MarlinTPC - LCCD
Conditions Handling and Database

Ralf Diener

- Usage of LCCD and Conditions Data
- Which Condition Data should we store?
- Design of Conditions Objects
- Database Layout
  Physical and Logical

- Use of LCCD package which is based on ConditionsDBMySQL
- Conditions can be stored in several ways, all writing and reading over lccd::DBInterface class.
  - Simple File:
    - LCIO file with one event that has a collection with the conditions data for the given time stamp and tag
  - MySQL database:
    - LCCollections of condition objects are stored in a folder structure in a MySQL database (one kind of object per folder)
    - Conditions data has a start and end time information:
      usually end time is set to far future and conditions handler always chooses the newest information (in the conditions time line)
    - Information can be tagged, that's the only way to get older conditions data if you write newer data for the same time slot
  - DBFile:
    - Creates an LCIO file with the all conditions data in the folder for the given tag
    - Useful when no database connection is available
- An example can be found in LCCD code under `example/calomap`

- Condition Objects:
  - Derived from `EVENT::LCGenericObject` or `UTIL::LCFixedObject<NIintVals,NFloatVals,NDoubleVals>`

- These Objects are stored in a LCCollectionVec:

```
lcColVec->addElement( condObj );
```

- Then the data can be stored in the database:

```
lccd::DBInterface
dbConnection( "localhost:tpcconditions:tpcuser:passwd",
"/test/conditionsObjects", true ) ;

dbConnection.storeCollection( runStart.timeStamp(),
farFuture.timeStamp(), lcColVec, "ConditionsName");
```

- For this to work you need a MySQL Server running and the user `tpcuser` has to have the rights to create databases, tables and write into them
  If this is given, everything is created automatically if it does not exists before

Retrieving Conditions Data

- Include ConditionsProcessor in Reconstruction Chain:

```
<processor name="ConditionsInitialization" type="ConditionsProcessor">
<parameter name="DBCondHandler" type="StringVec">TPCPedestal /test/pedestals
HEAD</parameter>
<parameter name="DBInit" type="string">localhost:tpcconddata:tpcuser:passwd</
parameter>
</processor>
```

- Access via event time stamp in your processor:
  - Either by a special handler, derived from `IconditionsChangeListener` :

```
class PedestalHandler : public lccd::IConditionsChangeListener
```

    This should be done if you have to do some calculations when the conditions have changed. So the handler object stores the calculated/derived data and only updates this when conditions change via `conditionsChanged` member function (it registers with the ConditionsManager:

```
lccd::LCConditionsMgr::instance()->getHandler(ColName)
->registerChangeListener(this); )
```

  - Or simply from the event (the ConditionsProcessor packs it in the stream)

```
evt->getCollection("TPCDummyCond" );
```

  - The you cast the elements of the LCCollectionVector back to the condition objects

- **ADCChannelMapping:**
  Mapping of H/W channels to GEAR pad ind.
  - ChannelID
  - PadID
  - Type
- **ChannelCorrection:** Per channel calib.
  - Quality flags (broken, noisy)
  - Calibration factors
  - Time offset
- **Pedestal:** Per channel
  - Value
  - Width
- **TPCConditions:** Calibr. TPC Parameters
  - DriftVelocity
  - Diffusion (trans/long)
  - "Defocussing"
  - Amplification
- **GasConditions**
  - Pressure
  - Temperature
  - Flow
  - OxygenContent
  - WaterContent
- **GasMixture**
  - Contents
  - Fractions
- **FieldSettings**
  - Nominal drift field
  - Nominal B-Field
  - Especially for GEMs:
    - GEM voltages
    - Transfer fields
  - Or general voltage 1,2,..., n

See http://forum.linearcollider.org/index.php?t=msg&th=565&rid=0

Which Condition Data → Objects

- **TimePixPixelMode**
  - Mode
  - Status (broken/noisy)
- **Read-Out Electronics**
  - Polarity
  - Readout frequency
  - Precision of electronics (maximum ADC count)
  - Specific settings for each type of read-out (ALTRO, T2K, TDC) information? → specific objects per type
- **WeatherConditions**
  - Temperature
  - Humidity
  - Pressure
- **Unsorted**
  - ADC ↔ Primary Electrons

- **My proposal:**
  - Take this list as a basis and check if we have everything that is needed
  - Every main item in this list will get an object assigned
  - Implement these Objects to store them in the database
- **Already in SVN** (status unknown to me)
  - GasConditions + GasMixture
  - Pedestal
  - Fieldsettings
  - ChannelCorrection + Mapping
  - TPCConditions
  - WeatherConditions
  - TimePixPixelMode

See http://forum.linearcollider.org/index.php?t=msg&th=565&rid=0

- **Need to decide on folder structure**
  - CALICE has sorted by detector and location
  - Proposal: /lctpc/large_prototype_1/<location>/<condition>[/<module>]
    Should be discussed!

- **Tagging?**
  - Proposal: start with 1.0 for all conditions and then increase version number by 0.1 for minor, by 1.0 for major corrections

- **Physical servers and database setup**
  - ConditionsDBMySQL is able to store data splitted over several database servers
    Do we need this?
  - CALICE uses 2 machines:
    - One is the main database, to which the Slow Control etc. writes the data
    - This data is duplicated to a second machine, which can be accessed by the users for the reconstruction

- Remove obsolete DBEntryMaker and DBWriter classes from the repository

- Set up test database somewhere
- Review the different Condition Objects:
  do they have the functionality we need and do they do what is expected?
- Implement a test setup: create dummy data and read out from database for all condition objects and check if the values are right and it works

- Decide on a folder structure
- Set up the physical computers to host a central database
  (includes backup strategies)

- Personal agenda:
  Set up a wiki page with step by step guide how to use the conditions (LCCD):
  set up database, avoid pitfalls and a simple example