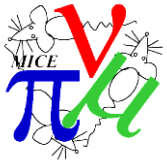

Configuration Database

Janusz Martyniak, Imperial College London

MICE CM44 Analysis, Software and Computing

Configuration Database

project status

- Run Control (beamline) and Cooling Channel C API status
- Absorber table
- Data and reco quality flags API
- Geometry corrections proposal

Configuration DB (contd.)

Beamline C API

- The API has been written in C (gSOAP) to facilitate integration with run control code
- Successfully implemented for:

1. Retrieving beamline settings (tags)form the database
2. Storing beamline magnet data to the hardware
3. Storing the settings to the CDB for a given run
4. Storing new tags

- Project on Launchpad:

bzr+ssh://bazaar.launchpad.net/~janusz-martyniak/mcdb/mice.cdb.client.api-C/

Beamline API – experience so far

- Used for the current run period
- The setter functions signature found to be too restrictive (they return an *int*)
- We cannot make use of a message (error or success) returned by the server as it is currently dropped and only a non zero return code signals problems with writing
- We don’t know the request XML message which would be very useful for client side logging

Beamline API (contd.)

We decided to change the type returned by the setter functions to be a struct.

The struct contains:

- Return code (*int*, 0 means OK)
- Return XML message for display
- Request XML message for client-side logging

It is **extremely important** to verify that the return code is 0, failing to do so might leave us with incomplete run information in the CDB.

The logs (client and server side) will allow to retrofit missing information. We have done it once (!) already…

The Cooling Channel API

- Cooling channel magnet setting are now written to the CDB during data taking.
- The settings are stored in from of tags in the CDB, and we know the run ranges the tags apply to.
- We have found a design problem the the Cooling Channel tables:
  1. The CC data were originally stored with the (server) timestamp and the timestamp was used to match the CC records with beamline records.
  2. The beamline data were stored with the client timestamp with lower accuracy (no milliseconds) which made the comparison prone to errors.
  3. We think that the CC should be written out *after* the beamline data to avoid orphaned CC records
  4. The CC timestamp is later than the Beamline timestamp which gives us wrong matching (we select the latest CC record <= run record)

Cooling Channel CDB API (contd.)

To avoid potential problems with CC – run number matching , we decided to change the server

code to handle run based CC records which is more natural. Implemented:

- Settting the CC data by run number
- getCoolingchannelForRun(run number) – critical
- getAllCoolingchannels()
- It handles both old style timestamp based CC and the new ones.
- Tags (insertion and retrieval are not affected)
- Ready for testing (we need a non running period)

Absorber Table

We design a table holding absorber data (issue #1816) :

for each absorber:

- name (string)
- material (string)
- shape (string)
- comments (string)

We need C-API to allow run control to store the data. We plan to keep absober handling within the Cooling Channel Web Service, but just add another method to set absorber data.

The API will use similar style as other Cooling Channel APIs

Data and Reco Quality Flags

Only reconstruction quality flags implemented so far (issue #1689).

Python API:

def get_reconstruction_flags(self, run_number, maus_version, batch_iteration_number):

- The flag will be a hex string 0xFFFFFFFFFFF, with each hex digit (4 bits) representing one of the following detectors

- TOF0, TOF1, CkovA, CkovB, Tracker0, Tracker1, TOF2, KL, EMR, RF1, RF2

We also have:

def get_reconstruction_flag_for_detector(self, detector, run_number, maus_version, batch_iteration_number)

Geometry Corrections

We want to store in a CDB table, alignment and analysis-based corrections to the surveyed geometry (issue #1817).

- The corrections should be bound to geometry id
- We should allow run by run correction i.e. multiple correction sets for one gid.
- We should be able to modify (update) corrections in a similar way we are updating geometry itself (by selecting the latest geometry creation time)

Geometry Corrections

This leads us to 2 tables:

geometrycorrections table:

- ID (autoincrement int ?)
- Geometry ID (int)
- Run start
- Run end (problematic, not easy to find one, if omitted)
- Creation time

Geometry Corrections

And a module table:

- Correction id (int)
- Module_Name (string)
- dx(float)
- dx_err (float)
- dy (float)
- dy_err (float)
- dz (float)
- dz_err (float)
- dx_rot (float)
- dx_rot_err (float)
- dy_rot (float)
- dy_rot_err (float)
- dz_rot (float)
- dz_rot_err (float)

CDB - Summary

- CDB is fully operational.
- New public interface WeB Service has been installed
- Failover has been tested
- C-API for Beamline and Coolingchannel has been updated and is awaiting installation in in the Control Room
- Work on geometry corrections started
- Work on absorber tables next in a queue
- State machine C-API written not yet installed