Status LCCD

Steve Aplin

EUDET Annual Meeting 2010 – DESY
29th September 2010

Overview

- Default Collections
- Folder Tagging
- LCCD Exceptions

Overview

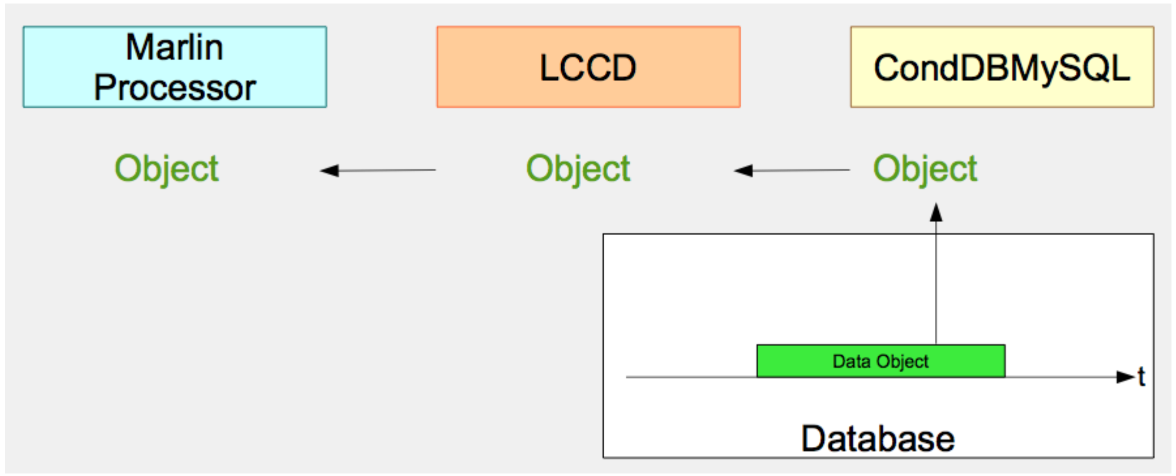
Linear Collider Conditions Data Toolkit

- Supports test-beam efforts by meeting the need to store and retrieve conditions data, e.g. slow control, electronics setup and calibration constants.
- LCCD provides a toolkit that allows conditions data to be stored either in a Database or within an LCIO file in a transparent way.

- Current Release – v01-00
- Available since iLCSoft release – v01-09
- Currently used by Calice and LC-TPC

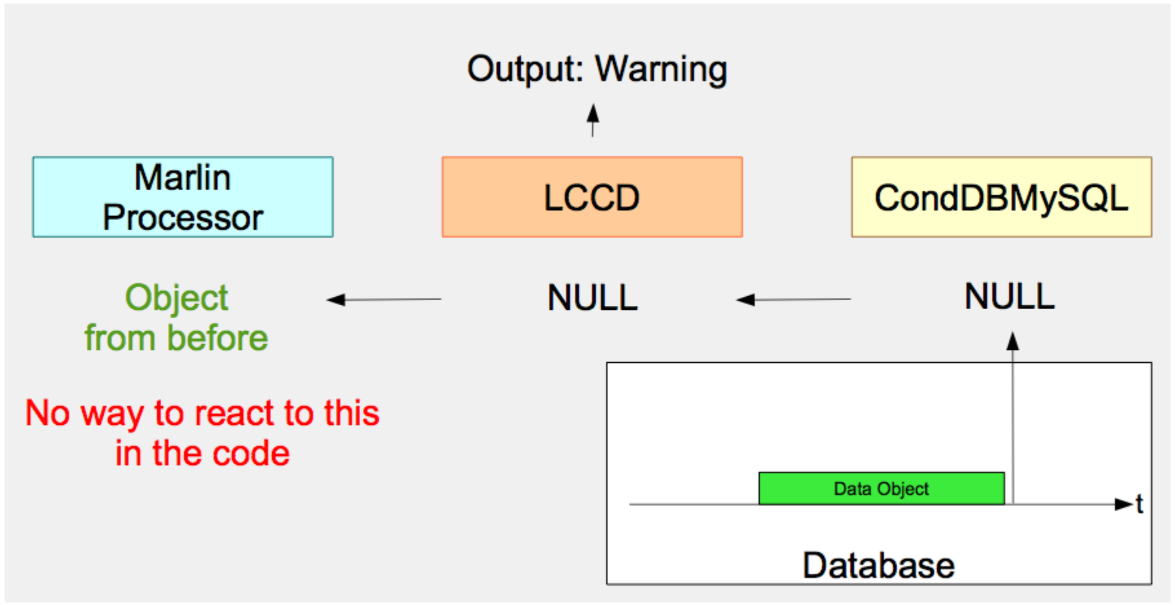
Default Collections

- Originally the LCCD did not foresee valid regions of time where no collection stored



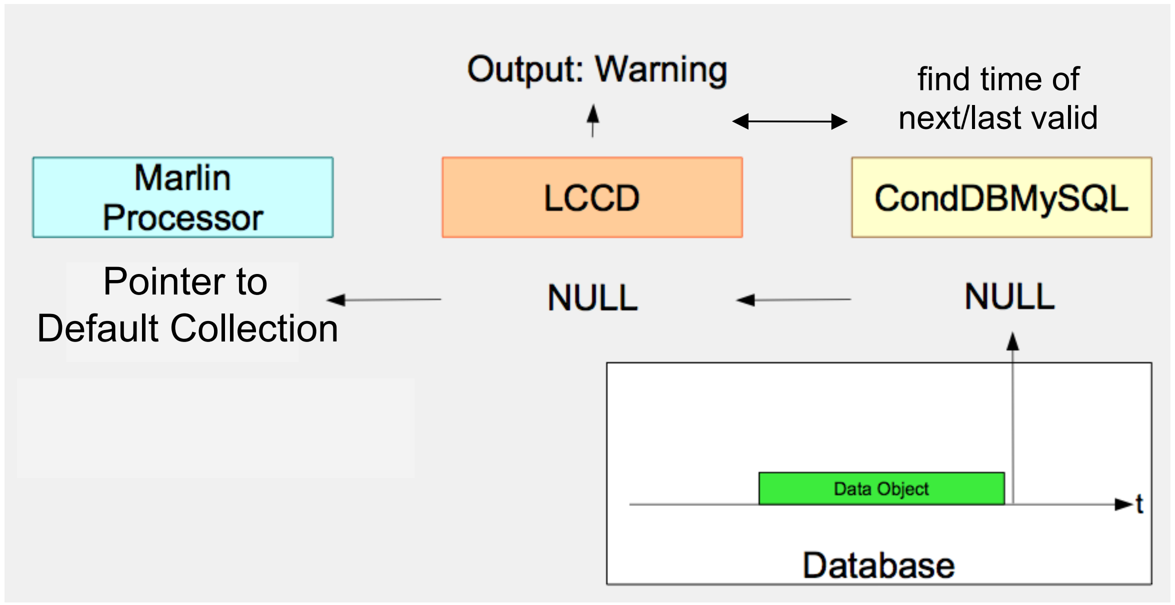
Default Collections

- In the past LCCD was modified to catch the exception for the case of no collections found so as to allow further processing.
- Due to the use of the Listener mechanism, this meant that the Marlin Processors were now blinded to real problems with missing collections.
- As a consequence of missing collections this lead to very high DB load.



Default Collections

- LCCD interface has now been extended to allow users to register a Default Collection which will be returned if no valid collection is found in the Data Base or DBFile.
- IConditionsChangeListener is no longer a pure abstract base class and now contains the two additional call back methods:



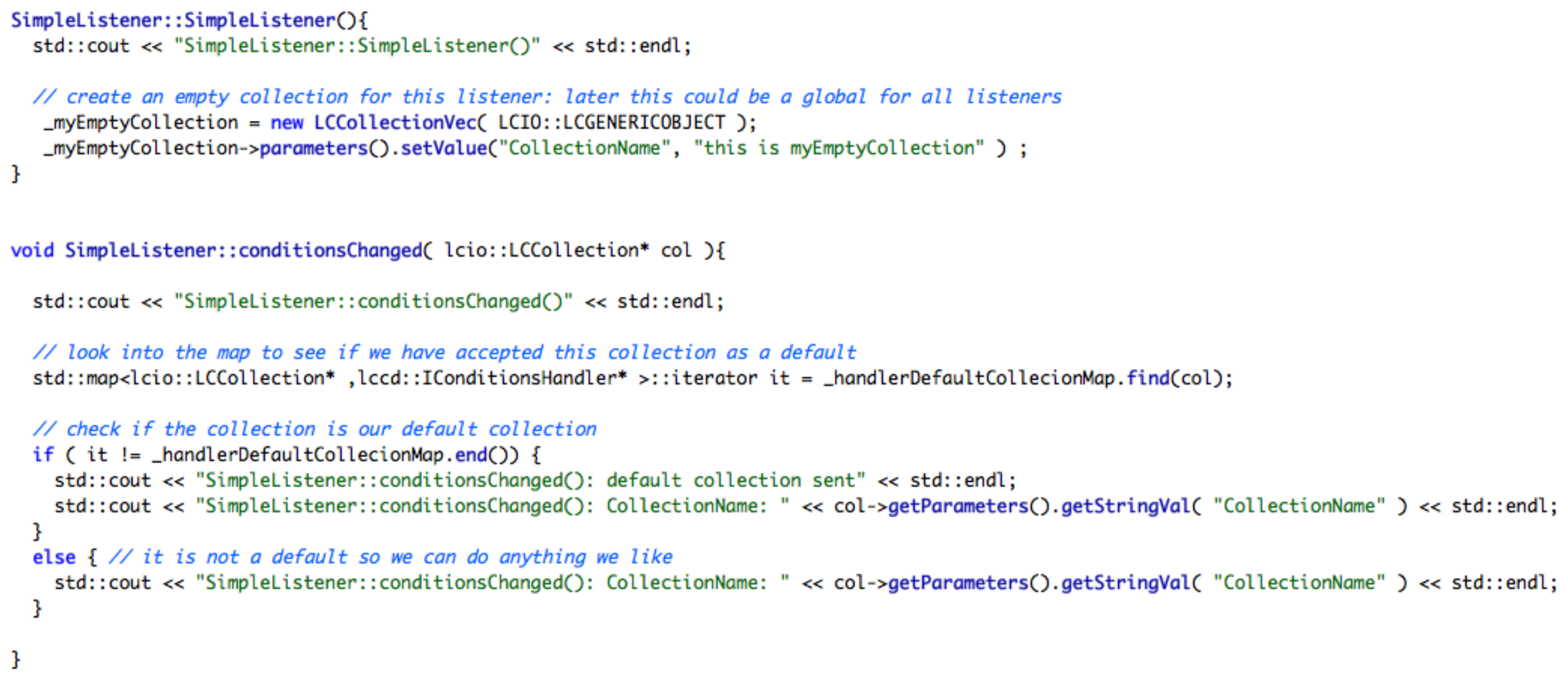
Default Collections

- LCCD interface has now been extended to allow users to register a Default Collection which will be returned if no valid collection is found in the Data Base or DBFile.
- IConditionsChangeListener is no longer a pure abstract base class and now contains the two additional call back methods:
  - virtual void registeredWithHandler(IConditionsHandler* ch) ;
  - virtual void deRegisteredWithHandler(IConditionsHandler* ch) ;
- These are used to maintain a std::list of pointers to the handlers with which the listener has been registered.
- Note: this functionality is only implemented in the DBCondHandler and DBFileHandler classes.
  - Using these methods with SimpleFileHandler and DataFileHandler classes will cause an exception to be thrown.

Default Collections

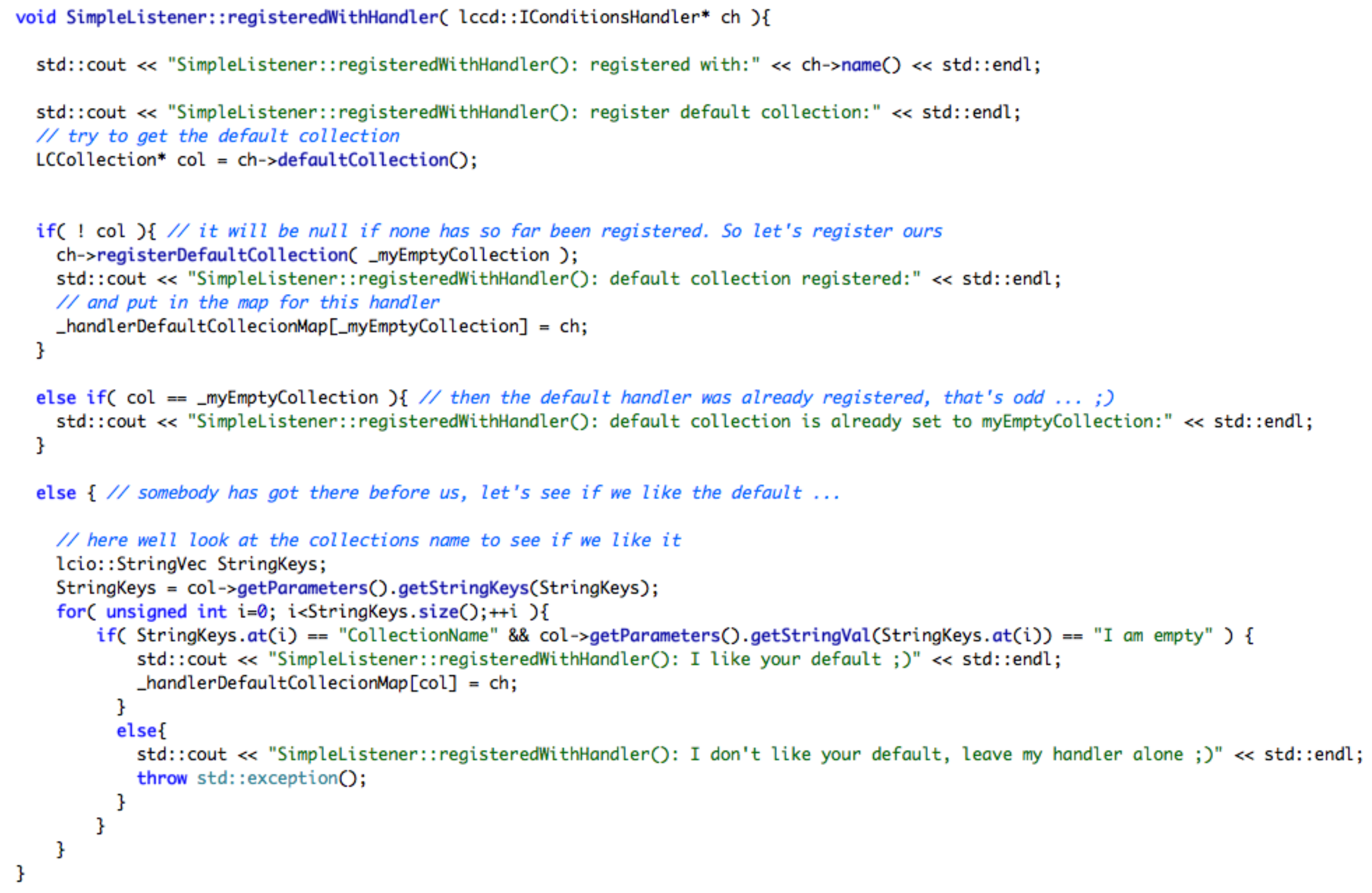
- The ConditionsHandlerBase class has been declared a friend class of IConditionsChangeListener and uses the call-back methods when a listener is registered or de-registered respectively, providing a pointer to itself as the argument.
- The IConditionsHandler has also been extended to provide a pointers to the default collection and the last valid collection.
  - virtual lcio::LCCollection* defaultCollection() = 0 ;
  - virtual lcio::LCCollection* lastValidCollection() = 0;
- The IConditionsHandler has also been extended to check if a given IConditionsChangeListener is register with it
  - virtual bool isChangeListenerRegistered(IConditionsChangeListener* cl) ;
- Note: LCConditionsMgr no longer catches exceptions in the update and updateEvent methods

Default Collections



```
SimpleListener::SimpleListener(){
  std::cout << "SimpleListener::SimpleListener()" << std::endl;

  // create an empty collection for this listener: later this could be a global for all listeners
   _myEmptyCollection = new LCCollectionVec( LCIO::LCGENERICOBJECT );
   _myEmptyCollection->parameters().setValue("CollectionName", "this is myEmptyCollection" ) ;
}


void SimpleListener::conditionsChanged( lcio::LCCollection* col ){

  std::cout << "SimpleListener::conditionsChanged()" << std::endl;

  // look into the map to see if we have accepted this collection as a default
  std::map<lcio::LCCollection* ,lccd::IConditionsHandler* >::iterator it = _handlerDefaultCollecionMap.find(col);

  // check if the collection is our default collection
  if ( it != _handlerDefaultCollecionMap.end()) {
    std::cout << "SimpleListener::conditionsChanged(): default collection sent" << std::endl;
    std::cout << "SimpleListener::conditionsChanged(): CollectionName: " << col->getParameters().getStringVal( "CollectionName" ) << std::endl;
  }
  else { // it is not a default so we can do anything we like
    std::cout << "SimpleListener::conditionsChanged(): CollectionName: " << col->getParameters().getStringVal( "CollectionName" ) << std::endl;
  }

}
```

Default Collections



```
void SimpleListener::registeredWithHandler( lccd::IConditionsHandler* ch ){

  std::cout << "SimpleListener::registeredWithHandler(): registered with:" << ch->name() << std::endl;

  std::cout << "SimpleListener::registeredWithHandler(): register default collection:" << std::endl;
  // try to get the default collection
  LCCollection* col = ch->defaultCollection();


  if( ! col ){ // it will be null if none has so far been registered. So let's register ours
    ch->registerDefaultCollection( _myEmptyCollection );
    std::cout << "SimpleListener::registeredWithHandler(): default collection registered:" << std::endl;
    // and put in the map for this handler
    _handlerDefaultCollecionMap[_myEmptyCollection] = ch;
  }

  else if( col == _myEmptyCollection ){ // then the default handler was already registered, that's odd ... ;)
    std::cout << "SimpleListener::registeredWithHandler(): default collection is already set to myEmptyCollection:" << std::endl;
  }

  else { // somebody has got there before us, let's see if we like the default ...

    // here well look at the collections name to see if we like it
    lcio::StringVec StringKeys;
    StringKeys = col->getParameters().getStringKeys(StringKeys);
    for( unsigned int i=0; i<StringKeys.size();++i ){
        if( StringKeys.at(i) == "CollectionName" && col->getParameters().getStringVal(StringKeys.at(i)) == "I am empty" ) {
          std::cout << "SimpleListener::registeredWithHandler(): I like your default ;)" << std::endl;
          _handlerDefaultCollecionMap[col] = ch;
        }
        else{
          std::cout << "SimpleListener::registeredWithHandler(): I don't like your default, leave my handler alone ;)" << std::endl;
          throw std::exception();
        }
      }
    }
}
```

Folder Tagging

- Previously not possible to tag a folder with a **tag** which has been used to tag another folder.
- To solve this, a recursive search is now done when trying to tag a folder. This checks if the desired **tag** has been already used for the folder in question, or for any of its sub-folders.
- If the **tag** is found in the folder branch by the recursive search an exception is thrown and no tagging is performed.

Ralf Diener

LCCD Exceptions

- Similar to those defined in LCIO
- Part of the lccd namespace

```
class LCCDException : public std::exception

 LCCDException( const std::string& text ) {
   message = "lccd::Exception: " + text ;
 }

 DatabaseException( std::string text ){
   message = "lccd::DatabaseException: " + text ;
 }

 DataNotAvailableException( std::string text ) {
   message = "lccd::DataNotAvailableException: " + text ;
 }

 ReadOnlyException( std::string text ){
   message = "lccd::ReadOnlyException: " + text ;
 }

 InconsistencyException( std::string text ) {
   message = "lccd::InconsistencyException: " + text ;
 }

 MemberNotImplementedException( std::string text ) {
   message = "lccd::MemberNotImplementedException: " + text ;
 }
```

Welcome improvement
in terms of error handling

Ralf Diener

Summary

- Default Collections – available since v01-00
- Folder Tagging – to be available from v01-01 *
- LCCD Exceptions – to be available from v01-01 *
- Next Release v01-01 – within iLCSoft v01-10

* needs new release of CondDBMySQL