`, `align-20100115`, `rich-20100113`. To check what would have been the effect if the new global tag was not including the Rich changes, use these options:

```
from Gaudi.Configuration import *
from Configurables import CondDB, LHCbApp

# Define the tags
base_tag = "head-20091211" # base tag for head-20100119
local_tags = [ "prs-20100117", "align-20100115" ]

# Set the default tags
LHCbApp(CondDBtag = base_tag)

# Get the instance of the CondDB configurable
CondDB().LocalTags["LHCBCOND"] = local_tags
```

Use a sub-tree from the old tag

If removing a local tag from the global tag is not what is needed, may be because it will hide too much, you can use the old local tag for a defined subtree.

Let's say that we want to ignore the changes to Rich2, but not those for Rich1 between the two global tags. To do that, we need a CondDBAccessSvc configured to access `LHCBCOND` with the old tag and use it only to access the subtree "/Conditions/Rich2":

```
from Gaudi.Configuration import *
from Configurables import CondDB, LHCbApp, CondDBAccessSvc

# Define the tags
base_tag = "head-20091219"
old_tag = "head-20091211"
LHCBCOND_old = CondDBAccessSvc("LHCBCOND_old")
LHCBCOND_old.ConnectionString = "sqlite_file:$SQLITEDBPATH/LHCBCOND.db/LHCBCOND"
# or "CondDB/LHCBCOND" if using Oracle
LHCBCOND_old.DefaultTAG = "head-20091211"

# Set the default tags
LHCbApp(CondDBtag = base_tag)

# Get the instance of the CondDB configurable
CondDB().addAlternative(LHCBCOND_old, path = "/Conditions/Rich2")
```

Note: if this use case is popular, the configuration can be simplified by instrumenting the `CondDB` configurable.

Use the Oracle database

To use the Oracle database you need some form of authentication. Currently, we use grid proxy certificates.

On lxplus, you have to call `lhcb-proxy-init` end enter your certificate password when prompted. Then

```
# lb-run LHCb bash --norc --use-grid
# CondDBBrowser.py CondDB/LHCBCOND
```

where `CondDB/LHCBCOND` is an example of connection string. All the valid connection strings for Oracle are:

- `CondDB/DDDB`
- `CondDB/LHCBCOND`
- `CondDB/SIMCOND`
- `CondDBOnline/ONLINE`

On plus (PIT), you do not need authentication because it is a trusted environment, so just call the wrapper script:

```
/group/online/condb_viewer/CondDBBrowser.sh CondDBOnline/ONLINE
```

which embeds a call to `lb-run`. You can use the same connection strings mentioned for lxplus, plus `CondDBPrivate/PRIVATE`.

Override the heartbeat and the run-stamp conditions (e.g. when using Oracle in the online farm)

In order to ensure that replicas of the ONLINE partition of the database are sufficiently up to date two special conditions are inserted in the database at specific times:

- the "heartbeat" condition is inserted periodically during the physics runs and at their end
- the "run stamp" condition is added for the IOVs of the runs for which alignment/calibration have been produced (before HLT2 processing)

The framework blocks access to the CondDB if one of the two special conditions is not valid for the current event time with a message like

```
ONLINE              ERROR Database not up-to-date. Latest known update is at 1269840266.0, event
```

or

```
RunStampCheck       ERROR Database not up-to-date. No valid data for run at 2015-06-10 12:00:00.0
```

Obvioulsly this check does not make sense in some special cases (e.g. when preparing alignments, or when using the Oracle DB in the online farm). In these cases the above checks can be disabled adding the following option:

```
CondDB().IgnoreHeartBeat = True
CondDB().EnableRunStampCheck = False
```

This implementation of this feature is discussed in Savannah task 13270 and in LHCBPS-1421.

Production Tasks

Request changes to the content of the official CondDB

To request changes to the content of the official CondDB, use the CondDB JIRA tracker. Before submitting new changes please read the following:

- If you are requesting changes for LHCBCOND partition please check that the IOV(s) for your files are not covering the current closed IOVs (the latter can be checked using CondDB Browser). If you are confident that the current closed IOV scheme should be modified please inform the CondDB manager about that explicitly in the JIRA task.
- Please check the actual encoding of your xml files matches the one declared in the xml header in the 'encoding' attribute.

- Please be sure that your changes to the db intended for particular global tag branch are made on the base of the same branch of your interest.

The format in which changes should be provided is described below.

Add new XML to the Conditions Database

Users can provide new XML to be included in the CondDB in 2 formats:

- XML files
- an SQLite file

From XML files

- Copy the modified files in dedicated directories, for example `~/tmp/DDDB-new` and `~/tmp/LHCBCOND-new`
- Create COOL databases to be used as layers

```
# lb-run LHCb bash --norc
# copy_files_to_db.py -c sqlite_file:$HOME/tmp/DDDB-new.db/DDDB -s ~/tmp/DDDB-new
# copy_files_to_db.py -c sqlite_file:$HOME/tmp/LHCBCOND-new.db/LHCBCOND -s ~/tmp/LHCBCOND-
```

- Create an option file to be used for testing (e.g. after `LoadDDDB.py` in `DetDescChecks`) with the following lines:

```
from Gaudi.Configuration import *
from Configurables import CondDB
conddb = CondDB()

# For DDDB
conddb.addLayer(dbFile = "$HOME/tmp/DDDB-new.db", dbName = "DDDB")

# For LHCBCOND
conddb.addLayer(dbFile = "$HOME/tmp/LHCBCOND-new.db", dbName = "LHCBCOND")
```

- If the validation is successful, add the XML files to the master database

```
# set MasterDDDB = "sqlite_file:$SQLDDDBROOT/db/DDDB.db/DDDB"
# set MasterLHCBCOND = "sqlite_file:$SQLDDDBROOT/db/LHCBCOND.db/LHCBCOND"
# copy_files_to_db.py -c $MasterDDDB -s ~/tmp/DDDB-new
# copy_files_to_db.py -c $MasterLHCBCOND -s ~/tmp/LHCBCOND-new
```

- Tag the new data either manually with the CondDB browser (boring but avoids creating duplicated tags) or automatically

```
# python
Python 2.4.2 (#1, Mar 24 2006, 16:38:17)
[GCC 3.4.5 20051201 (Red Hat 3.4.5-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from CondDBUI import CondDB
>>> import os
>>> NewTag = "DC06-p1"
>>> MasterDDDB = CondDB("sqlite_file:%s/db/DDDB.db/DDDB"%os.environ["SQLDDDBROOT"], readOn
>>> MasterDDDB.recursiveTag("/", NewTag)
>>> MasterLHCBCOND = CondDB("sqlite_file:%s/db/LHCBCOND.db/LHCBCOND"%os.environ["SQLDDDBRO
>>> MasterLHCBCOND.recursiveTag("/", NewTag)
```

From SQLite files

TODO

Update the Oracle Connections string on LFC

From `lxplus5`, prepare the environment with

```
lhcb-proxy-init -g lhcb_conddb
lb-run LHCb bash --norc --use-grid
setenv PATH `echo $LD_LIBRARY_PATH | tr ':' '\n' | grep "CORAL.*lib"`/../bin:$PATH
```

Go to a temporary directory under `$HOME/private` (e.g. `$HOME/private/tmp`) and dump the old version of the connection details for the interested site (replace `LCG.NIKHEF.nl` with the corerct site name):

```
cd $HOME/private/tmp
set site = LCG.NIKHEF.nl
coral_replica_manager -exp -l CondDB -h $site
mv authentication.xml offline_auth.xml
mv dblookup.xml offline_lookup.xml
coral_replica_manager -exp -l CondDBOnline -h $site
mv authentication.xml online_auth.xml
mv dblookup.xml online_lookup.xml
```

Extract from the `*_auth.xml` files the passwords for the accounts

- `lhcb_conddb`
- `lhcb_online_conddb`
- `lhcb_conddb_reader`.

Now you can remove the old entries from LFC

```
setenv LFC_HOST lfc-lhcb.cern.ch
coral_replica_manager -del -l CondDB -h $site
coral_replica_manager -del -l CondDBOnline -h $site
```

With the passwords retrieved before and the Oracle connection details (taken from TNSNAMES), set some variables

```
set conddb_pwd = "..."
set online_pwd = "..."
set reader_pwd = "... "
set oracle_string = "..."
```

then add the new entries to LFC

```
coral_replica_manager -add -l CondDB -c "oracle://$oracle_string/lhcb_conddb" -h "$site" -r owner
coral_replica_manager -add -l CondDB -c "oracle://$oracle_string/lhcb_conddb" -h "$site" -r reade
coral_replica_manager -add -l CondDB -c "oracle://$oracle_string/lhcb_conddb" -h "$site" -ro -u l
coral_replica_manager -add -l CondDBOnline -c "oracle://$oracle_string/lhcb_online_conddb" -h "$s
coral_replica_manager -add -l CondDBOnline -c "oracle://$oracle_string/lhcb_online_conddb" -h "$s
coral_replica_manager -add -l CondDBOnline -c "oracle://$oracle_string/lhcb_online_conddb" -h "$s
```

The last thing is to use "export" the new values to validate the content of LFC. Also, don't forget to make the corresponding changes to all the files in the SVN directory DBASE/AppConfig/conddb, and to ask for a new release of AppConfig with these changes.

Synchronize Oracle Databases from SQLITE versions (OBSOLETE, see new procedure)

From `lxplus5` (has not been validated on `lxplus6`), prepare the environment with

```
lhcb-proxy-init -g lhcb_conddb
lb-run LHCb bash --norc --use-grid
```

then call the `coolReplicateDB` tool on the three partitions `DDDB`, `LHCBCOND` and `SIMCOND`

```
coolReplicateDB sqlite_file:$SQLITEDBPATH/DDDB.db/DDDB "CondDB(owner)/DDDB"
coolReplicateDB sqlite_file:$SQLITEDBPATH/LHCBCOND.db/LHCBCOND "CondDB(owner)/LHCBCOND"
coolReplicateDB sqlite_file:$SQLITEDBPATH/SIMCOND.db/SIMCOND "CondDB(owner)/SIMCOND"
```

If there are new files in the propagated changes, then also the permissions have to be updated:

```
coolPrivileges "CondDB(owner)/DDDB" GRANT READER lhcb_conddb_reader
coolPrivileges "CondDB(owner)/LHCBCOND" GRANT READER lhcb_conddb_reader
coolPrivileges "CondDB(owner)/SIMCOND" GRANT READER lhcb_conddb_reader
```

Snapshotting the "ONLINE"

You need to have a GRID certificate in order to be authenticated to snapshot the ONLINE partition from Oracle.

First, decide for what time interval You want to make the snapshot of the ONLINE partition. For this exercise we will create the snapshot for the time interval from 2009-10-01 to 2009-11-01 to renew the ONLINE partition for the Det/SQLDDDB package (located in DEV area) with the last missing snapshot for October of 2009 (For the note: The ONLINE partition in SQLDDDB package consists of monthly snapshots of the Oracle online database).

From `lxplus5`, setup the environment with enabling auto selection of LHCbGrid project:

```
lb-run LHCb bash --norc --dev --use-grid
```

Next, initialize the proxy:

```
lhcb-proxy-init
```

and enter the certificate password. Now, put Yourself to the folder, where You want to place the snapshot:

```
cd $SQLITEDBPATH/
```

and execute the following script, specifying the option file "SQLDDDB-Oracle.py" (to use the Conditions database) from Det/SQLDDDB package, start and end points of time interval to snapshot (either local, or UTC), the name of the partition and the connection string for the output snapshot file. Note, that currently the snapshotting is done internally using the UTC time, so if you define the range in local time (e.g. ... -s 2010-02-01 -u 2010-03-01 ...) the snapshot will be done using the time range converted to UTC. For LHCb ONLINE snapshots it is important to have a snapshot exactly from midnight to midnight for every month in UTC and to avoid taking into account 2 (or 1, depending on summer time) hours shifts it is convenient to define the time range always in UTC as it is shown in the example below:

```
CondDBAdmin_MakeSnapshot.py --options $SQLDDDBROOT/options/SQLDDDB-Oracle.py -s
2010-02-01UTC -u 2010-03-01UTC ONLINE sqlite_file:ONLINE-201002_new.db/ONLINE
```

The snapshot is done.

For our purposes, as far as we wanted to update the Det/SQLDDDB package, the last additional thing we must do is to edit the file `$SQLITEDBPATH/options/SQLDDDB.py` to declare the last available snapshot for the ONLINE partition. For that purpose, we must specify the year and the month of the last snapshotted ONLINE part in the end of the file in the line:

```
latest_snapshot = (2010,02)
```

Now Det/SQLDDDB package contains the most recent version of ONLINE partition.

This topic: LHCb > CondDBHowToForCOOL
Topic revision: r69 - 2017-06-30 - unknown