



# LCCD

## Linear Collider Conditions Data Toolkit



FLC Software Meeting

February 24, 2005

Frank Gaede DESY -IT-



# Outline

---

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



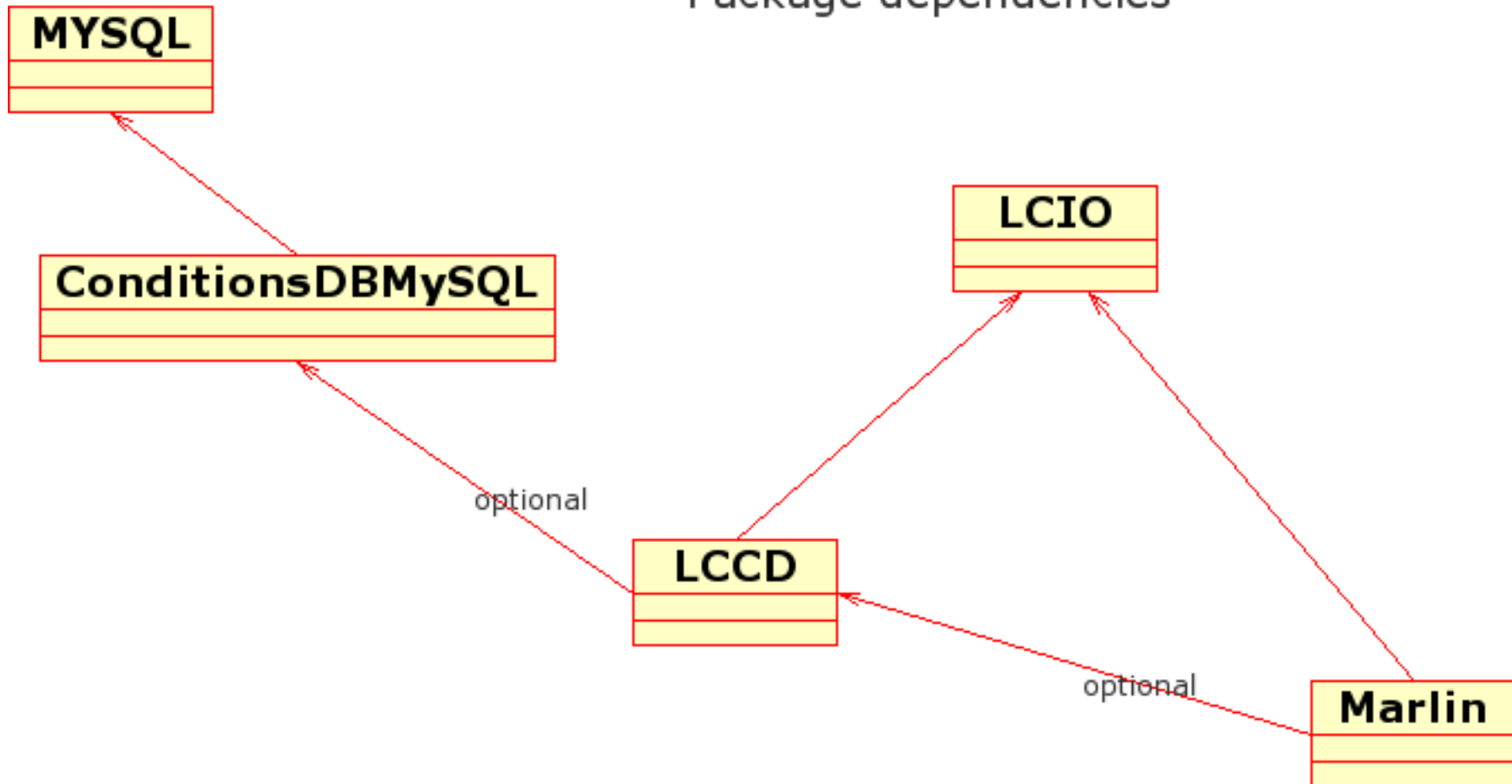
# Introduction

- LCCD – 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



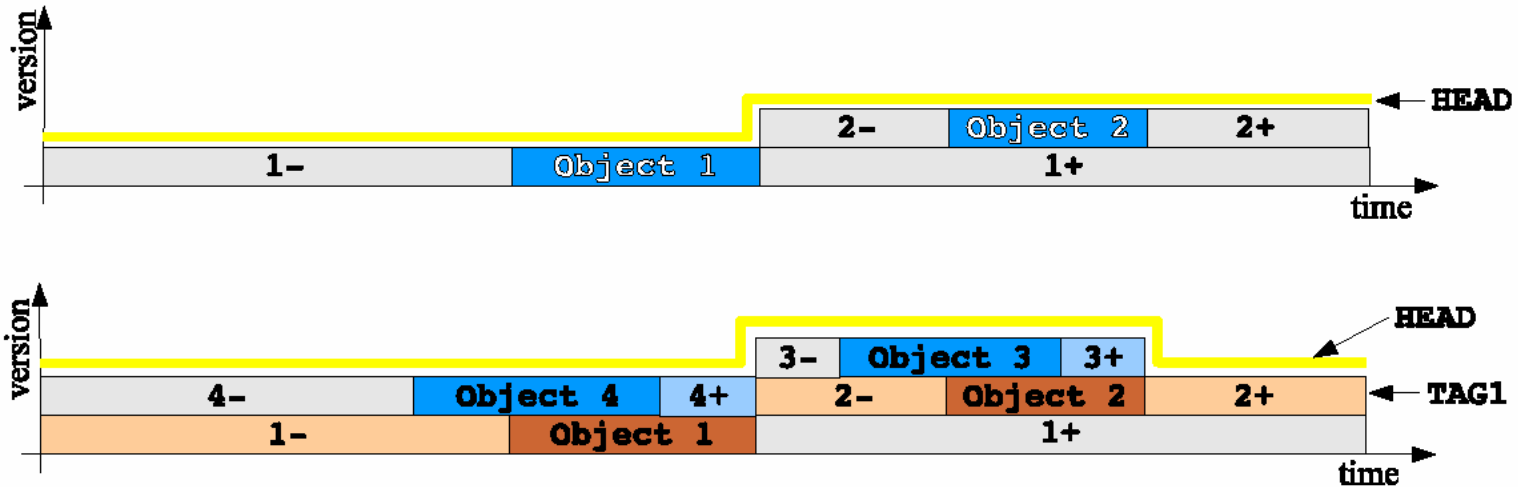


# ConditionsDBMySQL - Overview

- 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: seems to be somewhat stable, but really needs to be tested in 'simulated production' environment before relied on for testbeams !



# CondDBMySQL - Tagging



## Browse objects in HEAD



## Browse objects in tag TAG1

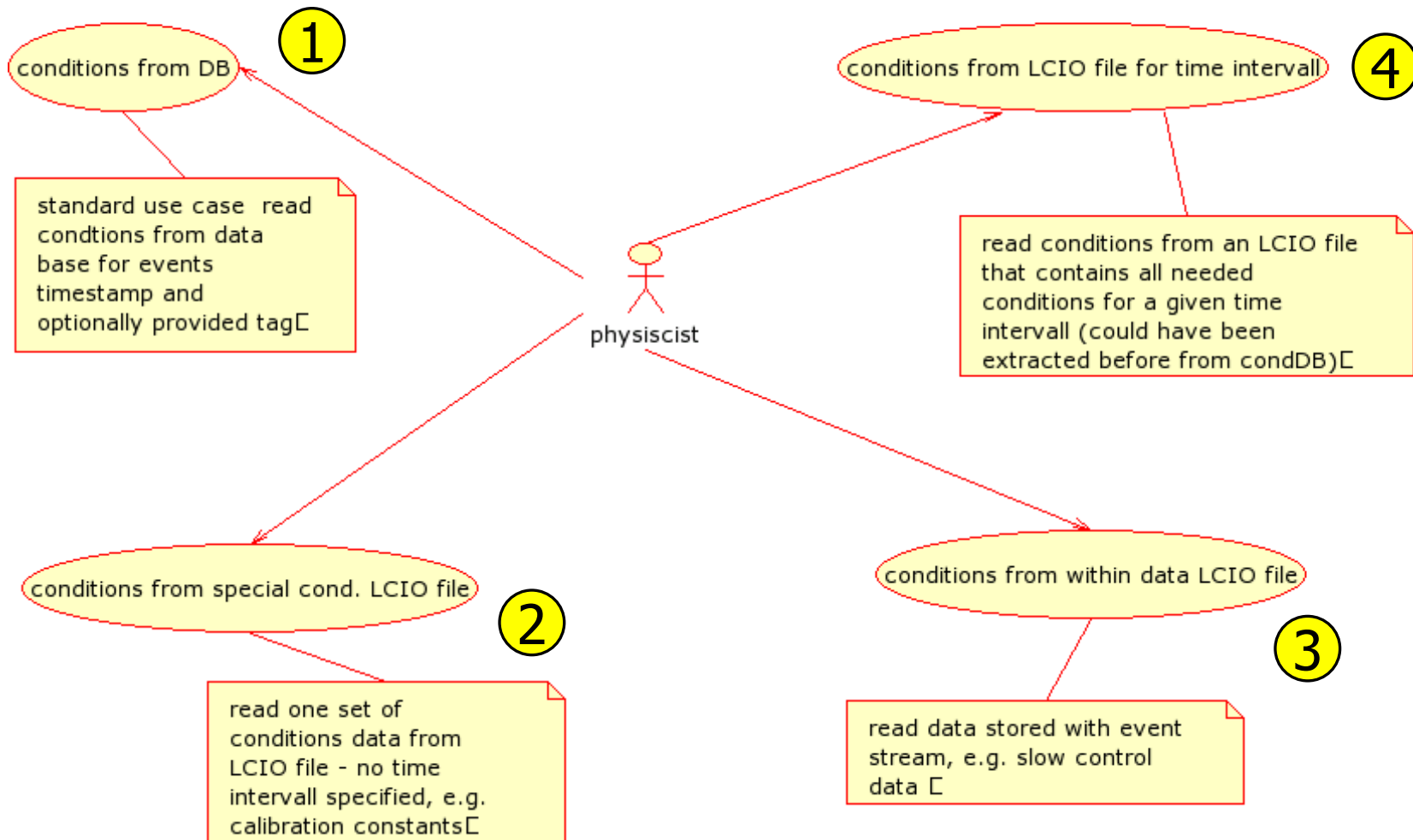


Figure 3: tagging and browsing example in the ConditionsDB mySQL's implementation.



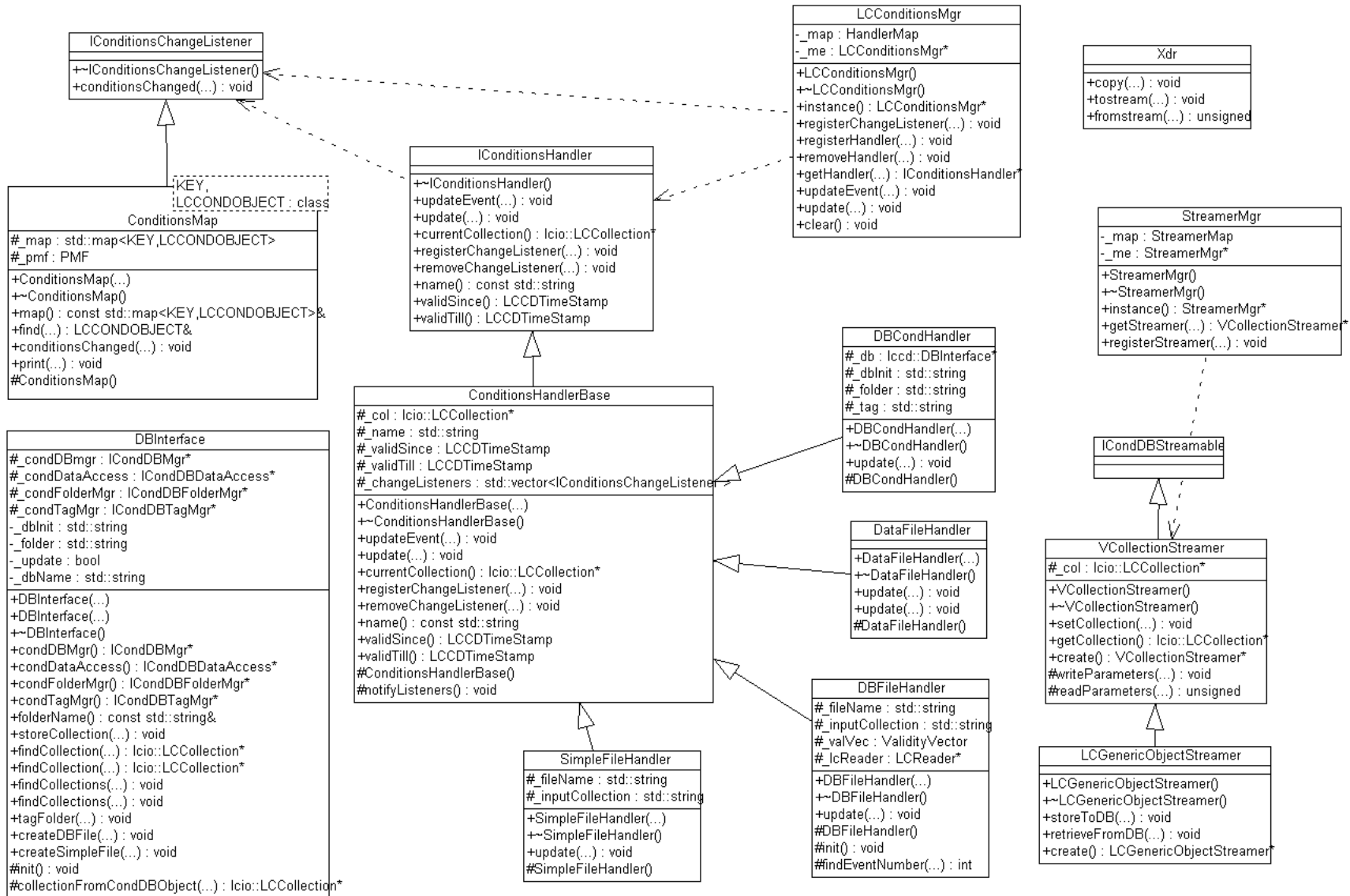
# LCCD Uses Cases (Reading)

## LCCD Use Cases





# LCCD Design







# 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 !
  - different from proposal



# 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
  - time stamps are ignored
  - use for slow control data in LCIO data stream



# 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



# ConditionsMap

---

- template class  
`ConditionsMap<key,CondObject>`
- example:  
`ConditionsMap<int, Pedestal> myMap = new  
ConditionsMap<int, Pedestal>( & Pedestal::getCellID)`
- if registered with `IConditionsHandler` the map will be updated automatically whenever the conditions have changed



# LCCD Status I

- first release v00-01 available via anonymous cvs checkout from Zeuthen repository:
  - export CVS\_RSH=ccvssh
  - export  
CVSROOT=:ext:anonymous@cvssrv.ifh.de:/lccd
  - ccvssh login (prompted for password: should be blank)
  - cvs co -r v00-01 -d v00-01 lccd
- also installed on afs (for DL5):
  - /afs/desy.de/group/it/ilcsoft/lccd/v00-01



# LCCD Status II

---

- v00-01
- has fairly complete functionality
- passes simple tests
- see: example/test code
- ConditionsMap needs further work to make handling easier
- complete API documentation, also at <http://www.desy.de/~gaede/lccd>
- need further tests by users
  - 'production' environment
- also feedback on functionality/interface



# LCCD and LCIO

- LCCD requires current version of LCIO (lccdv00-01)
- changes in LCIO:
  - unique definition of 64bit timestamp 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
- installed at `/afs/desy.de/group/it/ilcsoft/lcio/lccdv00-01`



# Example: LCGenericObject subclass

```
emacs@ZITPCX8315
File Edit Options Buffers Tools C++ Help
#ifndef CalibrationConstant_h
#define CalibrationConstant_h 1

#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.
 */
class CalibrationConstant : public LCFixedObject<NINT,NFLOAT,NDOUBLE> {

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 LCCollection 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 getTypeName() const {
    return "CalibrationConstant" ;
}

const std::string getDataDescription() const {
    return "i:cellID,f:offset,f:gain" ;
}

}; // class
--(Unix)** CalibrationConstant.hh (C++ CVS-1.3 Abbrev)--L56--Top-----
Auto-saving...done
```





# LCCD and Marlin

- Marlin v00-07 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 -----
```
  - not yet: registering ConditionsMaps
  - no example/test code yet
- installed at `/afs/desy.de/group/it/ilcsoft/marlin/v00-07`



# Summary & Outlook

- LCCD is a simple toolkit for retrieving and storing conditions data transparently through
  - LCIO files
  - conditions database (ConditionsDBMySQL)
- v00-01 released
- need testing, testing, testing
- user input very welcome/needed
- **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****need a dedicated person for this !**