Table of Contents

Conditions System layout and How to create conditions objects

Complete:

Layout of the conditions packages in CMSSW

CondCore subsystem

contains centralised conditions modules and services which are responsible for reading data as conditions from POOL databases and writing conditions data into POOL databases as well as managing interval of validity of the data. It includes in following packages:

- **ESSources** pool db ESSource implementation of the conditions service. More details can be found here
- **PluginsSystem** code that handles the dynamic loading of the plugins
- **[subtask]Plugins** instantiates the [subtask] related plugins needed to deliver the [subtask] related objects/Records.
- **DBOutputService** edm::Service module which takes care of writing conditions data into the database. More details can be found here
- **DBCommon** exceptions and classes common to database service implementations
- **IOVService** Definition of the IOV object and IOV interfaces
- **MetaDataService** IOV tag management classes
- **CMS.PopCon** Framework for standard "Populator of Condition"

CondFormats subsystem

holds persistent conditions data object definitions and their associated event setup data record definitions. It includes the following packages

- **DataRecord** holds all definitions of the Records used by the EventSetup
- **[subtask]Objects** holds all [subtask] related persistent object definitions, their dictionary files and EventSetup data registration macros where [subtask] represents a subtask of calibration or alignment.

CondTools subsystem

- **Utilities**
- **CMS.IntegrationTests**
- **Others detector or task specific tools**

Guideline for creating new packages under the Cond system

- Each subdetector or subtask of calibration and/or alignment should require its own packages under CondCore and CondFormats. If you request a [subtask]Objects package in CondFormats, you will need a corresponding [subtask]Plugins package in CondCore as well
- the EventSetup plugins, EventSetup objects, and EventSetup data records are placed in different librarie.
- the directory **CondFormats/DataRecord** is shared by all the detectors and/or tasks

- the granularity of the data record is that each detector has its own sets of records and there should be one record per IOV. If object A and object B are likely to have different IOV, then they should be put in different EventSetup records.
- The detailed directory layout in the Cond system is shown in the following table:

Subsystem	Package	directory	content	owner
CondFormats	[subtask]Objects	interface	persistent object definition [objectname].h	this object
		src	persistent object implementation [objectname].cc(if necessary)	this object
		src	T_EventSetup_[objectName].cc EventSetup data registration macro	this object
		src	classes.h dictionary include file	this task
		src	classes_def.xml dictionary generator selection file	this task
	DataRecord	interface/src	EventSetup record registrations	all tasks
CondCore	[subtask]Plugins	interface	None	
		plugins	plugin.cc EventSetup data proxy registration macros	this task
	Utilities	src	CondDBFetch.cc, CondDBImport.cc, CondFormats.h	all tasks

How to add new conditions data object

The steps described in the following sections are necessary for reading conditions data as object from EventSetup data record. The EventSetup system and PoolDBESSource module combined automatically guarantee the delivered data are valid for the current run or event time. Users do not need to check the validity of the conditions data.

Define the new persistent object in CondFormats/[subtask]Objects package

- add its definition in interface and implementation in src, example

```
class Pedestals {
 public:
   struct Item {
    float m_mean;
    float m_variance;
   };
 Pedestals(){}
 virtual ~Pedestals(){}
 std::vector<Item>  m_pedestals;
};
```

Note

one **must** define a default constructor

- add it in the dictionary selection file src/classes_def.xml, and include file src/classes.h (see later sections)
- register the data object to the event setup system in src/T_EventSetup_[classname].cc, example:

```
#include "CondFormats/Calibration/interface/Pedestals.h"
#include "FWCore/Framework/interface/eventsetupdata_registration_macro.h"
EVENTSETUP_DATA_REG(Pedestals);
```

Use the datarecord code generator "mkrecord" to generate a new EventSetup record

- generate the record as described here, example:

```
#include "CondFormats/DataRecord/interface/PedestalsRcd.h"
#include "FWCore/Framework/interface/eventsetuprecord_registration_macro.h"
EVENTSETUP_RECORD_REG(PedestalsRcd);
```

- put the new data record in **CondFormats/DataRecord**

Tell the plugin system the new data object is associated with which data record

- in **CondCore/[subtask]Plugins/src/plugin.cc** add the data-record pair in the registration macro, example:

```
#include "CondCore/PluginSystem/interface/registration_macros.h"
#include "CondFormats/Calibration/interface/Pedestals.h"
#include "CondFormats/DataRecord/interface/PedestalsRcd.h"
 REGISTER_PLUGIN(PedestalsRcd,Pedestals);
```

How to create a classes_def.xml

The **classes_def.xml** file guides the Seal dictionary generator to select which simbols to be put in the data dictionary. It must contain the names of the classes that you want to store and any of its constituents. In addition, you will likely need to mention any of the class template instantiations. For top level classes is good practice to declare the class version explicitly. For data members that need to be stored as **blob** the mapping shall be declared. classes in `Framework/StdDictionaries` do not need to be declared. Example:

```
<lcgdict>
  <class name="Pedestals" class_version="0"/>
  <class name="Pedestals::Item"/>
  <class name="std::vector<Pedestals::Item>"/>

  <class name="BlobPedestals" class_version="0">
    <field name="m_pedestals" mapping="blob" />
  </class>


 <class name="BitArray<9>"/>

</lcgdict>
```

Detailed description of *Reflex* dictionary generator syntax can be found here.

How to create classes.h

This file should include the header of each top level persistable class(the compiler will include automatically all the embedded classes). Instances of template top level classes shall be instantiated explicity. For example:

```
#include "CondFormats/Common/interface/PayloadWrapper.h"
#include "CondFormats/Calibration/interface/Pedestals.h"
#include "CondFormats/Calibration/interface/BitArray.h"
```

```
namespace {
  struct dictionary {
    BitArray<9> c;
  };

}
```

Build everything

You can now build your object dictionary and event setup plugin libraries. You will see the dictionary generation step. In the CMS.BuildFile of the data directory, you will have to include this flag to the dictionary generator:

```
 <flags GENREFLEX_ARGS="--">

     cd CondFormats/[subtask]Objects
     scramv1 b
     cd CondCore/[subtask]Plugins
     scramv1 b
```

After a successful build, check the EventSetup data proxy for the new data and the record is registered correctly, example

```
 edmPluginDump | grep PedestalsRcd@NewProxy
```

should give non-empty result. If you see an empty result, go back to rebuild [subtask]Plugins library.

Testing

You can perform a minimal test of all the condition objects in your package simply by running the script

```
git cms-addpkg CondFormats/Common
CondFormats/Common/test/runAllPayloadsTest [subtask]Objects
```

it will write (using the default constructor) and read ten objects for each class in **classes_def.xml** file identified by a *class_version* attribute.

Restrictions

Since conditions data persistency is similar to that of event data, the requirements and restrictions on the conditions data packages are similar to those applied to event data (see here.

Dependencies on POOL, CORAL, SEAL, or ROOT (other than rootrflx and packages) are prohibited.

Each persistable class must have a public **default** constructor.

If your package directly throws any exceptions, they must be of the class "cms::Exception", or "cond::Exception". Do not throw a std::exception or define your own exception classes.

Quick links

- **AlCaDB** : https://twiki.cern.ch/twiki/bin/viewauth/CMS/AlCaDB
- **Edm Utilities** : https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookEdmUtilities
- **Tutorial: creating conditions objects** : https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCondObjectsTutorial

Review Status

Editor/Reviewer and date	Comments
ZhenXie - 29 May 2006	page author
JennyWilliams - 03 April 2007	page moved into SWGuide
PalHidas - 14 Feb 2014	git, quick links

Responsible: ZhenXie
Last reviewed by: -- PalHidas - 14 Feb 2014

This topic: CMSPublic > SWGuideCondSystemLayout
Topic revision: r22 - 2015-05-05 - PalHidas