Table of Contents

9.2 ECAL Reconstruction Tutorials

Complete:
Detailed Review status

Goals of this page

This WorkBook topic follows on from the overview material in SWGuideEcalOverview and provides walk-through tutorials to access and manipulate electromagnetic calorimeter objects

Contents

Introduction

This tutorial will walk the user through basic examples showing different functionalities of the Ecal CMSSW objects and software:

- How to read ECAL raw data (in DCC format) and reconstruct up to the level of EcalUncalibratedHit
- How to simulate single electron/photon events, digitize and produce EcalRecHit
- How to read H4 Test Beam raw data files and reconstruct EcalUnlibratedHit
- How to read EcalUncalibrated hits, apply calibration and store only calibrated hits
- How to run a simple user analysis on test beam reconstructed data

Preliminaries

Before working through this tutorial, you should read the introductory chapter SWGuideEcalOverview.

It is also important to know that standard tag collections are created on topo of standard CMSSW releases in order to ease the analysis, in order to give a coherent set of tags to be used. More information on the

ECALTBH4 releases can be found here.

Last ECALTBH4 release is ECALTBH4_0_4_0_pre1 based on CMSSW_1_1_0.

Set up your Environment

First of all you need to know to which CMSSW release the ECALTBH4 is based on. This can be found here. The reference version which is used for this tutotial is ECALTBH4_0_4_0_pre1 and CMSSW_1_1_0. You need to create a release area based on the CMSSW release with the usual scramv1 commands. It can be useful to give an area name different from the usual CMSSW_x_y_z to distinguish from CMSSW releases. One can usually issue from an lxplus machine (NOTE: Feb07: for details of how the changeover to SLC4 affects LXPLUS usage see WorkBookTroubleShooting#LxPlus)

- Log on lxplus

```
ssh lxplus.cern.ch
```

- Create your own release area with a pre-prepared release

```
scramv1 project -n ECALTBH4_x_y_z CMSSW CMSSW_a_b_c
```

where obviously (x,y,z) identifies the ECALTBH4 release and (a,b,c) the CMSSW one.
Then you should do

```
cd ECALTBH4_x_y_z
```

At this point you should use the PackageManagement.pl perl application (available from any CMS account on lxplus/cern machines) to checkout all the packages
which constitute the ECALTBH4 release.

```
PackageManagement.pl --rel ECALTBH4_x_y_z
```

Now you should see from 20/30 packages which are being checkout from the CVS under the src/ directory.
Now you have to build the release using scramv1

```
cd src
 scramv1 b
```

15/30 mins depending on your machine are necessary to complete the build phase.
At this point you can setup your environment

```
eval `scramv1 runtime -csh`
```

To run each example, the required syntax given the configuration file name is:

```
cmsRun <configurationFile>
```

Under the Configuration/EcalTB/data dir of you ECALTBH4 release, you can find a repository of several configuration files which can be used for different tasks, from the simulation of TB data to the reconstruction of H4 TB data from RAW DATA (2004 or 2006).

Read in DCC Raw ecal data and do some reconstruction

In this example you will read in some Ecal Raw Data and reconstruct it up to the level of an UncalibratedRecHit object. This example is applicable only to the RAW DATA files written directly from the ECAL DCC without using the FilterFarm EVB/Storage manager. Files of this kind can be found under CASTOR at CERN in the directories /castor/cern.ch/cms/archive/ecal/h4-SM. These are the output of for example of COSMIC setup in H4. You need to copy one of these file from CASTOR using rfcp, or you can access them in castor using rfio:/castor/.... in the fileNames field of the DAQEcalTBInputService. The following is the content of the config file runOnRawData.cfg:

```
process ProcessOne = {
  source = DAQEcalTBInputService{
            untracked  vstring fileNames = {&#65533;file:H4-000006708-SM5-LASER-STD'}
            untracked int32 maxEvents = 10
            uint32 runNumber = 72999
            # next line is needed only for the older ascii raw data files.
            # when running on the latest raw data in binary format comment the following line
            bool isBinary = false
  }

 # unpack raw data from test beam
  module ecalEBunpacker = EcalDCCUnpackingModule{ }

 # producer of rechits starting from digis. Analytic Fit producer does not require
 # any condition from the DB
  module uncalibHitMaker = EcalAnalFitUncalibRecHitProducer {
   string digiProducer   = "ecalEBunpacker"
   string digiCollection = ""
   string hitCollection  = "EcalEBAnalFitUncalibRecHits"
   untracked int32 nMaxPrintout = 1 }

 module out = PoolOutputModule  { untracked string fileName = "uncalibRechits.root" }

 path p = { ecalEBunpacker, uncalibHitMaker  }
 endpath ep = { out }

}
```

You should save this file with the above name and run it with cmsRun.

Some possible modifications to the config file

- You can use the MessageLogger for your output. To do this, you have to add:

```
 service = MessageLogger
   {
      untracked vstring destinations = {"cerr", "cout"}
      PSet cerr                      = { string threshold = "WARNING" }
      PSet cout                      = { string threshold = "INFO"    }

      untracked vstring fwkJobReports = {"CMS.FrameworkJobReport.xml"}
      vstring categories = { "FwkJob" }

      PSet CMS.FrameworkJobReport.xml = {
                               PSet default = { int32 limit = 0 }
                               PSet FwkJob = { int32 limit = 10000000 }
                                   }
```

For more information about the MessageLogger, see
https://twiki.cern.ch/twiki/bin/view/CMS/MessageLogger

- To use the standard CMS weight algorithm instead of analytic fit:

```
module ecaluncalibrechit = EcalWeightUncalibRecHitProducer {
   string digiProducer   = "ecaldigi"
   string EBdigiCollection = ""
   string EEdigiCollection = ""
   string EBhitCollection  = "EcalUncalibRecHitsEB"
   string EEhitCollection  = ""
```

- ♦ To use weight algorithm you need also a source of the weights from the EventSetup. To use hardcoded/default (usually not optimized for your needs)

```
 # Get hardcoded conditions
  es_source = EcalTrivialConditionRetriever {
    untracked double adcToGeVEBConstant = 0.035
}
```

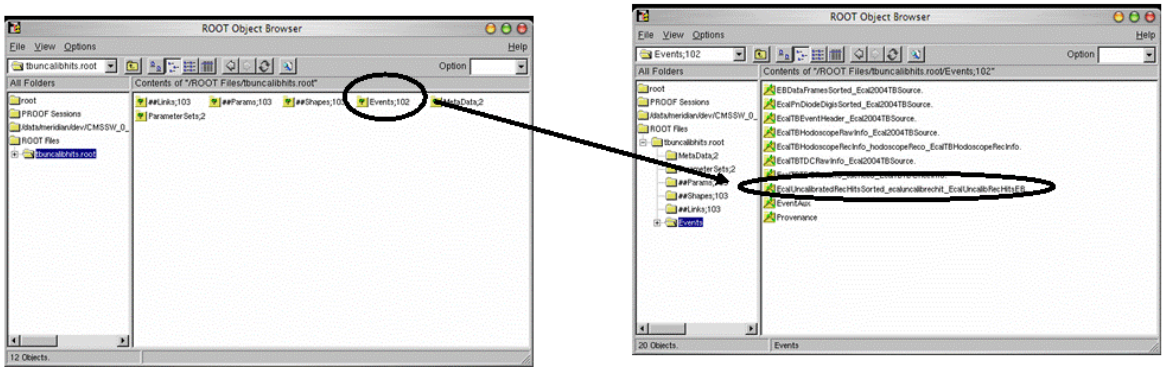
- ♦ In the above code you can customize the setting by specifying parameters that are different to the default ones.

Checking the output

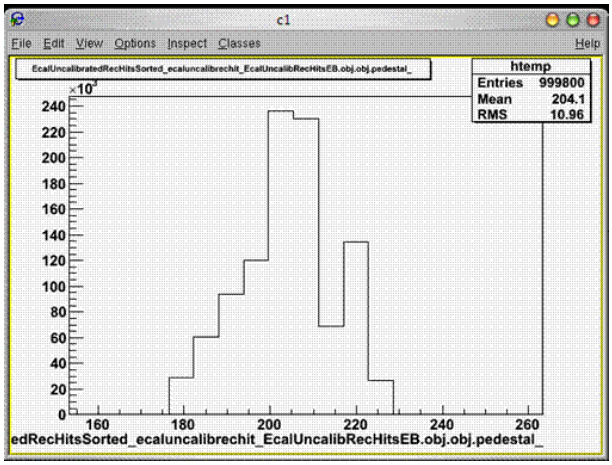
With the new CMS Event Data Model (EDM) it is possible to do a quick check on the produced POOL file reading it directly with ROOT. To do this set up your environment and launch ROOT and then load the FWLite library within ROOT with:

```
gSystem->Load("libPhysicsToolsFWLite");
AutoLibraryLoader::enable();
```

Then you can launch the TBrowser and load the `uncalibRechits.root` file. The TBrowser stages should look like this:



Where finally the Pedestals plot directly accessed from within the TBrowser should look like:



To make a deeper test, you need to write a ROOT macro better interrogate the POOL file.

Simulate single electron/photon events, digitize and produce EcalRecHit objects

In this example you will simulate some single electron/photon events and run the digitization process.

Under the Configuration/EcalTB/data dir you will find

`reco_application_tbsim_DetSim-Digi.cfg`

Simply run this file with cmsRun.

Read H4 Test beam Raw Data h4ana data and produce uncalibrated Hits

The required config files are

`reco_application_2004rawdata_localreco.cfg`

for the 2004 H4 raw data,

`reco_application_2006rawdata_localreco.cfg`

for the 2006 raw data, which can always be found in the dir Configuration/EcalTB/data. 2004 raw data are the so called rrf files, which can be read with CMSSW using a special InputSource. (RRF Files are available in castor at
`/castor/cern.ch/cms/testbeam/tbh4/2004RawRoot`)

The 2006 RAW data, written directly in CMSSW POOL format, are instead available at /castor/cern.ch/cms/archive/ecal/h4tb.pool-SM[SM number].

Some code which is worth studying in this config file is:

```
process uncalibRecHitProd  = {

 # initialize  MessageLogger
 include "FWCore/MessageLogger/data/MessageLogger.cfi"
```

```
 # input source for 2004
 source = Ecal2004TBSource {
     untracked vstring fileNames = { 'file:/u1/meridian/data/h4/2004/ecs73276' }
     untracked int32 maxEvents = 1000
 }

 # Read 2004 offline DB (v2 for 0_8_x)
 include "Configuration/EcalTB/data/readConfiguration2004_v2.cff"

 # Reconstruction for 2004 rawData
 include "Configuration/EcalTB/data/localReco2004_rawData.cff"

  module out = PoolOutputModule
   {
     untracked string fileName = 'tbuncalibhits.root'
   }

  path p = {  getCond, localReco2004_rawData }

  endpath ep = { out }
}
```

Download the full version of this file and run it with cmsRun.

Files already processed in this way for the 2006 H4 test beam are already available at /castor/cern.ch/cms/archive/h4tb.pool-cmssw-SM[SM number]

Read uncalibrated hits, apply calibration and store calibrated hits

The required config file is

```
reco_application_2006_calibrate.cfg
```

Some code which is worth studying in this config file is:

```
process recHitProd  = {

  # initialize  MessageLogger
  include "FWCore/MessageLogger/data/MessageLogger.cfi"

  # input
  source = PoolSource {
     untracked vstring fileNames = {'file:/u1/meridian/data/h4/2006/h4b.00013247.A.0.0.root'}
     untracked int32 maxEvents = -1
     untracked uint32 debugVebosity = 1
     untracked bool   debugFlag     = false
  }

  # Read Intercalibrations from offline DB (v2 for 0_8_x)
  include "Configuration/EcalTB/data/readIntercalibrationsFromAscii2006_v0.cff"
  # rechit producer
  module ecal2006TBRecHit = EcalRecHitProducer {
    string uncalibRecHitProducer   = "ecal2006TBWeightUncalibRecHit"
    string EBuncalibRecHitCollection = "EcalUncalibRecHitsEB"
    string EEuncalibRecHitCollection = ""
    string EBrechitCollection        = "EcalRecHitsEB"
    string EErechitCollection        = ""
    }

  #output module dropping everything apart calibrated containers
```

```
  module out = PoolOutputModule
   {
        untracked vstring outputCommands =
          {
            "drop *",
            "keep EcalRecHitsSorted_*_*_*",
            "keep EcalTBHodoscopeRecInfo_*_*_*",
            "keep EcalTBEventHeader_*_*_*",
            "keep EcalTBTDCRecInfo_*_*_*"
          }
     untracked string fileName = 'tbhits.root'
   }

  #Use this to get calibrations from the DB
  path p = { ecal2006TBRecHit }

  endpath ep = { out }
}
```

You need a calibration file for the data of your specific SM. These files can be found at /afs/cern.ch/cms/ECAL/testbeam/pedestal/2006/CALIBRATIONS.

Run a simple user analysis on Test beam (TB) reconstructed data

The required config file is reco_application_2006_simpleTBanalysis.cfg.

Highlights of code in this config file include:

```
process simpleAnalysis = {

 # initialize  MessageLogger
 include "FWCore/MessageLogger/data/MessageLogger.cfi"

 ###################################################################
 # input
  source = PoolSource {

        untracked vstring fileNames = {'rfio:/castor/cern.ch/cms/archive/ecal/
                                                h4tb.pool-cmssw/h4b.00017056.A.0.0.root'}
#       untracked vstring fileNames = {'file:hits.root'}
     untracked int32 maxEvents = 500
  }
 ###################################################################
  # analysis module for HV Scan
  module simpleTBanalysis = EcalSimpleTBAnalyzer
  {
   string digiProducer   = "ecalTBunpack"
   string digiCollection = ""

   string hitProducer   = "ecal2006TBWeightUncalibRecHit"
   string hitCollection = "EcalUncalibRecHitsEB"

   string hodoRecInfoProducer   = "ecal2006TBHodoscopeReconstructor"
   string hodoRecInfoCollection = "EcalTBHodoscopeRecInfo"

   string tdcRecInfoProducer   = "ecal2006TBTDCReconstructor"
   string tdcRecInfoCollection = "EcalTBTDCRecInfo"

   string eventHeaderProducer   = "ecalTBunpack"
   string eventHeaderCollection = ""

   untracked string rootfile = "ecalSimpleTBAnalysis.root"
```

```
  }

  path p = { simpleTBanalysis  }
}
```

Analysis example: Let's analyze together the code in.cc

The code in the analysis module features a `beginJob` and an `endJob` module that are called at the beginning and end of each job, and which therefore contain constructors and destructors for objects which are required in the analysis job. The method **`analyze(...)`** is the main method which is called at every event, and is therefore connected with instances of `Event` and `EventSetup`.

The main code from the EcalSimpleTBAnalyzer which is of interest to us is:

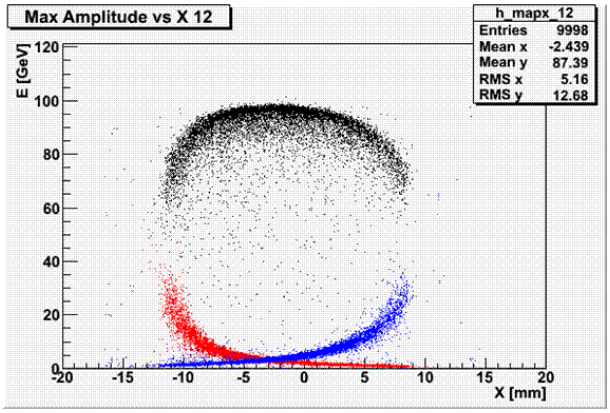
```
class EcalSimpleTBAnalyzer : public edm::EDAnalyzer {
   public:
      explicit EcalSimpleTBAnalyzer( const edm::ParameterSet& );
      ~EcalSimpleTBAnalyzer();


      virtual void analyze( const edm::Event&, const edm::EventSetup& );
      virtual void beginJob(edm::EventSetup const&);
      virtual void endJob();

void EcalSimpleTBAnalyzer::analyze( const edm::Event& iEvent, const edm::EventSetup& iSetup ) {
// fetch the calibrated RecHit collection
   Handle<EBRecHitCollection> phits;
   const EBRecHitCollection* hits=0;
   try {
     iEvent.getByLabel( hitProducer_, hitCollection_,phits);
     hits = phits.product(); // get a ptr to the product
   } catch ( std::exception& ex ) {
     std::cerr << "Error! can't get the product " << hitCollection_.c_str() << std::endl;
   }
//fetch the hodoscope reconstructed information
   Handle<EcalTBHodoscopeRecInfo> pHodo;
   const EcalTBHodoscopeRecInfo* recHodo=0;
   try {
     iEvent.getByLabel( hodoRecInfoProducer_, hodoRecInfoCollection_, pHodo);
     recHodo = pHodo.product(); // get a ptr to the product
   } catch ( std::exception& ex ) {
     std::cerr << "Error! can't get the product " << hitCollection_.c_str() << std::endl;
   }
//fetch the tdc reconstructed information
   Handle<EcalTBTDCRecInfo> pTDC;
   const EcalTBTDCRecInfo* recTDC=0;
   try {
     iEvent.getByLabel( tdcRecInfoProducer_, tdcRecInfoCollection_, pTDC);
     recTDC = pTDC.product(); // get a ptr to the product
   } catch ( std::exception& ex ) {
     std::cerr << "Error! can't get the product " << hitCollection_.c_str() << std::endl;
   }

//fetch the tb event header
 Handle<EcalTBEventHeader> pEventHeader;
   const EcalTBEventHeader* evtHeader=0;
   try {
     iEvent.getByLabel( eventHeaderProducer_ , pEventHeader );
     evtHeader = pEventHeader.product(); // get a ptr to the product }
     catch ( std::exception& ex ) {
     std::cerr << "Error! can't get the product " << hitCollection_.c_str() << std::endl;
   }
 //Create a matrix of 5x5 EBDetId
```

```
 EBDetId maxHitId(1,704,EBDetId::SMCRYSTALMODE);
 EBDetId Xtals5x5[25];
   for (unsigned int icry=0;icry<25;icry++)
     { unsigned int row = icry / 5;
       unsigned int column= icry %5;
       try { Xtals5x5[icry]=EBDetId(maxHitId.ieta()+column-2,maxHitId.iphi()+row-2,
                                                    EBDetId::ETAPHIMODE); }
       catch ( std::runtime_error &e )
         { std::cout << "Cannot construct 5x5 matrix around EBDetId " << maxHitId << std::endl;
           return; } }
.......
//Read amplitudes crystal by crystal
for (unsigned int icry=0;icry<25;icry++) {
       amplitude[icry]=(hits->find(Xtals5x5[icry]))->energy();
       amplitude5x5 += amplitude[icry];
.......
//Reconstructed energy vs tdc offset
if (recTDC)
     h_ampltdc->Fill(recTDC->offset(),amplitude[12]);
........
//Reconstructed energy vs hodoscope X&Y
   if (recHodo) {
       float x=recHodo->posX();
       float y=recHodo->posY();
       float xslope=recHodo->slopeX();
       float yslope=recHodo->slopeY();
       float xqual=recHodo->qualX();
       float yqual=recHodo->qualY();
........
```

Some plots that can be produced using this analysis file

Amplitude plot

The first plot is of response amplitude vs distance (from source?)

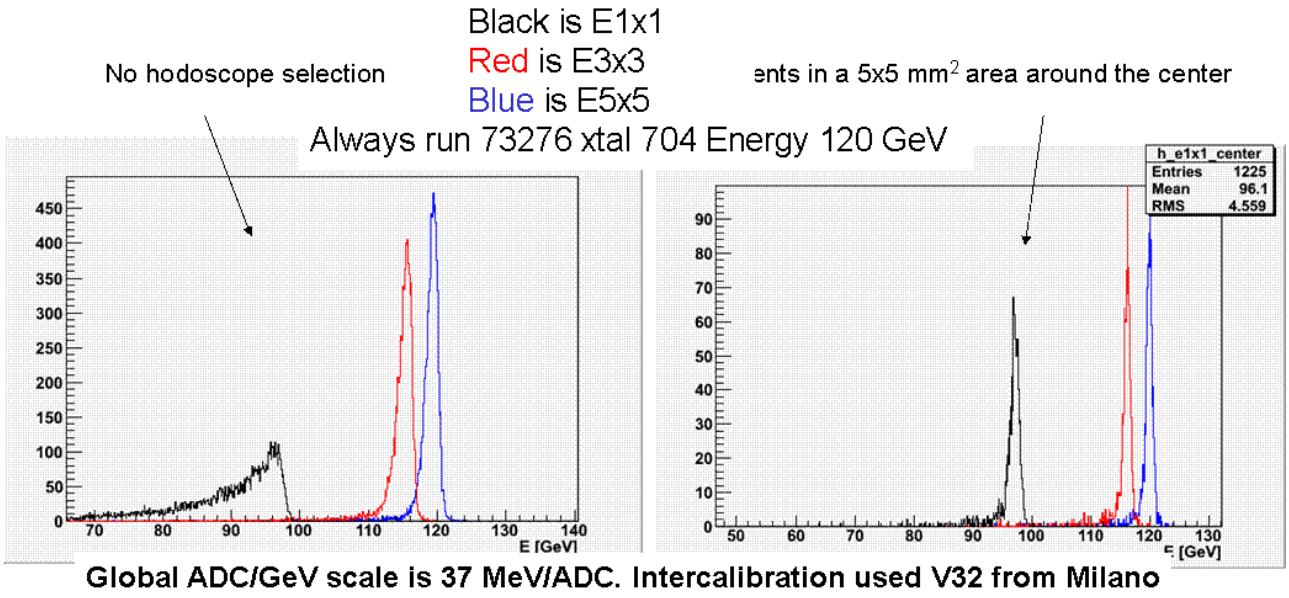


Data for plot:

```
Run 73276
Xtal in Beam 704
Beam Energy 120 GeV
Last version of the weights for 2004 TB analysis from A.Zabi
Intercalibration from h4rootdb V32
Condition DB used to produce this results is:
sqlite_file:/afs/cern.ch/cms/ECAL/testbeam/pedestal/2004/ecal2004condDB.db

Black is xtal 704 response vs X
```

```
Red is xtal 684 response vs X
Blue is xtal 724 response vs X
```

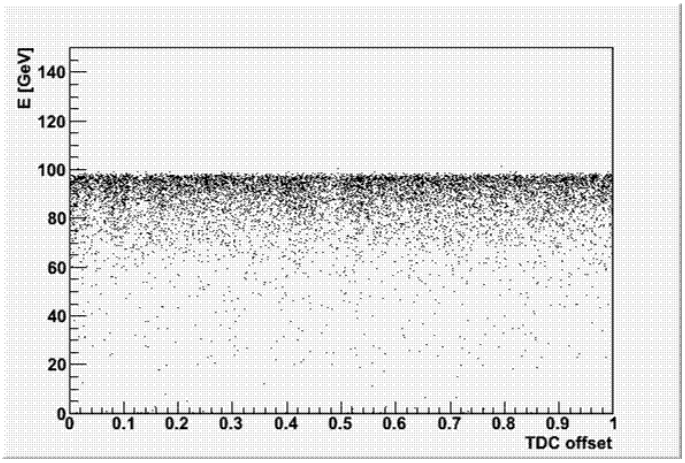
Crystal cluster response plot



Global ADC/GeV scale is 37 MeV/ADC. Intercalibration used V32 from Milano

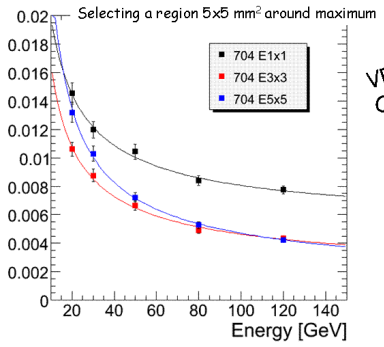
Response vs TDC offset

This plot shows crystal 704's response vs the time-to-digital converter (TDC) offset. Optimized 3+5 weights (from Xabi) are used, and no bias is observed.



Resolutions

The following image shows the resolution of the crystal cluster plots from above. The results shown are preliminary only and shown here only to represent what can easily be extracted from the Ecal software.



VERY PRELIMINARY
Only to show functionality

Energy	RUN
20	73504
30	73474
50	73416
80	73304
120	73276

Information Sources

- Ecal CMSSW tutorial (source of material in this tutorial)
- Ecal offline software Twiki page

Review status

Original Author: Paolo Meridiani, INFN Roma 1

Reviewer/Editor and Date (copy from screen)	Comments
JennyWilliams - 27 Jun 2006	created page from Ecal CMSSW tutorial
PaoloMeridiani - 01 Dec 2006	Updated for using ECALTBH4 releases
JennyWilliams - 14 Dec 2006	General tidying
JennyWilliams - 12 Feb 2007	added link to lxplus changeover info to troubleshooting topic

Responsible: PaoloMeridiani
Last reviewed by: ChrisSeez - 13 Dec 2006

This topic: CMSPublic > SWGuideEcalRecoTutorials
Topic revision: r19 - 2013-01-20 - ColinBaus