Table of Contents

Goal

This is my private documentation page on CT-PPS Database development.

For the moment, the focus will be on the conditions related to the accelerator and beam as well to the pot insertion into the beam pipe.

Beam and Roman Pot position related conditions

- Crossing angle
- beta-star
- magnets currents
- Raw Roman Pot insertion

Contact People

Subsystem/Activity	Name	E-mail
CT-PPS (offline conditions)	Clemencia Mora Herrera	clemencia.mora.herrera@cernNOSPAMPLEASE.ch
	Helena Malbouisson	helena.malbouisson@cernNOSPAMPLEASE.ch
CT-PPS	Diego Figueiredo	dmf@cernNOSPAMPLEASE.ch
OMDS	Umeshwar Joshi	joshi@fnalNOSPAMPLEASE.gov
DB Loader setup	Aivaras Silale	aivaras.silale@cernNOSPAMPLEASE.ch
OMS (replacement for WBM)	Valdas Rapsevicius (main contact)	valdas.rapsevicius@cernNOSPAMPLEASE.ch
	Aivaras Silale	aivaras.silale@cernNOSPAMPLEASE.ch

Related links

- CTPPS Database page
- Clemencia's page
- Helena's page
- CMS CondDB Browser
- AlcaDB
- Conditions DB Software Guide
- Conditions Objects Tutorial
- IOV in a Nutshell
- CMS Pixel Database Instructions page with lots of links and information
- Persistency Framework project web site
- CORAL WebSVN repository
- Diego's tools:
    - CTPPS DB Viewer
    - View **RUN** info
    - View **FILL** info
    - Roman Pots info

Accelerator databases at CERN

- LSA (LHC Software Architecture): accelerator optics models, control sequences, reference values, etc.
- Timber
- DIP (Data Interchange Protocol): communication system which allows relatively small amounts of soft real-time data to be exchanged between very loosely coupled heterogeneous systems.
    - Dip Browser

At work

Set up working area and environment

```
$ cmsrel CMSSW_9_4_0_pre2
$ cd CMSSW_9_4_0_pre2/src
$ cmsenv
$ git cms-init
$ git cms-addpkg CondCore/CTPPSPlugins
$ git cms-addpkg CondFormats/DataRecord
$ git cms-addpkg CondFormats/CTPPSReadoutObjects
$ scramv1 b -j20
```

Step by step

How to create a Conditions Object

1. Move to `CondFormats/CTPPSReadoutObjects` package
2. Define new class `LHCInfoForCTPPS` to hold LHC related info: `interface/LHCInfoForCTPPS.h`
3. Add the following line to `src/headers.h`: `#include "CondFormats/CTPPSReadoutObjects/interface/LHCInfoForCTPPS.h"`
4. For the class be available as EventSetup data, an LCG dictionary must be created for it. This is achieved by changing `classes_def.xml` and `classes.h` files in the project `/src` area:
    - `src/classes.h` holds the class definitions.
    - `src/classes_def.xml` contains the names of all classes to be stored and their constituents, as well as class template instantiations.
5. Add test serialization code `test/testSerializationLHCInfo.cc` and edit `test/BuildFile.xml` accordingly:
    - ```
      <bin file="testSerializationLHCInfo.cc">
          <use   name="CondFormats/CTPPSReadoutObjects"/>
      </bin>
      ```
6. For the class to be useable as EventSetup data, it needs to be registered into the CMSSW framework as such. Add `T_EventSetup_LHCInfoForCTPPS.cc` to src and change `BuildFile.xml` accordingly by adding:
    - ```
      <use name="FWCore/Utilities"/>
      ```
7. Next define a **Record** where the objects are actually stored and from which it can be retrieved. The record class must be defined in `CondFormats/DataRecord`.
    - ```
      $ cd $CMSSW_BASE/src/CondFormats/DataRecord/interface
      $ mkrecord LHCInfoForCTPPSRcd
      $ mv LHCInfoForCTPPSRcd.cc ../src
      ```

      followed by

      ```
      $ cd ..
      $ scramv1 b -j20
      ```
8. Tell the DB interface which data do associate to which record. Go to `$CMSSW_BASE/src/CondCore/CTPPSPlugins/src/` and add the following lines to
      ```

plugin.cc:

```
#include "CondFormats/CTPPSReadoutObjects/interface/LHCInfoForCTPPS.h"
#include "CondFormats/DataRecord/interface/LHCInfoForCTPPSRcd.h"

REGISTER_PLUGIN(LHCInfoForCTPPSRcd,LHCInfoForCTPPS);
```

9. Build the plugin system and check that the proxy for the EventSetup has been correctly built:

```
$ scramv1 b -j20
$ edmPluginDump | grep LHCInfoForCTPPSRcd
LHCInfoForCTPPSRcd@NewProxy
```

How to fill the DB with data

1. Create a new (dummy) subsystem

```
$ cd $CMSSW_BASE/src
$ mkdir MyCondCTPPS
```

2. Create an analyzer

```
$ mkedanlzr LHCInfoMaker
```

3. Go to `LHCInfoMaker/plugins/` and change `LHCInfoMaker.cc` for the analyze method to look like

```
void LHCInfoMaker::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
   using namespace edm;

   LHCInfoForCTPPS* pInfo = new LHCInfoForCTPPS();

   // Form the data here

   edm::Service<cond::service::PoolDBOutputService> poolDbService;
   if( poolDbService.isAvailable() )
      poolDbService->writeOne( pInfo, poolDbService->currentTime(),
                                                    "LHCInfoForCTPPSRcd"  );
   else
      throw std::runtime_error("PoolDBService required.");

}
```

4. Change also `BuildFile.xml` to look like

```
<use name="FWCore/Framework"/>
<use name="FWCore/PluginManager"/>
<use name="FWCore/ParameterSet"/>
<use name="FWCore/ServiceRegistry"/>
<use name="CondCore/DBOutputService"/>
<use name="CondFormats/CTPPSReadoutObjects"/>
<flags EDM_PLUGIN="1"/>
```

5. Compile

```
$ cd $CMSSW_BASE/src/MyCondCTPPS
$ scram b
```

6. Add the `LHCInfoMaker/test/test-maker_cfg.py` configuration file to run the analyzer and create an output file. DB service is configured to write to a SQLite file.

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("lhcInfoTest")

# Load CondDB service
process.load("CondCore.CondDB.CondDB_cfi")

# output database (in this case local sqlite file)
process.CondDB.connect = 'sqlite_file:LHCInfoForCTPPS.db'

# A data source must always be defined. We don't need it, so here's a dummy one.
```

```
process.source = cms.Source("EmptyIOVSource",
    timetype = cms.string('runnumber'),
    firstValue = cms.uint64(1),
    lastValue = cms.uint64(1),
    interval = cms.uint64(1)
)

# We define the output service.
process.PoolDBOutputService = cms.Service("PoolDBOutputService",
    process.CondDB,
    timetype = cms.untracked.string('runnumber'),
    toPut = cms.VPSet(cms.PSet(
        record = cms.string('LHCInfoForCTPPSRcd'),
        tag = cms.string('LHCInfoForCTPPS_test')
    ))
)

process.pedestals_maker = cms.EDAnalyzer("LHCInfoMaker",
    record = cms.string('LHCInfoForCTPPSRcd'),
    loggingOn= cms.untracked.bool(True),
    SinceAppendMode=cms.bool(True),
    Source=cms.PSet(
        IOVRun=cms.untracked.uint32(1)
    )
)

process.path = cms.Path(process.pedestals_maker)
```

7. Run the configuration file
   - `$ cmsRun test/test-maker_cfg.py`

   creating the file `LHCInfoForCTPPS.db`.

How to retrieve the data

1. Requires the configuration of the DB interface, this time using the `ESSource`. The `LHCInfoMaker/test/test-retrieve_cfg.py` configuration file is used.
   - `$ cmsRun test/test-retrieve_cfg.py`

To Do:

- Test "lumisection" as IOV timetype
- Try to store/retrieve the object through **ORACLE** on cmsprep. Using the following parameters in the _cfg.py
  - `process.CondDB.connect = "oracle://cmsprep/CMS_COND_TASKNAME"`

  Note: replace "CMS_COND_TASKNAME" with the account name allocated to your task.

  `process.CondDB.DBParameters.authenticationPath = 'where_it_is'`

LHCInfo

From: Giacomo Govi <giacomo.govi@cern.ch> Date: 2018-05-15 6:48 GMT-03:00
Subject: LHCInfo data collected by the O2O

the LHCInfo O2O has been successfully deployed and it is currently collecting IOV and Payloads since the 17 april 2018 on the tag: LHCInfoEndFill_prompt_v0:

https://cms-conddb-prod.cern.ch/cmsDbBrowser/list/Prod/tags/LHCInfoEndFill_prom

The O2O is probing every hour the LHC tables for the existence of a completed Fill, which is sampled every ten minutes during its entire duration. In others words, we have IOVs within a Fill every ten minutes. We have included in the payload all the information requested.

This tag is not yet part of any Global Tag, although Candidates GT can be produced for testing purposes.

We would like to receive feedback on the content as well as the data collection model. We have developed a tool for dumping the payload in a human readable way (in CMSSW since 10_1_4)

Example:

```
conddb_dump_LHCInfo -c frontier://FrontierProd/CMS_CONDITIONS -i b4918fe664d58a72a23d0364c2780fb3
```

It would be useful to start the development of the consumer code for the LHCInfo payloads.

Optical Functions

Production of conditions data for optical functions consumed by the proton reconstruction code developed by Jan Kaspar.

Data taking period	Run interval
2016 pre-TS2	273725 - 280385
2016 post-TS2	282730 - 284044
2017	294730 - 306462
2018 before 90 m run	314158 - 319077
90 m run	? - ?
2018 between 90 m and 900 GeV runs	319337 - 324420
900 GeV run	? - ?
2018 after 900 GeV run	324612 - 325175

-

-- WagnerDePaulaCarvalho - 2017-09-15

This topic: Main > WPCPrivateCTPPSDBPage
Topic revision: r22 - 2019-04-26 - WagnerDePaulaCarvalho