CMS Software

Content

- Introduction
- Event data model
- Processors
- Event Setup
- Services
- Examples
- A few utilities
- Concrete example with a detector
- How to write bad codes

Sunanda Banerjee

Pre-History

- CMS detector design started around 1992 and soon there was a need of software to carry out simulation studies for an optimum design of the detector
- Utilize the available technology to build the first level of software (CMSIM I): 1992
  - Use FORTRAN77 as the language
  - Use BOSS as memory management package as well as IO
  - Use GEANT3 as the basic simulation toolkit
  - First level of simulation results to make the Letter of Intent
- Usage of two memory management packages (for CMS software and for GEANT3) seems to be very inappropriate. Get rid of BOSS with ZEBRA (CMSIM II): 1995
  - Build a special language to describe geometry (first decoupling of code and constants)
  - Start building reconstruction code
  - C++ is just becoming popular to High Energy Physicists
  - Part of reconstruction code (for calorimeters) written in C++
  - Write the technical proposal and all technical design reports (upto Tracker) with the results obtained using CMSIM II

History

- Stronger need to move to C++. There was no obvious replacement for GEANT3 available – so move reconstruction code starting from digitization to C++
- Idea of reconstruction on demand –
  - let the raw data stay as it is
  - User asks for a track
  - Track reconstruction code is initiated – it needs tracker hits
  - Tracker rechit reconstruction code is initiated

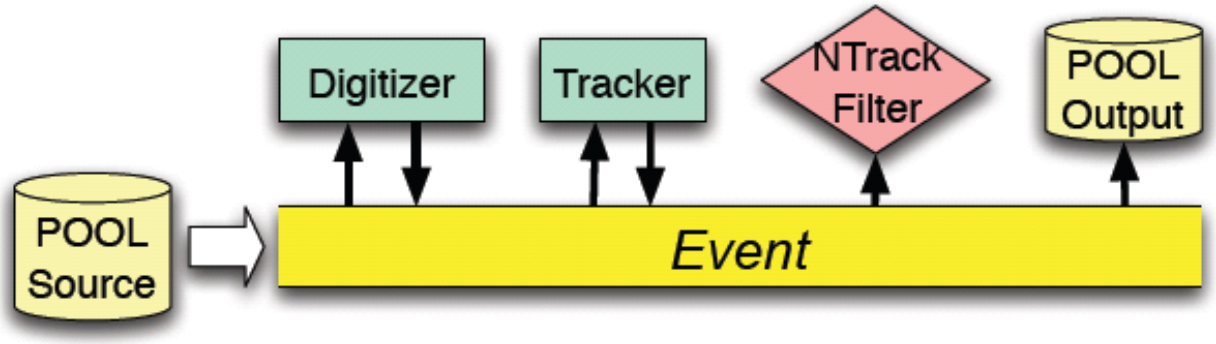
    .....
  - Ultimately reaches raw data
- Can be achieved using call back facility, plugins and shared library
- Use object data base technology for IO and the basic architecture is designad on top of ODBMS (Objectivity DB) → ORCA
- Alternative to GEANT3 came as a C++ toolkit: GEANT4. Replace the FORTRAN based simulation package with a C++ version → OSCAR I
- Integrate simulation and reconstruction packages → COBRA, OSCAR II, ORCA
  - Write the physics and trigger TDR's using this combination

Transition to Present

- The IO package of COBRA was commercial and supported by a small company. ODBMG did not gain popularity
- Usage of Objectivity DB within BaBar was a disaster
- ORCA timeline was rather slow
  - Physics TDR time line slipped
  - Output of ORCA needs to be translated to be viewed by the new analysis package (ROOT)
  - All other visualization and analysis packages could not compete with ROOT
- A new framework was suggested (CMSSW)
  - Use ROOT IO package which is the biggest strong point of ROOT
  - ROOT is extensively modified to store and retrieve user defined data (Reflex)
  - Simplify data flow and forget about complications like call back
  - Basic requirements of a HEP software was strongly debated and the transition was more a revolution rather than an evolution
- Use of Python and XML is becoming popular
  - Replace CMS's own scripting language to Python for job control
  - Use XML's to describe the detector, software building, database transient objects

Data Flow

- All event data are stored in a single container – the “Event”
- Algorithms are implemented in component “modules”
  - Modules communicate only through the event
  - Execution is scheduled explicitly
  - Scheduling is done in the job configuration
  - Required modules are dynamically loaded at the beginning of the job

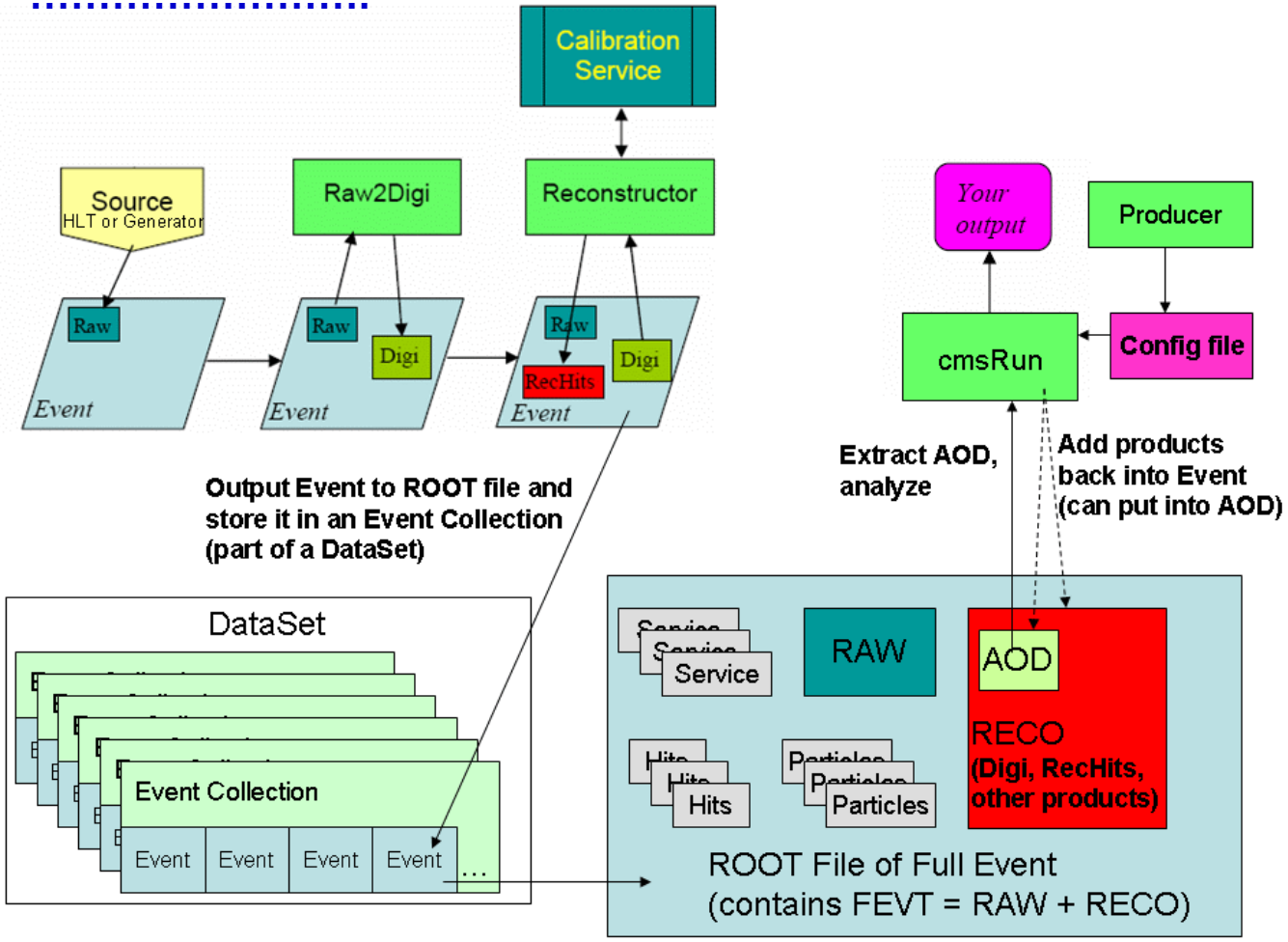


Compare with COBRA/ORCA:

- Objects obtained through RecCollection<T>
- Any operation on the collection triggers reconstruction on-demand

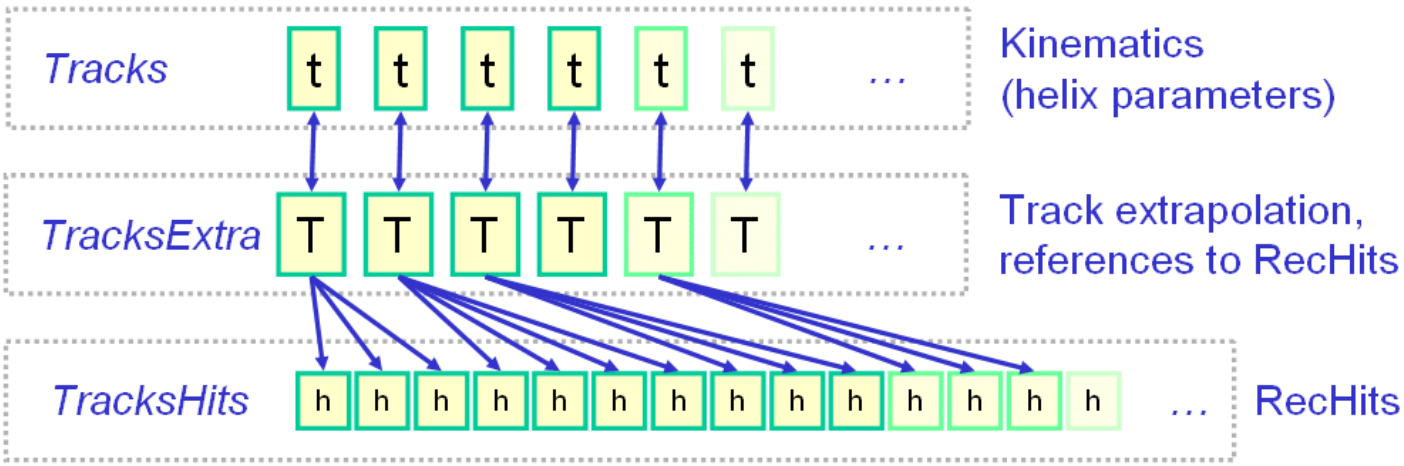
Event & FrameWork

- Event is a collection of containers each containing products of a given type.
- There are several levels of completeness of the Event
  - FEVT: Full event
  - AOD: Analysis object data
  - .....



Event Data

- Different data layers exist using different data formats and an application can use any layer(s).
- Branches map one to one with event data objects and are loaded or dropped on demand,
- Event data are identified by
  - C++ class type (like PCaloHitContainer)
  - Label assigned to the module that created the data (g4SimHits)
  - Product instance label assigned to the object within the module (HcalHits)
  - Process name that creates the data



Layer 0: The RAW data

- All detectors use the same class to hold the raw data (FEDRawData), which is basically just an array of 64-bit unsigned integers as they come off the detector electronics into the DAQ.
- The data for many channels is bundled together in the RAW data.
- In addition there are results from the Triggers
  - The Level 1 trigger results
  - The results of the HLT selections (trigger bits)
  - Some of the higher-level objects created during HLT processing
- The Raw data is converted into digis in the Raw2Digi (or unpacking) process.
- It can also be used by detector experts to monitor the happenings of the electronics – header and trailer words which are not copied into the digis.

Layer 0* : SimHits

- The simulation process begins with an event generator followed by the Geant4-based detector simulation. The output of the simulation is a set of simulated energy deposits (with deposit times) in detector channels. This may contain additional information like time, position of the hit in the detector (in local coordinates), …
- These informations are stored in the event as SimHits. The SimHits are labeled by DetIds (a packed integer identifying the detector element hit)
- SimHits are converted into digis in the electronics simulation or “digitization process”.
- In CMS, the digitization process mixes “minimum-bias” SimHits from a file with the simulated minbias event to simulate *pileup*.
- After the mixing module and digitization, the data content in real and Monte Carlo data are identical. There may be some additional data to trace back to generator level information

Example of a Transient Object

```
class CaloHit {
public:
  CaloHit(int deti, int layi, double ei, double etai, double phii, double timi,uint32_t idi=0);
  CaloHit();
  CaloHit(const CaloHit&);
  virtual ~CaloHit();
  int             det()     const {return deth;}
  int             layer()   const {return layerh;}
  double          e()       const {return eh;}
  double          eta()     const {return etah;}
  double          phi()     const {return phih;}
  double          t()       const {return timeh;}
  uint32_t        id()      const {return idh;}
  void            setId(const uint32_t idi) {idh = idi;}
  bool operator<( const CaloHit& hit) const;
private:
   int          deth, layerh;
  double         eh, etah, phih, timeh;
  uint32_t      idh;
};
std::ostream& operator<<(std::ostream&, const CaloHit&);
```

The Corresponding Persistent Class

```
class PCaloHit {
public:
   PCaloHit(float e = 0., float t = 0., int i = 0, float emFraction = 1.,
                    uint16_t d = 0) : myEnergy(e), myEMFraction(emFraction), myTime(t),    myItra(i), myDepth(d) { }
  static const char *name() { return "Hit"; }
  const char * getName() const { return name (); }
  double energy()    const { return myEnergy; }
  double energyEM()  const { return myEMFraction*myEnergy; }
  double energyHad() const { return (1.-myEMFraction)*myEnergy; }
  double time() const { return myTime; }
  int geantTrackId() const { return myItra; }
  unsigned int  id() const { return detId; }
  uint16_t depth() const { return myDepth; }
  void setEventId(EncodedEventId e) { theEventId = e; }
  EncodedEventId eventId() const {return theEventId;}
  void setTime(float t) {myTime=t;}
  bool operator<(const PCaloHit &d) const { return myEnergy < d.myEnergy; }
protected:
   float myEnergy, myEMFraction, myTime;
   int   myItra;
   unsigned int detId;
   uint16_t myDepth;
   EncodedEventId  theEventId;};
#include<iosfwd>
std::ostream &operator<<(std::ostream &, const PCaloHit &);
```

Layer 1 : Digis

- Digis are the per-channel data from the detector. They are labeled by DetId, so the RAW/SIM data must be correctly labeled in the unpacking step.
  - In the calorimeters a Digi is a set of ADC readings for that event (3 readings before the triggered bunch-crossing and 7 readings containing the maximum pulse)
- Digis show the pulse shape of the detector and are used to reconstruct the energy and time of the hit in the calorimeter or the position, dE/dx and time in the tracking devices.

```
class EBDataFrame {
 public:
  typedef EBDetId key_type; ///< For the sorted collection
  EBDataFrame(); // for persistence
  explicit EBDataFrame(const EBDetId& id);
  const EBDetId& id() const { return id_; }
  int size() const { return size_; }
  const EcalMGPASample& operator[](int i) const { return data_[i]; }
  const EcalMGPASample& sample(int i) const { return data_[i]; }
  void setSize(int size);
  void setSample(int i, const EcalMGPASample& sam) { data_[i]=sam; }
  static const int MAXSAMPLES = 10;
 private:
  EBDetId id_;
  int size_;
  std::vector<EcalMGPASample> data_;
};
```

```
class EcalMGPASample {
 public:
  EcalMGPASample() { theSample=0; }
  EcalMGPASample(uint16_t data) { theSample=data; }
  EcalMGPASample(int adc, int gainId);

  /// get the raw word
  uint16_t raw() const { return theSample; }
  /// get the ADC sample (12 bits)
  int adc() const { return theSample&0xFFF; }
  /// get the gainId (2 bits)
  int gainId() const { return (theSample>>12)&0x3; }
  /// for streaming
  uint16_t operator()() { return theSample; }

 private:
  uint16_t theSample;
};
```

Layer 2 : RecHits

- This is the first level of local reconstruction and common to all detector components. Each Digi element is converted into a hit with more physical quantity depending on the detector type.
- For calorimeters, RecHits are the energy and time for each hit (per channel)
- ECAL uses a two layer DataFormats for RecHit:
  - EcalUncalibratedRecHit amplitude/jitter/chi2 reconstruction. Global scale ADCtoGeV and intercalibration not applied
  - EcalRecHit fully calibrated rechit
- RecHits are the primary input for many higher-level reconstruction tasks: eg. electron/photon clustering

```
class EcalUncalibratedRecHit {
  public:
  typedef DetId key_type;
  EcalUncalibratedRecHit();
  EcalUncalibratedRecHit(const DetId& detId, const double& ampl, const double& ped,
                         const double& jit, const double& chi2);
  virtual ~EcalUncalibratedRecHit();
  double amplitude() const { return amplitude_; }
  double pedestal() const { return pedestal_; }
  double jitter() const { return jitter_; }
  double chi2() const { return chi2_; }
  DetId  id() const { return id_; }
 private:
  double amplitude_;   //< Reconstructed amplitude
  double pedestal_;    //< Reconstructed pedestal
  double jitter_;      //< Reconstructed time jitter
  double chi2_;        //< Chi2 of the fit
  DetId  id_;          //< Detector ID
};
```

```
class CaloRecHit {
public:
  explicit CaloRecHit(const DetId& id, float energy, float time);
  virtual ~CaloRecHit();
  float energy() const { return energy_; }
  float time() const { return time_; }
  const DetId& detid() const { return id_; }
private:
  DetId id_;
  float energy_;
  float time_;
};
```

Module Loading and Execution

- Modules (e.g. a hit reconstructor) are implemented as “Plugins”
  - Compiled in fully-bound shared libraries and “registered” to the framework
  - The framework takes care to load the plugin and instantiate the module when it is requested by the job configuration (a “card file”, see later)
  - The module is called at every event according to the path scheduled in the configuration
- No need to build binary executables for user code
  - Single pre-built configurable applications instead, e.g.:
    - cmsRun – main application for simulation and reconstruction behaviour driven by scheduler/configuration file

In COBRA/ORCA:

- Instantiate: “static-like” instantiation of all objects defined in a library (PKBuilder)
- Call: Observer/Dispatcher pattern and implicit invocation
- User builds binary executables that needs to be correctly linked against all libraries

Component Architecture

- Six types of dynamically loadable processing components, whose interface is specified by the FW:
  - Source
    - Provides the Event to be processed
    - Notable example: DaqSource, creating Events from the global DAQ
  - Producer
    - Creates new data to be placed in the Event
  - Filter
    - Decides if processing should continue for an Event
  - Analyzer
    - Studies properties of the Event
  - Looper
    - Module to control multi-pass looping
  - OutputModule
    - Stores the data from the Event

EDAnalyzer

- Most commonly used by users. Gives user access to data, do the analysis and save information to ROOT trees and histograms
  - Create CMSSW release area.
  - Use cmsenv to get definition of some environment variables
  - Create a subdirectory in $CMSSW_BASE/src and go there
  - Use mkedanlzr Name to create a skeleton code
  - Edit the code and become a part of CMS analysis team

```
#include <memory>
#include "FWCore/Framework/interface/Frameworkfwd.h“
#include "FWCore/Framework/interface/EDAnalyzer.h“
#include "FWCore/Framework/interface/Event.h“
#include "FWCore/Framework/interface/MakerMacros.h“
#include "FWCore/ParameterSet/interface/ParameterSet.h“
class TestA : public edm::EDAnalyzer {
public:
  explicit TestA(const edm::ParameterSet&);
 ~TestA();
 static void fillDescriptions(edm::ConfigurationDescriptions& descriptions);
private:
  virtual void beginJob() ;
  virtual void analyze(const edm::Event&, const edm::EventSetup&);
  virtual void endJob() ;
  virtual void beginRun(edm::Run const&, edm::EventSetup const&);
  virtual void endRun(edm::Run const&, edm::EventSetup const&);
  virtual void beginLuminosityBlock(edm::LuminosityBlock const&, edm::EventSetup const&);
  virtual void endLuminosityBlock(edm::LuminosityBlock const&, edm::EventSetup const&);
};
```

EDAnalyzer is a plugin and needs to be defined as such

```
DEFINE_FWK_MODULE(TestA);
```

EDProducer

- CMSSW uses the concept of producer modules and products. It takes data from the Event in one format, analyzes the data and stores information into the event in another format.
- The key element in a producer is
  - Declare all possible products in the constructor
  - Use the method to attach a (sub)set of the products to the Event

  void produce(edm::Event& e, const edm::EventSetup& es)

To define a product has to include the name of the product in (Sim)DataFormat which is translated for ROOT through Reflex

classes.h

```
#include "SimDataFormats/CaloHit/interface/PCaloHit.h“
#include "SimDataFormats/CaloHit/interface/PCaloHitContainer.h“
#include "DataFormats/Common/interface/Wrapper.h“
namespace {
  struct dictionary {
     PCaloHit                                    rv3;
    edm::PCaloHitContainer                       v3;
    std::vector<const PCaloHit*>                 vcp3;
    edm::Wrapper<edm::PCaloHitContainer>         wc3;
  };
}
```

classes_def.xml

```
<lcgdict>
  <class name="PCaloHit" ClassVersion="10">
  <version ClassVersion="10" checksum="682759659"/>
 </class>
 <class name="std::vector<PCaloHit>"/>
 <class name="std::vector<const PCaloHit*> "/>
<class name="edm::Wrapper<std::vector<PCaloHit>>“
        splitLevel="0"/>
</lcgdict>
```

An example of Producer Class

- Need to publicly inherit from the class EDProducer
- Declare all products which can be produced in the constructor itself
- Attach all products to event in method produce

Declaration

```
class OscarProducer : public edm::EDProducer{

public:
   typedef std::vector<boost::shared_ptr<SimProducer> >
Producers;
   explicit OscarProducer(edm::ParameterSet const & p);
   virtual ~OscarProducer();
   virtual void beginRun(edm::Run & r,const edm::EventSetup & c);
   virtual void beginJob();
   virtual void endJob();
   virtual void produce(edm::Event & e, const edm::EventSetup & c);
```

Method produce

```
for (std::vector<SensitiveCaloDetector*>::iterator it = sCalo.begin();
       it != sCalo.end(); it++) {
      std::vector<std::string>  v = (*it)->getNames();
      for (std::vector<std::string>::iterator in = v.begin(); in!= v.end(); in++) {
         std::auto_ptr<edm::PCaloHitContainer>
           product(new edm::PCaloHitContainer);
         (*it)->fillHits(*product,*in);
         e.put(product,*in);
      }
   }
```

Constructor

```
OscarProducer::OscarProducer(edm::ParameterSet const & p) {

    ……
  produces<edm::PSimHitContainer>("TotemHitsT1");
  produces<edm::PSimHitContainer>("TotemHitsT2Gem");
  produces<edm::PSimHitContainer>("TotemHitsRP");
  produces<edm::PSimHitContainer>("FP420SI");
  produces<edm::PSimHitContainer>("BSCHits");
  produces<edm::PSimHitContainer>("PLTHits");
  produces<edm::PCaloHitContainer>("EcalHitsEB");
  produces<edm::PCaloHitContainer>("EcalHitsEE");
    ……
 m_runManager = new RunManager(p);
 //register any products
 m_producers= m_runManager->producers();
 for(Producers::iterator itProd = m_producers.begin();
     itProd != m_producers.end();
     ++itProd) {
    (*itProd)->registerProducts(*this);
 }
```

Input to cmsRun

- Input to cmsRun is a configuration file written in Python
- CMSSW provides various utilities and each such utility provides one interface file to configuration "cfi"
- To perform a specific task, many such "cfi"s can be used and this makes Configuration File Fragments "cff"
- Typically the input to cmsRun is to load the relevant "cff"s and then schedule a path for execution
- A non-expert can utilize cmsDriver.py to make the input configuration file

```
cmsDriver.py SingleElectronPt10_cfi --conditions auto:upgradePLS3 -n 10 --eventcontent FEVTDEBUG \
  --relval 9000,300 -s GEN,SIM --datatier GEN-SIM --beamspot Gauss \
  --customise SLHCUpgradeSimulations/Configuration/combinedCustoms.cust_2023HGCalMuon \
  --geometry Extended2023HGCalMuon,Extended2023HGCalMuonReco --magField 38T_PostLS1 \
 --fileout file:step1.root  --no_exec
```

will generate a cfg file as given in the next pages

Part 1 of cfg file

```
import FWCore.ParameterSet.Config as cms

process = cms.Process('SIM')
# import of standard configurations
process.load('Configuration.StandardSequences.Services_cff')
process.load('SimGeneral.HepPDTESSource.pythiapdt_cfi')
process.load('FWCore.MessageService.MessageLogger_cfi')
process.load('Configuration.EventContent.EventContent_cff')
process.load('SimGeneral.MixingModule.mixNoPU_cfi')
process.load('Configuration.Geometry.GeometryExtended2023HGCalMuonReco_cff')
process.load('Configuration.Geometry.GeometryExtended2023HGCalMuon_cff')
process.load('Configuration.StandardSequences.MagneticField_38T_PostLS1_cff')
process.load('Configuration.StandardSequences.Generator_cff')
process.load('IOMC.EventVertexGenerators.VtxSmearedGauss_cfi')
process.load('GeneratorInterface.Core.genFilterSummary_cff')
process.load('Configuration.StandardSequences.SimIdeal_cff')
process.load('Configuration.StandardSequences.EndOfProcess_cff')
process.load('Configuration.StandardSequences.FrontierConditions_GlobalTag_cff')

process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(10)
)
# Input source
process.source = cms.Source("EmptySource")
process.options = cms.untracked.PSet( )
```

Part 2 of cfg file

```
# Production Info
process.configurationMetadata = cms.untracked.PSet(
   version = cms.untracked.string('$Revision: 1.20 $'),
   annotation = cms.untracked.string('SingleElectronPt10_cfi nevts:10'),
   name = cms.untracked.string('Applications')
)

# Output definition
process.FEVTDEBUGoutput = cms.OutputModule("PoolOutputModule",
    splitLevel = cms.untracked.int32(0),
    eventAutoFlushCompressedSize = cms.untracked.int32(5242880),
    outputCommands = process.FEVTDEBUGEventContent.outputCommands,
    fileName = cms.untracked.string('file:step1.root'),
    dataset = cms.untracked.PSet( filterName = cms.untracked.string(''),
                                  dataTier = cms.untracked.string('GEN-SIM')
    ),
    SelectEvents = cms.untracked.PSet( SelectEvents = cms.vstring('generation_step') )
 )

# Additional output definition
# Other statements
process.genstepfilter.triggerConditions=cms.vstring("generation_step")

from Configuration.AlCa.GlobalTag import GlobalTag
process.GlobalTag = GlobalTag(process.GlobalTag, 'auto:upgradePLS3', '')
```

Part 3 of cfg file

```
process.generator = cms.EDProducer("FlatRandomPtGunProducer",
   PGunParameters = cms.PSet(
          MaxPt = cms.double(10.01),
          MinPt = cms.double(9.99),
          PartID = cms.vint32(11),
          MaxEta = cms.double(2.5),
          MaxPhi = cms.double(3.14159265359),
          MinEta = cms.double(-2.5),
          MinPhi = cms.double(-3.14159265359)
   ),
  Verbosity = cms.untracked.int32(0),
  psethack = cms.string('single electron pt 10'),
  AddAntiParticle = cms.bool(True),
  firstRun = cms.untracked.uint32(1)
)

# Path and EndPath definitions
process.generation_step = cms.Path(process.pgen)
process.simulation_step = cms.Path(process.psim)
process.genfiltersummary_step = cms.EndPath(process.genFilterSummary)
process.endjob_step = cms.EndPath(process.endOfProcess)
process.FEVTDEBUGoutput_step = cms.EndPath(process.FEVTDEBUGoutput)
```

Part 4 of cfg file

```
# Schedule definition
process.schedule = cms.Schedule(process.generation_step,
                                process.genfiltersummary_step,
                                process.simulation_step,
                                process.endjob_step,
                                process.FEVTDEBUGoutput_step)

# filter all path with the production filter sequence
for path in process.paths:
    getattr(process,path)._seq = process.generator * getattr(process,path)._seq

# customisation of the process.

# Automatic addition of the customisation function from
# SLHCUpgradeSimulations.Configuration.combinedCustoms
from SLHCUpgradeSimulations.Configuration.combinedCustoms  import cust_2023HGCalMuon

#call to customisation function cust_2023HGCalMuon imported from
# SLHCUpgradeSimulations.Configuration.combinedCustoms
process = cust_2023HGCalMuon(process)

# End of customisation functions
```

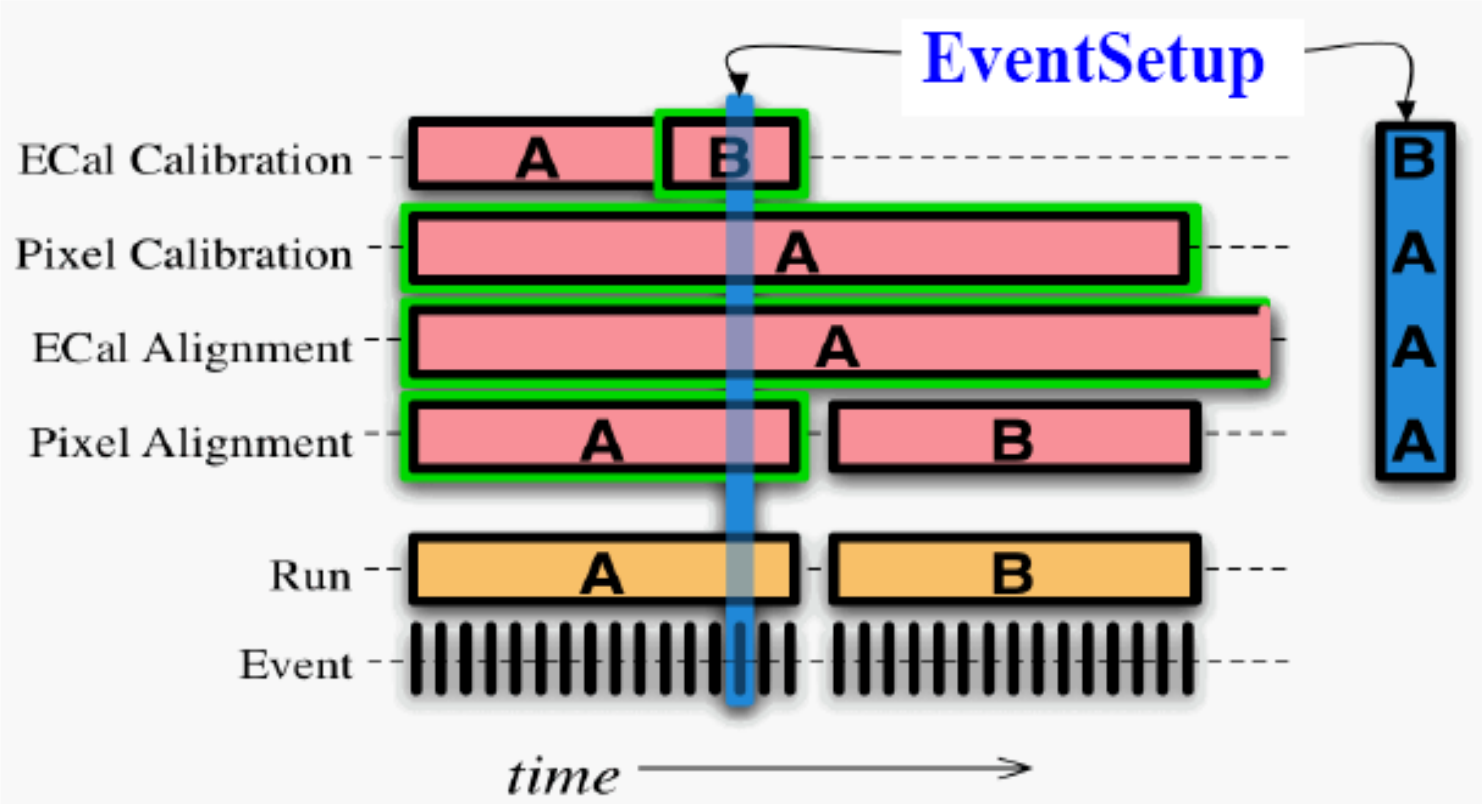
Execution Path

- The EDM framework allows two ways to set up an execution path (the order in which the job will be run)
  - Scheduled: the configuration file explicitly states the order
  - Unscheduled: the configuration file needs to specify the order of the filters and analyzers, The producers need not be specified and will be run the first time someone asks for their data
    - this is data on demand (old ORCA philosophy)
- For unscheduled path, user needs to specify

```
process.options = cms.untracked.PSet(
                                    allowUnscheduled = cms.untracked.bool(True)
)
```

Asynchronous Data

- The EventSetup is the system which delivers all non event data (calibration, geometry...) to producers and other modules.



Event Setup

- Like for Data, there are producer modules ESProducers which attaches asynnchronous data to EventSetup
- There are two types of data items which can be accessed. The main distinction is for memory foot print. The memory management is done by the framework:
  - Needs to be accessed once per IOV: ESTransientHandle<>
  - Needs to be accessed during the full IOV: ESHandle<>
- A data item can be retrieved from an EventSetup Record by passing an ESHandle<> to the Record's get method

```
edm::ESHandle<MagneticField> bFieldH;
iSetup.get<IdealMagneticFieldRecord>().get(bFieldH);
bField = bFieldH.product();
```

Structure of the code

- Single CVS repository: now CVS has retired. So move to git
  - The corresponding of COBRA, OSCAR, ORCA, COSINE, IGUANACMS is now in a single project CMSSW
  - Completely different release procedure
- Packages organized by similar functionality
  - Rather than by subdetector, like in ORCA
  - No single Calorimetry subsystem
- To learn about CVS basic command just seach Google for “CVS tutorial”
  - http://cvs.nongbu.org/
  - http://ximbiot,com/cvs/manual
- Now learn about basic git tutorial: use twiki page for git
- CMSSW Code Browsers:
  - LXR
    - http://cmslxr.fnal.gov/lxr

General tools

- Other tools that you need to know something is scramv1 (already used in ORCA/COBRA world). Things to know:
  - create a release area

```
cmsrel <CMSSW release>
```

  - Buildifiles. This is a XML file with package definition to be used to make the shared library
  - build code. From the src/ or inside the package dir you have to issue:

```
scram b
scram b –r (to rebuild and clear caches)
scram b clean
```

  - Always use the command "cmsenv" to define several environment variables
  - Each package has a file named "BuildFile.xml" which contains the dependencies to create libraries of a package
- Many other "new things" are not mentioned here…
  - Connection with databases
  - Developer tools
    - 2 Integration builds per day, Release procedure, etc.

Example session

- This illustrates how to start a session:

```
cmsrel CMSSW_7_1_0
cd CMSSW_7_1_0/src
cmsenv
git cms-addpkg SimG4CMS/Calo
scram b –r
cmsRun SimG4CMS/Calo/test/python/runHFPMT_cfg.py
```

- Each package have the following sub-directories
  - interface: header files for classes to be used in this as well as other packages
  - src: source files (also classes.h and classes_def.xml)
  - python: cfi and cff files (python scripts)
  - plugins: classes which can be loaded dynamically through python commands
  - data: data files if any
  - test: testing code and useful cfg files for testings

Parameter Set

- Most of the modules use some data cards to control the flow of code
- The data cards are specified in the configuration file
- The module has to declare the parameters to be used (preferably in the constructor)
- Parameters can be nested and variables come in many types

```
killBeamPipe = (p.getParameter<bool>("KillBeamPipe"));
theCriticalDensity = (p.getParameter<double>("CriticalDensity")*g/cm3);
 maxTrackTime    = (p.getParameter<double>("MaxTrackTime")*ns);
maxTrackTimes   = (p.getParameter<std::vector<double> >("MaxTrackTimes"));
maxTimeNames   = (p.getParameter<std::vector<std::string> >("MaxTimeNames"));
ekinMins           = (p.getParameter<std::vector<double> >("EkinThresholds"));
ekinNames         = (p.getParameter<std::vector<std::string> >("EkinNames"));
ekinParticles      = (p.getParameter<std::vector<std::string> >("EkinParticles"));
verbose            = (p.getUntrackedParameter<int>("Verbosity",0));
```

```
SteppingAction = cms.PSet(
    common_maximum_time,
    KillBeamPipe          = cms.bool(True),
    CriticalDensity       = cms.double(1e-15),
    EkinNames             = cms.vstring(),
    EkinThresholds        = cms.vdouble(),
    EkinParticles         = cms.vstring(),
    Verbosity = cms.untracked.int32(0)
```

```
common_maximum_time = cms.PSet(
   MaxTrackTime  = cms.double(500.0),
   MaxTimeNames  = cms.vstring('ZDCRegion','QuadRegion'),
   MaxTrackTimes = cms.vdouble(2000.0)
)
```

Services

- A facility that performs a well-defined task that is globally acessible and that does not affect physics results.
  - Histogram service
  - Timing Service
  - Message logging Service

Timing

```
process.Timing = cms.Service("Timing")
```

Memory Checks

```
process.SimpleMemoryCheck = cms.Service("SimpleMemoryCheck",
  oncePerEventMode = cms.untracked.bool(True),
  showMallocInfo = cms.untracked.bool(True),
  dump = cms.untracked.bool(True),
  ignoreTotal = cms.untracked.int32(1)
)
```

Message Logging

- Messages are categorized into 4 parts according to severity: Error, Warning, Information, Debug
- The corresponding functions in the code:
  - edm::LogError (string)
  - edm::LogWarning (string)
  - edm::LogInfo (string)
  - LogDebug (string)
- In the config file the “string” is used to activate/deactivate messaging

```
process.MessageLogger = cms.Service("MessageLogger",
  destinations = cms.untracked.vstring('cout'),
  categories = cms.untracked.vstring('CaloSim', 'EcalSim', 'HcalSim'),
  cout = cms.untracked.PSet(
       default = cms.untracked.PSet(
           limit = cms.untracked.int32(-1)
       ),
       CaloSim = cms.untracked.PSet(
           limit = cms.untracked.int32(0)
       ),
       EcalSim = cms.untracked.PSet(
           limit = cms.untracked.int32(0)
       ),
       HcalSim = cms.untracked.PSet(
           limit = cms.untracked.int32(0)
       )
)
```

TFile Service

- This is the most useful way to store tree/histogram (like using bare ROOT)
- It automatically writes on the file and saves the output on exit – the user has to do the booking and fill appropriate tree/histograms

User Code

```
edm::Service<TFileService> tfile;
 if ( !tfile.isAvailable() )
   throw cms::Exception("BadConfig") << "TFileService unavailable: “
                                     << "please add it to config file";
 h_NEventProc  = fs->make<TH1I>("h_NEventProc",  "h_NEventProc", 2, -0.5, 0.5);
 TFileDirectory dir = fs->mkdir("nearMaxTrackP");
 for (unsigned int ieta=0; ieta<NEtaBins; ieta++) {
   double lowEta  = genPartEtaBins[ieta];  double highEta = genPartEtaBins[ieta+1];
   for (unsigned int ipt=0; ipt<NPBins; ipt++) {
     double lowP    = genPartPBins[ipt];      double highP   = genPartPBins[ipt+1];
     sprintf (hname, "h_maxNearP31x31_ptBin%i_etaBin%i",ipt, ieta);
     sprintf (htit,  "maxNearP in 31x31 (%3.2f<|#eta|<%3.2f), (%2.0f<trkP<%3.0f)", lowEta,
              highEta, lowP, highP );
     h_maxNearP31x31[ipt][ieta] = dir.make<TH1F>(hname, htit, 220, -2.0, 20.0);    } }
```

Configuration:

```
process.TFileService = cms.Service("TFileService",
                                   fileName = cms.string('IsolatedTrack.root')
                                   )
```

Some Examples

- Example with no source, a producer producing a string of integer and an analyzer working on that:

Simple unit test example

```
process = Process("Test")

process.source = Source("EmptySource")
process.int = EDProducer("IntProducer",
                         ivalue = int32(2) )

process.test = EDAnalyzer("IntTestAnalyzer",
                  valueMustMatch = untracked.int32(2),
                  moduleLabel = untracked.InputTag("int") )

...

process.p = Path( process.int + process.test)
```

Examples

- More realistic example

input

```
process.source = cms.source("PoolSource",
                 fileNames = cms.untracked.vstring("file:input.root")
)
process.maxEvents = cms.untracked.PSet(
                  input = cms.untracked.int32(-1)
)
```

PROCESS

output

```
process.out = cms.OutputModule ("PoolOutputModule",
                 process.FEVTSimEventContent,
                 fileName = cms.untracked. string("output.root")
)
```

```
outputCommands = cms.untracked.vstring("keep *",
                                 "drop *_*_*_HLT",
                                 "keep FEDRawDataCollection_*_*_*")
```

Real Example

```
import FWCore.ParameterSet.Config as cms
process =cms.Process("Sim")
process.load("SimG4CMS.Calo.PythiaMinBias_cfi")
process.load("SimGeneral.HepPDTESSource.pythiapdt_cfi")
process.load("IOMC.EventVertexGenerators.VtxSmearedGauss_cfi")
process.load("Geometry.CMSCommonData.cmsIdealGeometryXML_cfi")
process.load("Geometry.TrackerNumberingBuilder.trackerNumberingGeometry_cfi")
process.load("Configuration.StandardSequences.MagneticField_cff")
process.load("SimG4Core.Application.g4SimHits_cfi")
process.load("SimG4CMS.Calo.CaloSimHitStudy_cfi")
process.source = cms.Source("EmptySource")
process.maxEvents = cms.untracked.PSet(
   input = cms.untracked.int32(200))
process.MessageLogger = cms.Service("MessageLogger",
   destinations = cms.untracked.vstring('cout'),
   categories = cms.untracked.vstring('SimG4CoreGeometry'),
   cout = cms.untracked.PSet(
     default = cms.untracked.PSet(
        limit = cms.untracked.int32(-1)
      ),
     SimG4CoreGeometry = cms.untracked.PSet(
        limit = cms.untracked.int32(0)
     )
)
process.Timing = cms.Service("Timing")
process.load("IOMCRandomEngine.IOMC_cff")
process.RandomNumberGeneratorService.generator.initialSeed = 456789
process.RandomNumberGeneratorService.g4SimHits.initialSeed  = 9876
 process.RandomNumberGeneratorService.VtxSmeared.initialSeed = 98765432

process.rndmStore = cms.EDProducer("RandomEngineStateProducer")
process.TFileService = cms.Service("TFileService",
   fileName = cms.string('minbias_QGSP_BERT_EML.root'))

process.load("Configuration.EventContent.EventContent_cff")
process.o1 = cms.OutputModule("PoolOutputModule",
   process.FEVTSIMEventContent,
   fileName = cms.untracked.string('simevent_minbias_QGSP_BERT_EML.root')
)
process.p1 = cms.Path(process.generator*process.VtxSmeared*process.g4SimHits*process.caloSimHitStudy*process.rndmStore)
process.outpath = cms.EndPath(process.o1)
process.g4SimHits.Physics.type = 'SimG4Core/Physics/QGSP_BERT_EML'
```

PYTHIA MinBias


```
import FWCore.ParameterSet.Config as cms

from Configuration.Generator.PythiaUESettings_cfi import *
generator = cms.EDFilter("Pythia6GeneratorFilter",
    pythiaHepMCVerbosity = cms.untracked.bool(False),
    maxEventsToPrint = cms.untracked.int32(0),
    pythiaPylistVerbosity = cms.untracked.int32(0),
    filterEfficiency = cms.untracked.double(1.0),
    comEnergy = cms.double(10000.0),
    PythiaParameters = cms.PSet(
        pythiaUESettingsBlock,
        processParameters = cms.vstring('MSEL=0         ! User defined processes',
            'MSUB(11)=1     ! Min bias process',
            'MSUB(12)=1     ! Min bias process',
            'MSUB(13)=1     ! Min bias process',
            'MSUB(28)=1     ! Min bias process',
            'MSUB(53)=1     ! Min bias process',
            'MSUB(68)=1     ! Min bias process',
            'MSUB(92)=1     ! Min bias process, single diffractive',
            'MSUB(93)=1     ! Min bias process, single diffractive',
            'MSUB(94)=1     ! Min bias process, double diffractive',
            'MSUB(95)=1     ! Min bias process'),
        # This is a vector of ParameterSet names to be read, in this order
        parameterSets = cms.vstring('pythiaUESettings',
            'processParameters')
    ))
```

Input Sources

- First ingredient to every execution schedule is the primary input source which builds the event
- Main sources are available up to now:
  - PoolSource
    - Read events from POOL/ROOT file containing data already in POOL format (written by CMSSW e.g. simulated data)
  - EmptySource
    - Just to produce fake events - mainly used for testing or using a built in generator, like PYTHIA, particle guns, ...
  - Generator interface
    - To produce MC events supplied in the standard LHE format. Several event generators, particle guns are directly interfaced to CMSSW. These are used for different purposes and do not appear as source – rather as producer or filter.
  - StreamerData
    - Read events output from a DCC raw data files

Examples of input sources

- PoolSource

```
process.source = cms.Source("PoolSource",
    fileNames =cms.untracked.vstring(
'/store/mc/Summer10/DiPion_E1to300/GEN-SIM-RECO/START36_V9_S09-v1/0024/4CEE3150-E581-DF11-B9C4-001A92971BDC.root')
)
```

- EmptySource

```
process.source = cms.Source("EmptySource",
    firstRun        = cms.untracked.uint32(1),
    firstEvent      = cms.untracked.uint32(1)
)
```

- Special generator output format

```
process.source = cms.Source("LHESource",
   fileNames = cms.untracked.vstring('file:monopole.lhe')
)
```

- HcalTBInputService

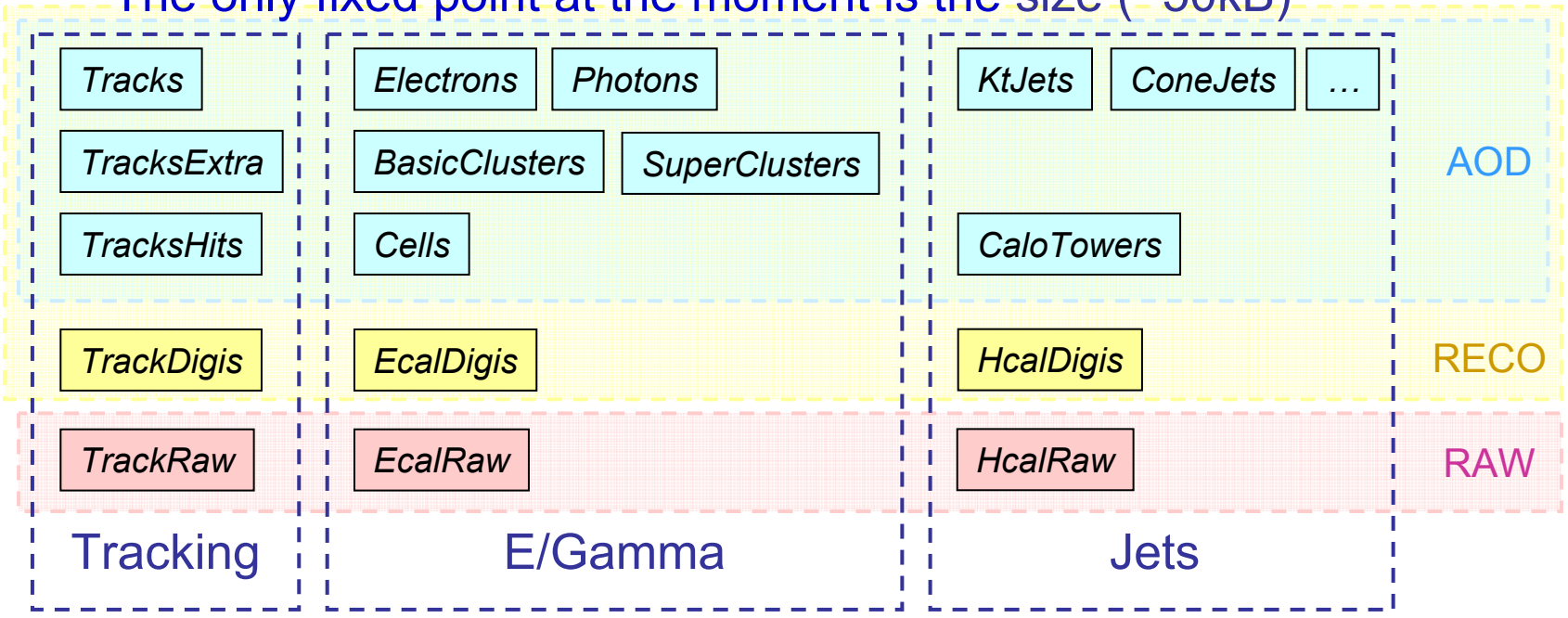
```
process.source = cms.Source("HcalTBSource",
   streams = cms.untracked.vstring('HCAL_Trigger','HCAL_DCC020','HCAL_SlowData:3','HCAL_TDC:5'),
   fileNames = cms.untracked.vstring('file:HTB_011609.root')
)
```

- SiliconPixelInputService

```
process.source = cms.Source("PixelSLinkDataInputSource",
    runNumber = cms.untracked.int32(-1),
    fileNames = cms.untracked.vstring('file:PixelAlive_070106d.dmp'),
    fedid = cms.untracked.int32(-1)
)
```

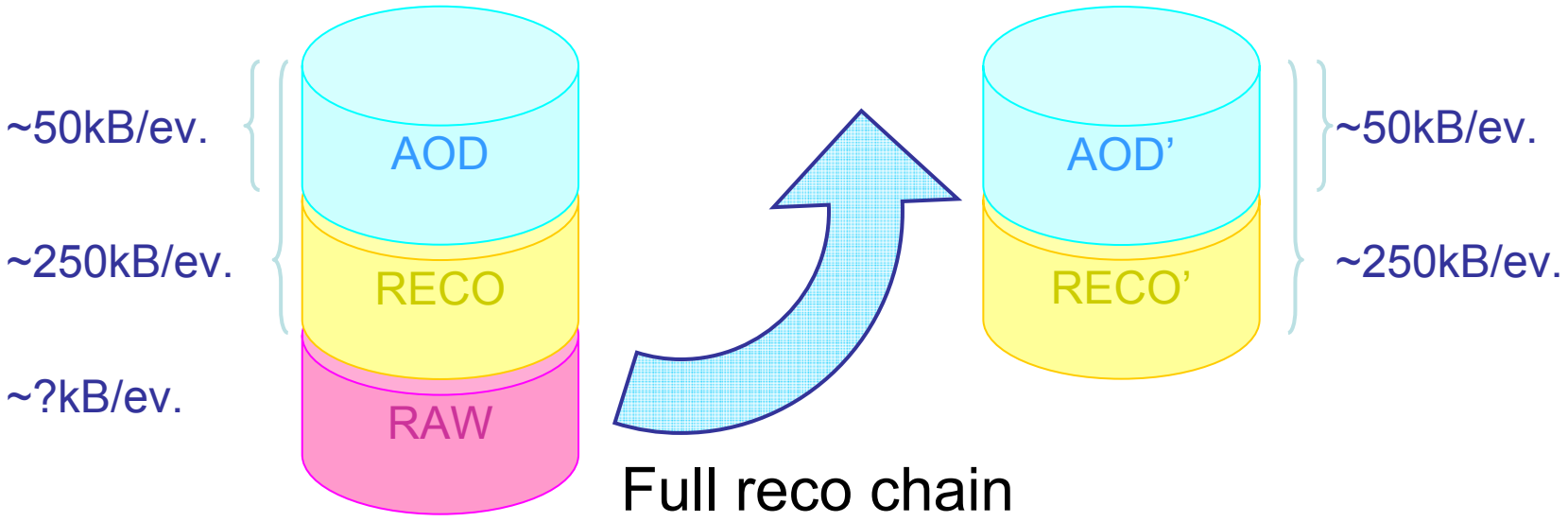
Contents in Output Module

- Output exists in the form of RAW (SIM), RECO and AOD
- AOD is a subset of the RECO
  - The exact content is a matter of definition
  - It is driven by Physics requirements
  - Changes with the lifetime/luminosity of the experiment
  - The only fixed point at the moment is the size (~50kB)



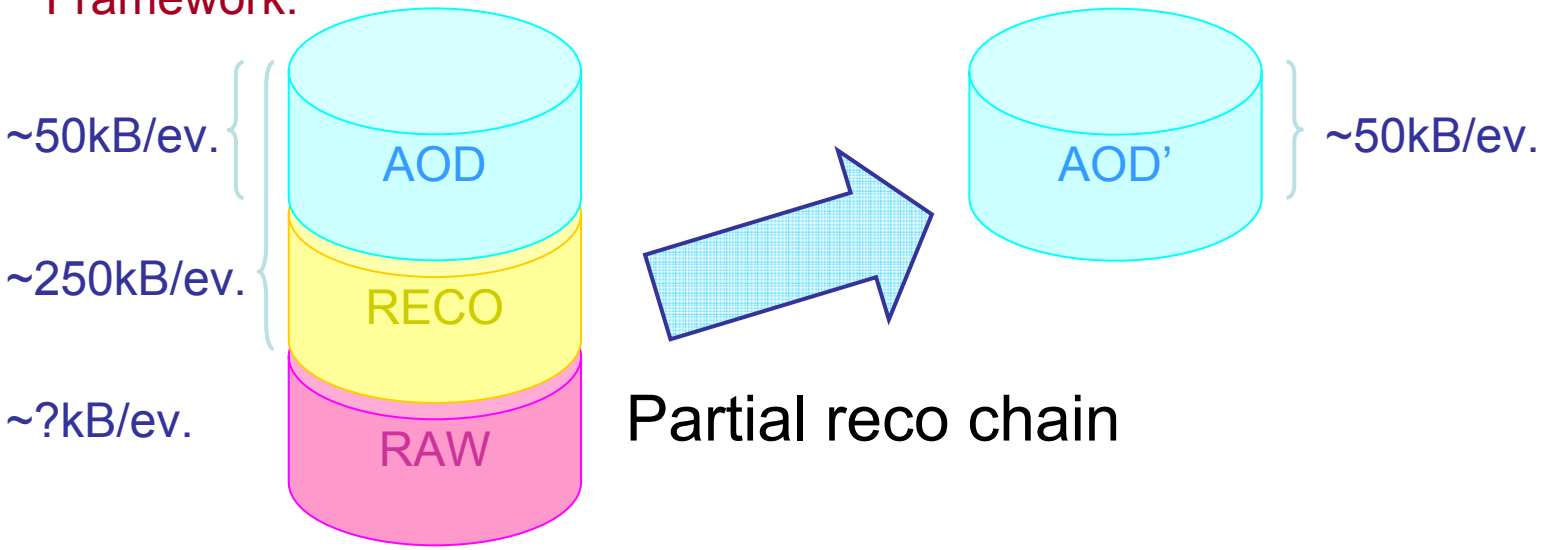
Reprocessing: RECO→RECO

- Restart the full reconstruction chain from RAW
  - (RECO, AOD)→(RECO', AOD')
- RAW need not be duplicated



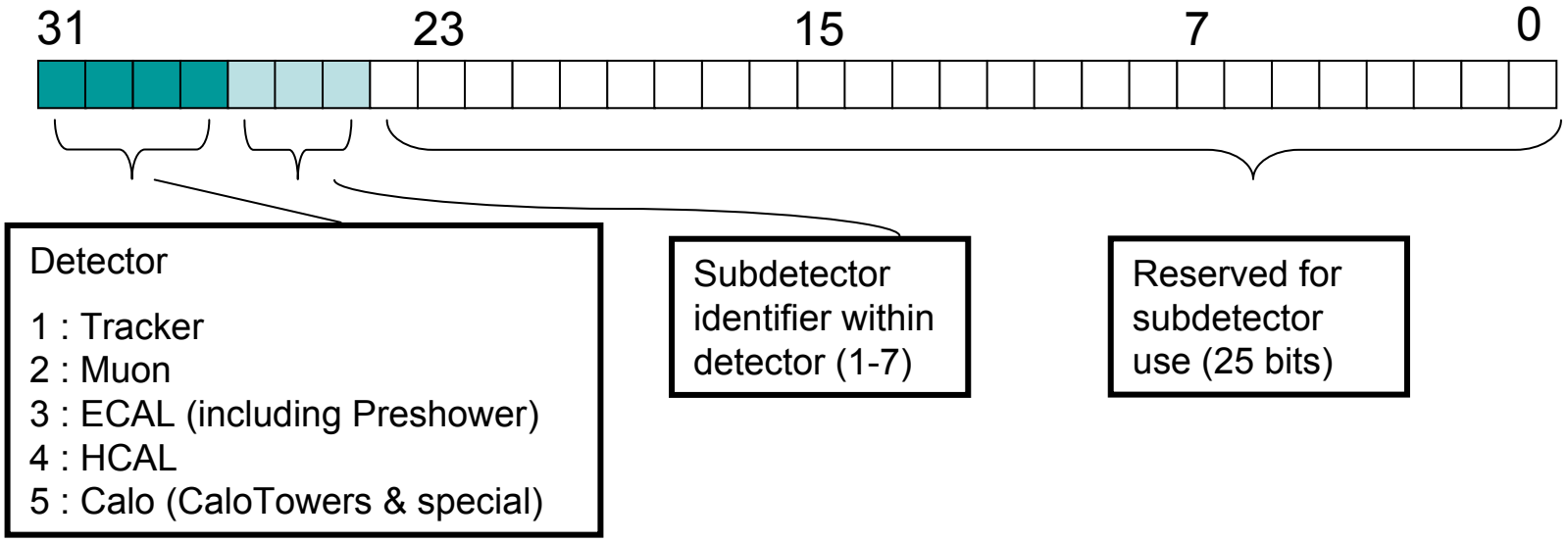
Reprocessing: RECO→AOD

- Restart a subset of reconstruction chain from RECO or AOD
  - (RECO, AOD)→(RECO, AOD') or (AOD)→(AOD')
  - Not necessarily a subset of RECO sequence (different modules!)
  - E.g.: refit tracks, don't redo pattern recognition
- RECO and RAW need not be duplicated
- Replicating RECO'=RECO is a duplication of disk space: 250kB / 50kB = extra factor ×5
- How to avoid name clash (Electrons → Electrons)? Not foreseen now in Framework.



DetId

- DetIds identify low-level objects like a single crystal, HCAL tower, a strip in the silicon tracker, …
- Fundamentally, a DetId is just a 32-bit unsigned integer packed in a specific way
  - High bits identify the part of CMS
  - Low bits are used by the sub-detectors



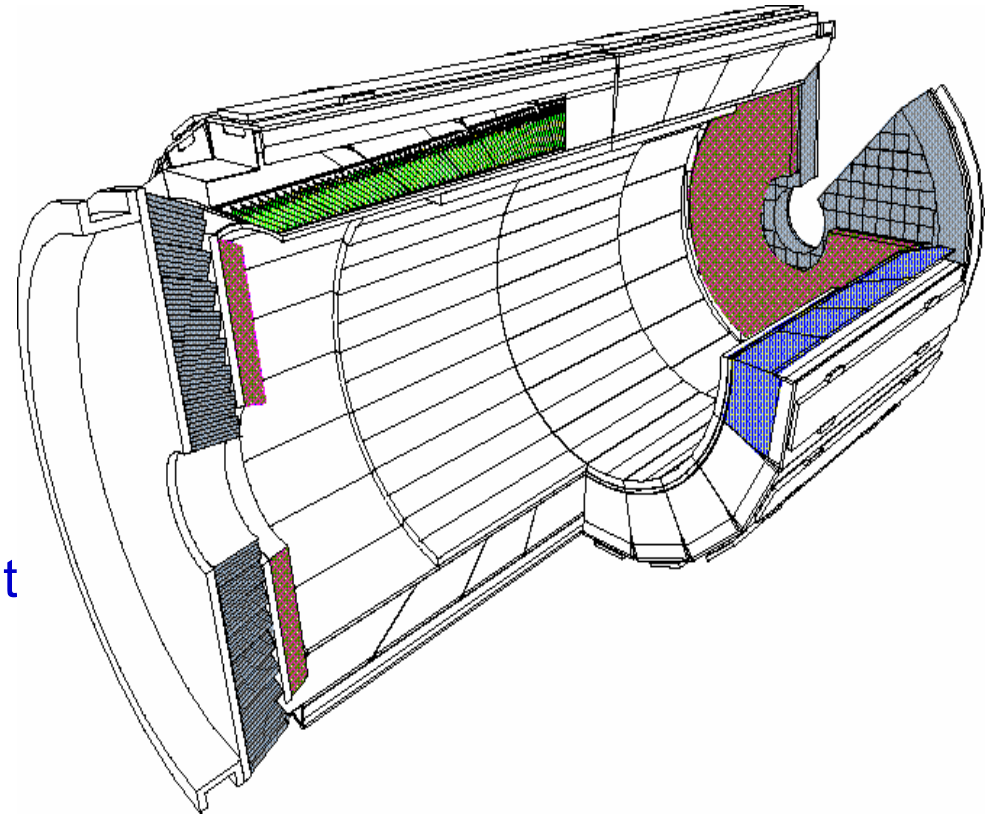
Calorimetry DetIds

- Many different Calorimetry Subdetectors:
  - ECAL: EB (Barrel), EE (Endcap), ES (Preshower), EcalTriggerPrimitives
  - HCAL: HB, HE, HF (Forward), HO (Outer), HcalTriggerPrimitives
- Calorimetry DetIds have been designed to match the topology of the built detector in a natural and useful way
  - ieta/iphi or x/y as appropriate

```
class DetId {
public:
  static const int kDetOffset          = 28;
  static const int kSubdetOffset    = 25;
  enum Detector { Tracker=1,Muon=2,Ecal=3,Hcal=4,Calo=5 };
  DetId()  : id_(0) { }  DetId(uint32_t id) : id_(id) { }
  DetId(Detector det, int subdet)  { id_=((det&0xF)<<28)|((subdet&0x7)<<25); }
  Detector det() const { return Detector((id_>>kDetOffset)&0xF); }
  int subdetId() const { return ((id_>>kSubdetOffset)&0x7); }
  uint32_t operator()() const { return id_; }
  bool operator==(DetId id) const { return id_==id.id_; }
protected:
  uint32_t id_;
};
  inline bool operator==(uint32_t i, DetId id)  { return i==id(); }
  inline bool operator==(DetId id, uint32_t i)  { return i==id(); }
```

EBDetId + EEDetId

- EB is an eta/phi grid
  - IETA : -85 → +85
    - No crystal at IETA=0
  - IPHI : 1 → 360
- EE is an X/Y grid
  - X and Y each run from 1-100
  - Many ids do not map to a crystal given the center cutout and circular profile.



EBDetId: the Class

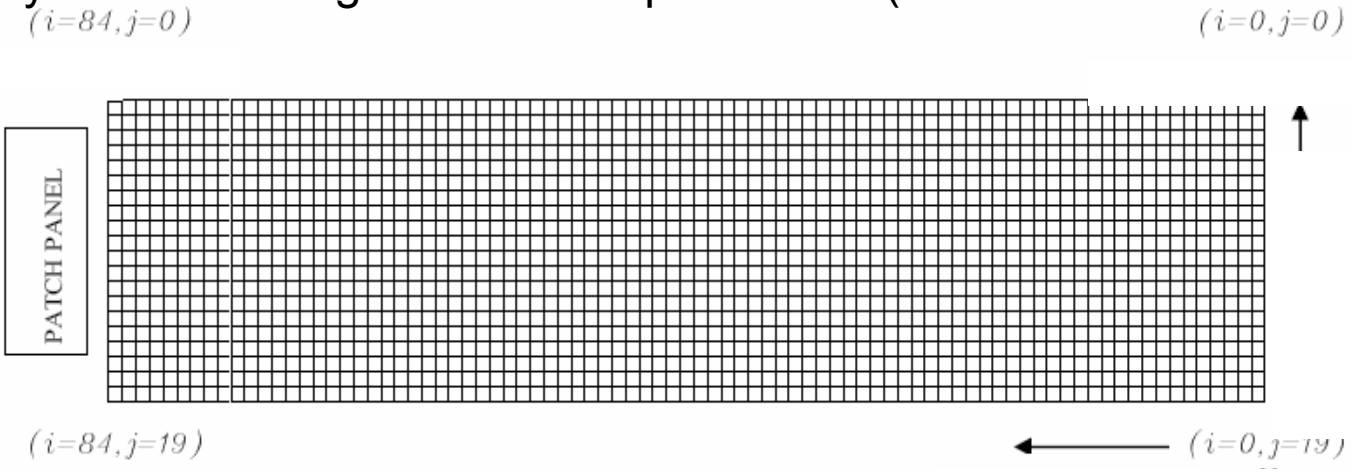
```
class EBDetId : public DetId {
public:
  enum { Subdet=EcalBarrel};
  /** Constructor of a null id */
  EBDetId() {}
  /** Constructor from a raw value */
  EBDetId(uint32_t rawid);
  /** Constructor from crystal ieta and iphi or from SM# and crystal# */
  EBDetId(int index1, int index2, int mode = ETAPHIMODE);
  /** Constructor from a generic cell id */
  EBDetId(const DetId& id);
  /** Assignment operator from cell id */
  EBDetId& operator=(const DetId& id);
  /// get the subdetector .i.e EcalBarrel (what else?)  //
  EcalSubdetector subdet() const;
  static EcalSubdetector subdet() { return EcalBarrel;}
  /// get the z-side of the crystal (1/-1)
  int zside() const { return (id_&0x10000)?(1):(-1); }
  /// get the absolute value of the crystal ieta
  int ietaAbs() const { return (id_>>9)&0x7F; }
  /// get the crystal ieta
  int ieta() const { return zside()*ietaAbs(); }
  /// get the crystal iphi
  int iphi() const { return id_&0x1FF; }
  /// get the HCAL/trigger ieta of this crystal
  int tower_ieta() const { return ((ietaAbs()-1)/5+1)*zside(); }
  /// get the HCAL/trigger iphi of this crystal
  int tower_iphi() const;
  /// get the HCAL/trigger iphi of this crystal
  EcalTrigTowerDetId tower() const;
  /// get the ECAL/SM id
  int ism() const;
  /// get the number of module inside the SM (1-4)
  int im() const;

  /// get ECAL/crystal number inside SM
  int ic() const;
  /// get the crystal ieta in the SM convention (1-85)
  int ietaSM() const { return ietaAbs(); }
```

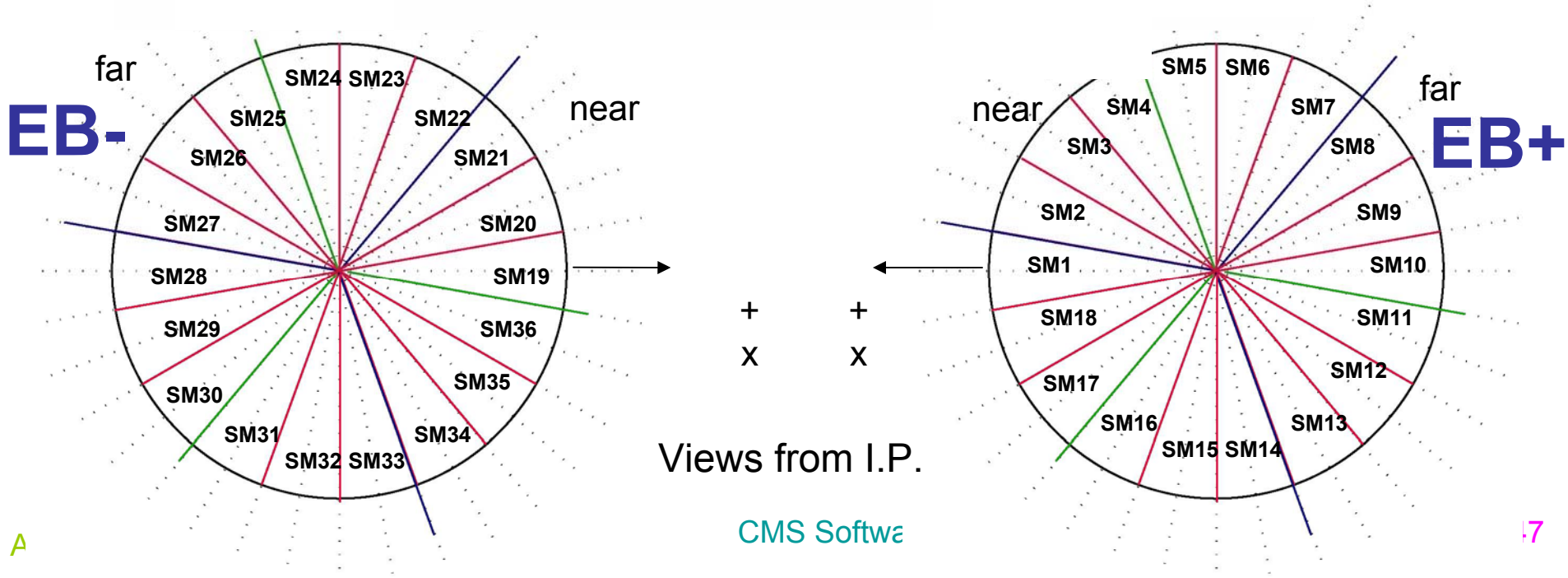
Two main modes for the constructor are given: one is ETAPHIMODE (providing ieta and iphi index), the other is SMCRYSTALMODE (providing sm number and crystal number).
If values of indices are found out of the expected range a std::runtime_error exception is thrown.
You can catch it in your code using:

```
try
    {
      EBDetId anId(iSM,iC,EBDetId::SMCRYSTALMODE);
    }
  catch ( std::runtime_error &e )
    {
      std::cout << "Cannot construct EBDetId" << std::endl;
      return;
    }
```

Numbering: SM & crystal number in CMSSW

Crystal numbering inside the supermodule (back view from the electronics)





Geometry & Topology

- Geometry associates DetIds to their global position in CMS. Topology instead fixes the relations between DetIds (e.g. adiacent DetIds in a certain direction)
- Geometry and Topology records are retrieved via EventSetup.
- Geomerty and topology specific to ECAL/HCAL derives from the base classes CaloGeometry, CaloTopology

```
class HcalGeometry : public CaloSubdetectorGeometry  {
public:
     HcalGeometry();
     HcalGeometry(const HcalTopology * topology);
     virtual ~HcalGeometry();
     virtual const std::vector<DetId>& getValidDetIds();
     virtual DetId getClosestCell(const GlobalPoint& r) const ;
     virtual CaloSubdetectorGeometry::DetIdSet getCells(const GlobalPoint&, double) const ;
private:
     const HcalTopology * theTopology;
     mutable std::vector<DetId> m_hbIds ;
     std::vector<IdealObliquePrism> m_hbCellVec ;
};
```

- Geometry is constructed using the DetectorDescription xml files or via a DB (as it was in ORCA), while Topology is constructed using the Geometry record (hence dinamically from xml). Simulation geometry is ideal and does not have misalignment. Reconstruction geometry starts with ideal geometry and adds misalignments as measured from the data.

Geometry Definition

- CMS has developed a detector description language so that one can describe the geometry for simulation and reconstruction within the same approach
- There are geometry builders which interprets the common source and constructs application specific views
- Detector description defines geometry in term of some standard nomenclature: solid, logical part, pos part, algorithms, spec par, …
  - Solids; defines dimensional quantities and shapes
  - Logical Parts: relates solids to materials
  - Pos Parts: defines relative positions of logical parts through simple translation/rotation or using algorithms
  - SpecPar: Special parameters accessible to the geometry builders
  - ……

Sample Geometry XML file

```
<?xml version="1.0"?><DDDefinition xmlns="http://www.cern.ch/cms/DDL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.cern.ch/cms/DDL
../../../DetectorDescription/Schema/DDLSchema.xsd">

<ConstantsSection label="hcalalgo.xml" eval="true">
 <Constant name="Z0HE"   value="398.250*cm"/>
<Constant name="z6HB"   value="[cms:CalorBeamZ2]"/>
<Constant name="zRat"   value="([Z0HE]-[z6HB])/([cms:CalorBeamZ1]-[z6HB])"/>
<Constant name="rMin1"  value="177.500*cm"/>
</ConstantsSection>

<SolidSection label="hcalalgo.xml">
 <Polycone name="HCal" startPhi="0*deg" deltaPhi="360*deg" >
   <ZSection z="-[z6HB]"      rMin="[cms:CalorBeamR2]" rMax="[cms:CalorMuonR]" />
   <ZSection z="-[Z0HE]"      rMin="[rMin2]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="-[Z0HE]"      rMin="[rMin3]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="-[Z0HBHE]"  rMin="[rMin3]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="-[Z0HBHE]"  rMin="[rMin1]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="[Z0HBHE]"   rMin="[rMin1]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="[Z0HBHE]"   rMin="[rMin3]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="[Z0HE]"       rMin="[rMin3]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="[Z0HE]"       rMin="[rMin2]"           rMax="[cms:CalorMuonR]" />
   <ZSection z="[z6HB]"       rMin="[cms:CalorBeamR2]" rMax="[cms:CalorMuonR]" />
 </Polycone>
</SolidSection>
```

Continuation of Geometry

```
<LogicalPartSection label="hcalalgo.xml">
 <LogicalPart name="HCal" category="unspecified">
  <rSolid name="HCal"/>  <rMaterial name="materials:Air"/>
</LogicalPart>
</LogicalPartSection>

<PosPartSection label="hcalalgo.xml">
 <PosPart copyNumber="1">
   <rParent name="caloBase:CALO"/>
   <rChild name="hcalalgo:HCal"/>
   <rRotation name="rotations:000D"/>
 </PosPart>
 <AlgoPosPart algo="global:angular">
   <rParent name="hcalalgo:HRCF"/>
   <rChild name="hcalalgo:HEC1"/>
   <ParE name="n"          value="8" />
   <ParE name="startCopyNo" value="0" />
   <ParE name="incrCopyNo"  value="1" />
   <ParE name="rangeAngle"  value="140*deg"/>
   <ParE name="startAngle"  value="20*deg"/>
   <ParE name="radius"      value="0"/>
   <ParE name="center"      value="0"/>
   <ParE name="center"      value="0"/>
   <ParE name="center"      value="0"/>
 </AlgoPosPart>
</PosPartSection>
</DDDefinition>
```

Geometry

- ESSource for the XML files is XMLIdealGeometryESSource. There are cfi files to read a number of xml files and prepare the DDD geometry.
- Finally a collection of cfi files are combined into a cff (configuration fragment) to describe simulation/reconstruction geometry

```
import FWCore.ParameterSet.Config as cms
XMLIdealGeometryESSource = cms.ESSource("XMLIdealGeometryESSource",
  geomXMLFiles = cms.vstring('Geometry/CMSCommonData/data/materials.xml',
        "Geometry/HcalCommonData/data/average/hcalforwardmaterial.xml",
        'Geometry/CMSCommonData/data/rotations.xml',
        "Geometry/HcalCommonData/data/hcalrotations.xml",
        'Geometry/CMSCommonData/data/normal/cmsextent.xml',
        'Geometry/CMSCommonData/data/cms.xml',
         'Geometry/CMSCommonData/data/cmsMother.xml',
        'Geometry/CMSCommonData/data/caloBase.xml',
         'Geometry/CMSCommonData/data/cmsCalo.xml',
        'Geometry/HcalCommonData/data/hcalalgo.xml',
         'Geometry/HcalCommonData/data/hcalbarrelalgo.xml',
        'Geometry/HcalSimData/data/CaloUtil.xml',
         'Geometry/HcalSimData/data/HcalProdCuts.xml',
        'Geometry/CMSCommonData/data/FieldParameters.xml'),
   rootNodeName = cms.string('cms:OCMS')
)
```

cfi

```
import FWCore.ParameterSet.Config as cms
from Geometry.EcalTestBeam.APDXML_cfi import *
from Geometry.CaloEventSetup.CaloTopology_cfi import *
from Geometry.CaloEventSetup.CaloGeometry_cff import *
from Geometry.CaloEventSetup.EcalTrigTowerConstituents_cfi import *
from Geometry.EcalMapping.EcalMapping_cfi import *
from Geometry.EcalMapping.EcalMappingRecord_cfi import *
```

cff

CaloGeometry interface

```
class CaloGeometry {
public:

  /// Get the position of a given detector id
  const GlobalPoint& getPosition(const DetId& id) const;

  /// Get the cell geometry of a given detector id
  const CaloCellGeometry* getGeometry(const DetId& id) const;

  /// Get the list of all valid detector ids
  std::vector<DetId> getValidDetIds() const;

  /// Get the list of valid detector ids for the given subdetector
  std::vector<DetId> getValidDetIds(DetId::Detector det, int subdet) const;

  /// is this detid present in the geometry?
  bool present(const DetId& id) const;

  /// access the subdetector geometry for the given subdetector directly
  const CaloSubdetectorGeometry* getSubdetectorGeometry(const DetId& id) const;

  /// access the subdetector geometry for the given subdetector directly
  const CaloSubdetectorGeometry* getSubdetectorGeometry(DetId::Detector det, int subdet) const;

  // Eventually --> Get closest cell, etc...
```

CaloTopology Interface

```
class CaloTopology {
public:
  /// access the subdetector Topology for the given subdetector directly
  const CaloSubdetectorTopology* getSubdetectorTopology(const DetId& id) const;
  /// access the subdetector Topology for the given subdetector directly
  const CaloSubdetectorTopology* getSubdetectorTopology(DetId::Detector det, int subdet) const;
  /** Is this a valid cell id? */
  bool valid(const DetId& id) const;
  /** Get the neighbors of the given cell in east direction*/
  std::vector<DetId> east(const DetId& id) const;
  /** Get the neighbors of the given cell in west direction*/
  std::vector<DetId> west(const DetId& id) const;
  /** Get the neighbors of the given cell in north direction*/
  std::vector<DetId> north(const DetId& id) const;
  /** Get the neighbors of the given cell in south direction*/
  std::vector<DetId> south(const DetId& id) const;
  /** Get the neighbors of the given cell in up direction (outward)*/
  std::vector<DetId> up(const DetId& id) const;
  /** Get the neighbors of the given cell in down direction (inward)*/
  std::vector<DetId> down(const DetId& id) const;
```

East, West, North, South directions have different meanings for different contexts:
EB East is –eta and West is +eta, North is +phi and South is –phi
EE East is +x and West is -x, North is +y and South is –y

How to Use Geometry/Topology

- Access the Geometry and Topology Records from the Event Setup

```
// get handles to calogeometry and calotopology
 edm::ESHandle<CaloGeometry> pG;
 iSetup.get<CaloGeometryRecord>().get(pG);  const CaloGeometry* geo = pG.product();
 const EcalBarrelGeometry *barrelGeom = (dynamic_cast< const EcalBarrelGeometry *>
                                        (geo->getSubdetectorGeometry(DetId::Ecal,EcalBarrel)));

 edm::ESHandle<CaloTopology> theCaloTopology;
 iSetup.get<CaloTopologyRecord>().get(theCaloTopology);
 const CaloTopology *caloTopology = theCaloTopology.product();
 const CaloSubdetectorTopology *barrelTopo =
                                         (caloTopology>getSubdetectorTopology(DetId::Ecal,EcalBarrel));
```

- Utilize them in getting useful information

```
  core = barrelGeom->getPosition(detId);
  const EcalBarrelGeometry::OrderedListOfEEDetId& ol( barrelGeom->getClosestEndcapCells(detId) ) ;


   std::vector<DetId> neighbours = barrelTopo->getNeighbours(detId,dir);
```

Geometry/Topology Class

- Register the class for a type lookup table

```
#include “FWCore/Utilities/interface/typelookup.h
TYPELOOKUP_DATA_REG(HGCalTopology)
```

- Choose a record where the instance for that class will reside
  - If the object depends on some other object, the new object has to reside in the same record or one for which the dependency has to be clearly stated

```
# include "FWCore/Framework/interface/EventSetupRecordImplementation.h“
# include "FWCore/Framework/interface/DependentRecordImplementation.h“
# include "Geometry/Records/interface/IdealGeometryRecord.h“
# include "Geometry/Records/interface/ShashlikNumberingRecord.h“
class ShashlikGeometryRecord :   public edm::eventsetup::
DependentRecordImplementation<  ShashlikGeometryRecord,  boost::mpl::vector<
   IdealGeometryRecord,    ShashlikNumberingRecord > > {};

 #include "Geometry/Records/interface/ShashlikGeometryRecord.h“
 #include "FWCore/Framework/interface/eventsetuprecord_registration_macro.h“
 EVENTSETUP_RECORD_REG( ShashlikGeometryRecord );
```

- Provide a class to build an object of Geometry/Topology class and attach this to the EventSetup
  - Provide the cfi file to activate the ESProducer

Example of an ESProducer

- Header for a typical ESProducer

```
#include <FWCore/Framework/interface/ModuleFactory.h>
#include "FWCore/Framework/interface/ESProducer.h“
#include "FWCore/Framework/interface/ESHandle.h“
#include "FWCore/ParameterSet/interface/ParameterSet.h“
#include "Geometry/CaloTopology/interface/HGCalTopology.h“
#include "Geometry/CaloTopology/interface/CaloSubdetectorTopology.h“
#include "Geometry/HGCalCommonData/interface/HGCalDDDConstants.h“
#include "Geometry/Records/interface/IdealGeometryRecord.h“
#include "Geometry/CaloTopology/interface/CaloSubdetectorTopology.h“
#include "DataFormats/ForwardDetId/interface/ForwardSubdetector.h“
class HGCalTopologyBuilder : public edm::ESProducer {
public:
  HGCalTopologyBuilder( const edm::ParameterSet& iP );
 ~HGCalTopologyBuilder() ;
 typedef boost::shared_ptr< HGCalTopology > ReturnType;
  ReturnType produce(const IdealGeometryRecord&);
private:
    std::string       name_;
     bool             halfType_;
    ForwardSubdetector subdet_;
};
```

- Implementation:

```
HGCalTopologyBuilder::HGCalTopologyBuilder(
  const edm::ParameterSet& iConfig) {
  name_     = iConfig.getUntrackedParameter<std::string>("Name");
  int type  = iConfig.getUntrackedParameter<int>("Type");
  halfType_ = iConfig.getUntrackedParameter<bool>("HalfType");
  if (type == 0) {subdet_ = HGCEE;} else if (type == 1) { subdet_ = HGCHEF;}
  else if (type == 2) {subdet_ = HGCHEB;}  else { subdet_ = HGCHET;}
 setWhatProduced(this, name_);
}
HGCalTopologyBuilder::~HGCalTopologyBuilder() { }
HGCalTopologyBuilder::ReturnTypeHGCalTopologyBuilder::produce(const IdealGeometryRecord& iRecord ) {
  edm::ESHandle<HGCalDDDConstants>  pHGDC;
  iRecord.get(name_, pHGDC) ;
  const HGCalDDDConstants & hgdc = (*pHGDC);
   ReturnType ct ( new HGCalTopology(hgdc, subdet_, halfType_) );
   return ct ;
}
DEFINE_FWK_EVENTSETUP_MODULE(HGCalTopologyBuilder);
```

How to retrieve Data from the event?

- Access using the GetByLabel method of Event:

```
edm::Handle<EBUncalibratedRecHitCollection> barrelRecHitsHandle;
iEvent.getByLabel("ecalUncalibHit","EcalUncalibRecHitsEB",barrelRecHitsHandle);
const EBUncalibratedRecHitCollection * recHitsEB = 0;
if (barrelRecHitsHandle.isValid())  recHitsEB = barrelRecHitsHandle.product();
```

- Iterate over the content and do according to the object type

```
EBUncalibratedRecHitCollection::const_iterator it;
 for (it  = recHitsEB->begin(); it != recHitsEB->end(); ++it) {
   DetId id  = it->id();
   int eta   =  (((EBDetId)(id)).ietaAbs());
   int phi   =  (((EBDetId)(id)).iphi());
   double ph = (it->amplitude());
```

Faster way to access data

- Use edm::EDGetToken instead of edm::InputTag or a set of std::string at the time of event loop
- At initialization time get the Token from edm::InputTag using

```
edm::EDGetToken<EcalRecHitCollection> tokenEB_;
tokenEB_ = consumes<EcalRechItCollection>(edm::InputTag("EcalRecHitsEB"));
```

- It assumes by default the block will be on the Event branch. If it is not, like Luminosity, need to specify the "TypeToGet"

```
edm::EDGetToken<LumiDetails>  tokenLumi;
tokenLumi_ = consumes<LumiDetails,edm::InLumi>(edm::InputTag("lumiproducer"));
```

- At execution time simply replace the GetByLabel statement

```
edm::Handle<EcalRecHitCollection> barrelRecHitsHandle;
iEvent.getByToken(tokenEB_, barrelRecHitsHandle);

Edm::Handle<LumiDetails> lumiD;
iEvent.getLuminosityBlock().getByYoken(tokenLumi_,lumiD);
```

Physics Objects

- Most of the physics objects are derived from some basic classes: 3-4 levels of dependency
- Take the example of Gsf Electrons:

```
namespace reco {
class GsfElectron : public RecoCandidate  {
public :
 // some nested structures defined later on
    struct ChargeInfo {
      int scPixCharge ;
      bool isGsfCtfScPixConsistent ;
      bool isGsfScPixConsistent ;
      bool isGsfCtfConsistent ;
      ChargeInfo()        : scPixCharge(0), isGsfCtfScPixConsistent(false),
          isGsfScPixConsistent(false), isGsfCtfConsistent(false)        {}
     } ;

  struct TrackClusterMatching ;
      ……..

    GsfElectron() ;
    virtual ~GsfElectron() {} ;
    int scPixCharge() const;
    bool isGsfCtfScPixChargeConsistent() const;
    bool trackerDrivenSeed() const;
    SuperClusterRef pflowSuperCluster();
    ……..
} ;
} // namespace reco
```

Many of the properties are described in the base class

Going one level deeper

- RecoCandidate is derived from LeafCandidate class
- Only a partial list of methods are listed here

```
namespace reco {
  class RecoCandidate : public LeafCandidate {
  public:
    RecoCandidate() : LeafCandidate() { }
    RecoCandidate( Charge , const LorentzVector &, const Point &, int pdgId, int status) ;
    virtual ~RecoCandidate();

    virtual bool overlap( const Candidate & ) const = 0;
    virtual RecoCandidate * clone() const ;
    virtual reco::TrackRef track() const;
    virtual reco::SuperClusterRef superCluster() const;
    virtual CaloTowerRef caloTower() const;
    const Track * bestTrack() const;
    TrackBaseRef bestTrackRef() const;
    enum TrackType { noTrackType, recoTrackType, gsfTrackType };
    TrackType bestTrackType() const;TrackType );
  }
```

LeafCandidate Class

- Derived from the Candidate class

```
namespace reco {
  class LeafCandidate : public Candidate {
  public:
    LeafCandidate();
    explicit LeafCandidate( const Candidate & c);
    virtual ~LeafCandidate();

    virtual const_iterator begin() const;
    virtual const_iterator end() const;
    virtual iterator begin();
    virtual iterator end();
    virtual size_t numberOfDaughters() const;
    virtual const Candidate * daughter( size_type ) const;
    virtual size_t numberOfMothers() const;
    virtual const Candidate * mother( size_type ) const;
    virtual int charge() const;
    virtual const LorentzVector & p4() const;
    virtual math::XYZVector momentum() const;
    virtual math::XYZVector boostToCM() const;
    virtual double p() const;
    virtual double energy() const;
    virtual double eta() const;
    virtual const math::XYZPoint & vertex() const;
```

A concrete class with all methods implemented

Candidate Class

- A pure abstract class – only provides the signatures

```
namespace reco {
  class Candidate {
  public:
    Candidate() {};
    virtual ~Candidate();
    /// electric charge
    virtual int charge() const = 0;
    virtual const LorentzVector & p4() const = 0;
    virtual math::XYZVector momentum() const = 0;
    virtual math::XYZVector boostToCM() const = 0;
    virtual double p() const = 0;
    virtual double energy() const = 0;
    virtual double et() const = 0;
    virtual double mass() const = 0;
    virtual double eta() const = 0;
    virtual const math::XYZPoint & vertex() const  = 0;
    virtual int pdgId() const  = 0;
```

Again a partial list

Conditions from the DB

```
es_source = PoolDBESSource { VPSet toGet = {
                        {string record = "EcalPedestalsRcd"
                         string tag = "EcalPedestals"},
                        {string record = "EcalTBWeightsRcd"
                         string tag = "EcalTBWeights"},
                        {string record = "EcalGainRatiosRcd"
                         string tag = "EcalGainRatios"},
                        {string record = "EcalIntercalibConstantsRcd"
                         string tag = "EcalIntercalibConstants"},
                        {string record = "EcalADCToGeVConstantRcd"
                         string tag = "EcalADCToGeVConstant"},
                        {string record = "EcalWeightXtalGroupsRcd"
                         string tag = "EcalWeightXtalGroups"}}
                         bool loadAll = true
                       string connect =
"sqlite_file:/afs/cern.ch/cms/ECAL/testbeam/pedestal/2004/ecal2004condDB.db"
                       untracked string catalog =
"file:/afs/cern.ch/cms/ECAL/testbeam/pedestal/2004/PoolFileCatalog.xml"
                      string timetype = "runnumber“  }
```

You need to provide list of Records and tags (tags can identify for example intercalibration)

```
module get = EventSetupRecordDataGetter { VPSet toGet =
     {      {string record = "EcalPedestalsRcd"
            vstring data = {"EcalPedestals"} },
            {string record = "EcalTBWeightsRcd"
            vstring data = {"EcalTBWeights"} },
            {string record = "EcalGainRatiosRcd"
             vstring data  = {"EcalGainRatios"}},
            {string record = "EcalIntercalibConstantsRcd"
             vstring data  = {"EcalIntercalibConstants"}},
            {string record = "EcalADCToGeVConstantRcd"
             vstring data  = {"EcalADCToGeVConstant"},
                           {string record = "EcalWeightXtalGroupsRcd"
            vstring data = {"EcalWeightXtalGroups"} } } }
```

This is the module which creates the EventSetup. Need to be added to scheduler path

S. Banerjee64

Hardcoded/default conditions

es_source = EcalTrivialConditionRetriever {}

- This will load hardcoded/default conditions for
  - Pedestals
  - Intercalibration
  - Weights
  - Default gain ratios
- All values are configurable in the config file:

```
double adcToGeVEBConstant
double intercalibConstant(Mean,Sigma)
double pedMean(X12,X6,X1)
double pedRMS(X12,X6,X1)
double gainRatio12over6
double gainRatio6over1
std::vector<double> amplWeights
std::vector<double> pedWeights
std::vector<double> jittWeights
```

- To be updated to support also the Endcap

How to Write Imperfect Code

- We are often in a hurry and try to write code without thinking through the process deep enough and we land up in a code which is difficult for us to understand even after one week
- Give an example: the purpose of the code
  - Test the electronics map in a setup
  - Look at the trigger counters and determine thresholds for good beam and differentiate muons from hadrons
  - Determine the pedestals of all channels and study their stability
  - Study the LED runs and find the # of pixels fired and gain factor
  - Look at the calibration runs with muons and find the calibration constants of HCAL towers by fitting Gaussian-convoluted Landau functions
  - Find the pedestal shift for EB crystals
  - Determine energy scale for ECAL
  - Find the weight factors for combining ECAL/HCAL ADC's in energy
  - Study the effect of saturation in the SiPM readout channels using pion beam data

Can you Avoid it?

- The problem is not too complex – but there was little time to think and writing the code started after data taking started and the results need to be reported as the data taking were going on.
- Result is that people wrote
  - 1300 lines long analyzer
  - Parameter set with 48 different parameters
  - 4 external Adhoc text files with constants
  - 1200 lines long ROOT Macro and 600 lines long tree analyzer
- Generic coding was tried with templated function – but a producer-analyzer option could have been used with creation of intermediate transient data and utilization of the data bus model to cut down the analyzer code and showing more transparency in the usage.
- Moral of the story
  - Think about the problem before writing a code
  - Do not use somebody's code blindly
  - Try to fragment the code in small chunks using sharing of responsibilities

How to Find the Central Crystal

- RunList provides the η,φ of the table position which corresponds to trigger tower – translate that to η,φ of crystal:

```
// Retrieve trigger tower map
edm::ESHandle<EcalTrigTowerConstituentsMap> hTtmap;
iSetup.get<IdealGeometryRecord>().get(hTtmap);
const EcalTrigTowerConstituentsMap& ttMap = *hTtmap;
// Get the central ECAL tower
int phiTowerEB = phiTower-3;
if (phiTowerEB <= 0) phiTowerEB += 72;
int etaTowerEB = etaTower;
if (etaTowerEB <= 0) etaTowerEB = 1;
EcalTrigTowerDetId trId = EcalTrigTowerDetId(-1,EcalBarrel,etaTowerEB,phiTowerEB);
vdets = ttMap.constituentsOf(trId);
etaEB = phiEB = 0;
for (unsigned int i=0; i<vdets.size(); ++i) {
  EBDetId id = vdets[i];
  etaEB += id.ietaAbs();
  phiEB += id.iphi();
}
etaEB /= vdets.size();
phiEB /= vdets.size();
```

Use Trigger Information

- Use the trigger information to set good beam condition and select between hadron/muon

Trigger Type

```
edm::Handle<HcalTBTriggerData> trig;
iEvent.getByLabel(hcalTrigLabel_,trig);
if (!trig.isValid()) {
  edm::LogError("TB2011Analyzer::analyze") << "No Trigger Data found";
  return;
}
const HcalTBTriggerData& trigdata = *trig;
if       (trigdata.wasLEDTrigger())                nLED++;
else if (trigdata.wasOutSpillPedestalTrigger()) nPed++;
else if (trigdata.wasBeamTrigger())                nBeam++;
else                                               nOther++;
```

Particle Type

```
edm::Handle<HcalTBBeamCounters> qadc;
iEvent.getByLabel(hcalTrigLabel_,qadc);
if (qadc.isValid()) {
  if (trigdata.wasBeamTrigger()) {
    if (qadc->S1adc()>s1Cut && qadc->S2adc()>s2Cut && qadc->S3adc()>s3Cut &&
        qadc->S4adc()>s4CutL && qadc->S4adc()<s4CutH && qadc->BH1adc()<bh1Cut &&
        qadc->BH2adc()<bh2Cut && qadc->BH3adc()<bh3Cut && qadc->BH4adc()<bh4Cut )   beam = true;
    if (beam) {
      if (qadc->VMBadc()>vmbMuCut && qadc->VMFadc()>vmfMuCut) muonSelected = true;
      if (qadc->VMBadc()<vmbPiCut && qadc->VMFadc()<vmfPiCut)    hadronSelected = true;
    }
```

Analyze data from SubDetector

- Need to access some data from EventSetup as well as from current Event to do a meaningful analysis

Get HCAL Condititons from EventSetup

```
// get conditions
 edm::ESHandle<HcalDbService> conditions;
 iSetup.get<HcalDbRecord>().get(conditions);
```

Get Digi Data for HB and enter analysis chain

```
 edm::Handle<HBHEDigiCollection> hbhedg;
  iEvent.getByLabel(hcalDigiLabel_,hbhedg);
  if (!hbhedg.isValid()) {
    edm::LogWarning("TB2011Analyzer::analyze") << "HBHE Digis not found";
  } else {
    const HBHEDigiCollection& hbhedigic = *hbhedg;
    analyzeHB(hbhedigic, conditions, trigdata);
  }
```

Main Analysis Loop

```
TB2011Analyzer::pulseHeightResults ph;
 const HcalQIEShape* shape = conditions->getHcalShape ();
 int ADC = 0;
 HBHEDigiCollection::const_iterator it;
 for (it  = hbhedigic.begin();it != hbhedigic.end();++it) {
   HcalDetId id = it->id();
   int idx = histoID(id);
   const HcalQIECoder* channelCoder = conditions->getHcalCoder (id);
   HcalCoderDb coder (*channelCoder, *shape);
   TB2011AnalyzerCode::getPH((*it), coder, useCoder, maxBin, debug, ph);
   double pedSub=0, ampl=0;
   if (doPeds) {
     pedSub = pedSubtracted(ph, conditions);
     if (doGain) ampl = gainCorrected(ph, conditions);
   }
```

Codes Common to HB and HO

```
namespace TB2011AnalyzerCode {
  template<class Digi>
  inline void getPH(const Digi& digi, const HcalCoder& coder,
                    bool useCoder, int maxbin, int debug,
                    TB2011Analyzer::pulseHeightResults& pulseHeight) {
    pulseHeight.cell = digi.id();
    pulseHeight.ped4 = pulseHeight.signal4 = pulseHeight.signal10 = 0;
    pulseHeight.imx = 1;
    if (maxbin > 0) pulseHeight.imx = (unsigned int)(maxbin);
    double binmx = -1;
    if (useCoder) {
      CaloSamples tool;
      coder.adc2fC(digi,tool);
      pulseHeight.nTools = tool.size();
      for(unsigned int bin=0; bin<pulseHeight.nTools; bin++){
        pulseHeight.capid[bin] = digi[bin].capid();
        pulseHeight.tools[bin] = tool[bin];
        pulseHeight.signal10  += tool[bin];
        if (bin<4) pulseHeight.ped4 += tool[bin];
        if (tool[bin] > binmx && maxbin <= 0) {
          pulseHeight.imx = bin; binmx = tool[bin];
        } else if (maxbin > 0 && maxbin == (int)(bin)) {
           binmx = tool[bin];
        }
      }
    } else {
        pulseHeight.nTools = digi.size();
        for(unsigned int bin=0; bin<pulseHeight.nTools; bin++){
          pulseHeight.capid[bin] = digi[bin].capid();
          double binvalue = (double)(digi[bin].adc());
          pulseHeight.tools[bin] = binvalue;
          pulseHeight.signal10  += binvalue;
          if (bin<4) pulseHeight.ped4 += binvalue;
           if (binvalue > binmx && maxbin <= 0) {
            pulseHeight.imx = bin; binmx = binvalue;
           } else if (maxbin > 0 && maxbin == (int)(bin)) {
              binmx = binvalue;
           }
        }
    }
  }
}
```