# Table of Contents

# LHCb Conditions Database How-To

# 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

Since 2017 we started using Git CondDB as Conditions Database Backend (since 2018 also in old versions of the software). The old instructions, referring to COOL, are kept in CondDBHowToForCOOL.

# Test and Development Tasks

Git CondDB uses, of course, a Git repository as storage for conditions, which means that normal Git based workflows can be used for conditions data.

## 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).

## 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

To create a local copy of one of the partitions of the Git CondDB (`DDDB`, `LHCBCOND`, `ONLINE`, `SIMCOND` or `DQFLAGS`) it is enough to clone one of the repositories in https://gitlab.cern.ch/lhcb-conddb. For example (you can replace `LHCBCOND` with the the partition you need):

```
git clone ssh://git@gitlab.cern.ch:7999/lhcb-conddb/LHCBCOND.git
```

**Note:** It's mandatory that the name of the directory is the same as the project (without `.git`) to be correctly found by the jobs.

#### Use the Copy in Your Application

See Git CondDB development instructions.

## Use a Local Tag of the database

"Local tags" are a feature of the old CondDB needed to work around the absence of proper development branches and merge requests.

You should use branches as described in Git CondDB development instructions.

## 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 can create a branch from one of the tags and inject the changes from the other tag:

```
export GITCONDDBPATH=$PWD
git clone ssh://git@gitlab.cern.ch:7999/lhcb-conddb/LHCBCOND.git
cd LHCBCOND
git checkout -b my-test-branch head-20100119
git checkout head-20091211 /Conditions/Rich2
git commit -m 'use Rich2 conditions from head-20091211'
lb-run Brunel/vXrY gaudirun.py my-options.py --option 'from Configurables import CondDB; CondDB()
```

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

In order to ensure that replicas of the ONLINE partition of the database are sufficiently up to date we use two special concepts:

- "heartbeat": we consider the ONLINE partition to be valid until the time of the last commit (we cannot know if an event taken after that moment requires new conditions or not)
- "run stamp": a special file (`/Conditions/Online/valid_runs.txt`) contain the list of all runs for which alignment/calibration have been produced and validated (before HLT2 processing)

The framework blocks access to the CondDB if one of the two conditions are not met 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
```

Obviously this check does not make sense in some special cases (e.g. when preparing alignments, or when running 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 LHCBPS-828 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, create a merge request in the relevant project(s) under https://gitlab.cern.ch/lhcb-conddb.

## Add new XML files to the Conditions Database

If you have the files to add to the database in a directory, you can use the script `add_files_to_gitconddb.py` to add the content of a directory to a (subdirectory of a) Git CondDB targeting a limited Interval of Validiy.

---