LCCD
Linear Collider Conditions Data Toolkit

Frank Gaede, DESY
SLAC Simulation Meeting 2005
March 16-17 2005
Outline

- Introduction
- Overview LCCD
- Design
- Implementation
- Status
- LCIO/Marlin
- Summary/Outlook
Introduction

**Linear Collider Conditions Data Toolkit**

- handles access to conditions data transparently from
  - conditions database (CondDBMySQL)
  - LCIO files

**Conditions Data:**

- all data that is needed for analysis/reconstruction besides the actual event data
- typically has lifetime (validity range) longer than one event
  - can change on various timescales, e.g. seconds to years
  - need for tagging mechanism, e.g. for calibration constants
LCCD dependencies

Package dependencies

MySQL

ConditionsDBMySQL

LCCD

LCIO

Marlin

optional
ConditionsDBMySQL - Overview

- Open source implementation of CondDB API
  - conditions data interface for ATLAS (Cern IT)
  - developed by Lisbon Atlas group
- features
  - C++ interface to conditions database in MySQL
  - data organized in folder/foldersets
  - objects stored as BLOBs (binary large objects)
  - tagging mechanism similar to CVS
  - scalability through partitioning options
  - outperforms implementation based on Oracle
- status
  - currently put on hold by CERN management
  - no active development - but bug fixes
- remark: first tests suggest that software runs stable
  - need extended tests before used in production environment
CondDBMySQL - Tagging

Figure 3: tagging and browsing example in the ConditionsDB mySQL’s implementation.
LCCD Uses Cases (Reading)

LCCD Use Cases

- conditions from DB
  - standard use case: read conditions from data base for events timestamp and optionally provided tag

- conditions from special cond. LCIO file
  - read one set of conditions data from LCIO file - no time intervall specified, e.g. calibration constants

- conditions from LCIO file for time intervall
  - read conditions from an LCIO file that contains all needed conditions for a given time intervall (could have been extracted before from condDB)

- conditions from within data LCIO file
  - read data stored with event stream, e.g. slow control data
LCCD Design

Frank Gaede, DESY, SLAC-Simulation-Meeting March 16/17 2005
LCCD DBInterface

- offers **read/write** access to the conditions db:
  - store collections of LCGenericObjects
  - tag folders
  - retrieve collection for time stamp and tag
    - used in DBCondHandler (reconstruction job)
- create conditions data files (LCIO)
  - for complete tag, used by DBFileHandler
    - horizontal browsing
  - for time stamp and tag, used by SimpleFileHandler
  - for all collections for given timestamp
    - vertical browsing
- in principle no direct access through ConditionsDB interface needed – all typical tasks can be performed through LCCD::DBInterface!
IConditionsHandlers

- need one instance of either implementation for every type of conditions data:
  - **SimpleFileHandler**
    - reads one collection from LCIO file
    - valid for all events
  - **DBCondHandler**
    - reads collection from db if needed by current event
  - **DBFileHandler**
    - reads collections from LCIO file that has events for all validity intervals for a given tag
    - database snapshot - create with DBInterface
  - **DataFileHandler**
    - takes collection out of current event and keeps them until another event provides a newer version
VCollectionStreamer

- provides streamer code to store conditions data in db-blob
- uses XDR format -> machine independent
- so far only LCGenericObjectStreamer
- in principle users can provide their own streamers for their classes but then they can only use the database and not LCIO files -> discouraged!
- LCGenericObject should be generic enough to store all conditions data
Example: LCGenericObject subclass

```cpp
#ifndef CalibrationConstant_h
#define CalibrationConstant_h

#include "lcio.h"
#include "UTIL/LCFixedObject.h"

#define NINT 1
#define NFLOAT 2
#define NDOUBLE 0
#define ID_INDEX 0
#define OFFSET_INDEX 0
#define GAIN_INDEX 1

using namespace lcio;

class CalibrationConstant {

/** Example for a simple calibration class based on the LCFixedObject template. */

public:

/** Convenient c'tor. */
  CalibrationConstant(int cellID, float offset, float gain) :
   obj()->setIntVal( ID_INDEX , cellID ) ;
   obj()->setFloatVal( OFFSET_INDEX , offset ) ;
   obj()->setFloatVal( GAIN_INDEX , gain ) ;

/** 'Copy constructor' needed to interpret LCObject read from file/database. */
  CalibrationConstant(LCObject* obj) : LCFixedObject<NINT,NFLOAT,NDOUBLE>(obj) {} 

/** Important for memory handling */
  virtual ~CalibrationConstant() { /* no op */ }

  // the class interface:
  int getCellID() { return obj()->getIntVal( ID_INDEX ) ; }
  float getOffset() { return obj()->getFloatVal( OFFSET_INDEX ) ; }
  float getGain() { return obj()->getFloatVal( GAIN_INDEX ) ; }

  void print( std::ostream& os ) ;

  // ------- need to implement abstract methods from LCGenericObject

  const std::string getObjectName() const {
    return "CalibrationConstant" ;
  }

  const std::string getDataDescription() const {
    return "1:cellID,2:offset,3:gain" ;
  }

private:

};
```

ConditionsMap

- template class ConditionsMap<key,CondObject>
- example:

  ConditionsMap<int, Pedestal> myMap =
  new ConditionsMap<int, Pedestal>( & Pedestal::getCellID);
  myCondHandler->registerChangeListener( myMap );

  ...
  float pedestal = myMap->find( hit->getCellID() ).getPedestal();
  ...

- if registered with IConditionsHandler the map will be updated automatically whenever the conditions have changed
LCCD Status

- v00-02 released
- fairly complete functionality
- passes simple tests
- example/test code
- complete API documentation
- available via cvs web: http://ilcsoft.desy.de/lccd
- feedback welcome
- CALICE will use LCCD for testbeams
LCCD and LCIO

LCCD requires current version of LCIO (v01-04)

changes in LCIO:

- unique definition of 64bit time stamp in LCEvent :
  - ns since 1/1/1970 (UTC)
- LCEvent::getTimeStamp() now long64
- UTIL::LCTime:
  - conversion between real date/time and time stamp
- UTIL::LCFixedObject
  - provides convenient and efficient way to define conditions data classes
- LCEvent::takeCollection( name )
  - technical: need way to tell event that it is no longer the owner of the collection (memory management)
- handling of multiple events and I/O streams
LCCD and Marlin

- Marlin v00-08 has ConditionsProcessor:
  - provides transparent access to conditions data
  - conditions data available in event collections
  - activate through steering file:
    ```
    .begin MyConditionsProcessor
    ProcessorType ConditionsProcessor
    DBInit localhost:lccd_test:calvin:hobbes
    DBCondHandler conditionsName /lccd/myfolder HEAD
    #DBFileHandler conditionsName conditions.slcio collectionName
    #DataFileHandler conditionsName
    #SimpleFileHandler conditionsName conditions.slcio collectionName
    .end
    ```
  - conditions available in event collections or conditions maps independent of concrete conditions handler
  - example/test code in `$LCCD/examples/calomap`
Summary & Outlook

- LCCD is a simple toolkit for retrieving and storing conditions data transparently through
  - LCIO files
  - conditions database (ConditionsDBMySQL)
- v00-02 released see [http://ilcsoft.desy.de/lccd](http://ilcsoft.desy.de/lccd)
- future plans:
  - need user feedback
  - Java interface / interoperability with `org.lcsim`?
  - use for geometry definition in reconstruction?

NB: while LCCD provides a simple user interface to access the conditions database it doesn’t deal with setting up and managing that database, e.g.
- security issues (read/write access)
- data integrity
- availability/ quality of service...