Table of Contents

Analysis recipe to use CASTOR objects with 2010 CMS

Commissioning10 data period

Introduction

This section is valid for 900 GeV and 7 TeV datasets taken during the Commissioning10 period in 2010. More specifically for data (OpenData records here):

```
/MinimumBias/Commissioning10-07JunReReco_900GeV/RECO
/MinimumBias/Commissioning10-May19ReReco-v1/RECO
```

with JSON file run selection specifically for usage with CASTOR data (note: these contain less runs):
https://twiki.cern.ch/twiki/pub/CMSPublic/CASTOROpenData2010/Commissioning10-May19ReReco_900GeV.json
https://twiki.cern.ch/twiki/pub/CMSPublic/CASTOROpenData2010/Commissioning10-May19ReReco_7TeV.json

And for MC:

```
/MinBias_TuneZ2_900GeV_pythia6_cff_py_GEN_SIM_START311_V2_Dec11_v2/hvanhaev-MinBias_TuneZ2_900GeV
/MinBias_TuneZ2_7TeV_pythia6_cff_py_GEN_SIM_START311_V2_Dec11_v2/hvanhaev-MinBias_TuneZ2_7TeV_pyt
/MinBias_Tune4C_900GeV_pythia8_cff_py_GEN_SIM_START311_V2_Dec11_v2/hvanhaev-MinBias_Tune4C_900GeV
/MinBias_Tune4C_7TeV_pythia8_cff_py_GEN_SIM_START311_V2_Dec11_v1/hvanhaev-MinBias_Tune4C_7TeV_pyt
```

The OpenData records can be found in this list

Installation and setup

Before you can analyse data taken with the CASTOR calorimeter you'll need install a few extra packages into the CMSSW software.
1) Install the CERN OpenData VM for 2010 data, for instructions see here: http://opendata.cern.ch/docs/cms-virtual-machine-2010. The recipe below is validated in version 'CMS-OpenData-1.1.2'.
2) Start up your VM and open the 'CMS shell'.
3) Install the correct version of CMSSW and activate it:

```
cmsrel CMSSW_4_2_8_lowpupatch1
cd CMSSW_4_2_8_lowpupatch1/src
cmsenv
```

4) When in CMSSW_4_2_8_lowpupatch1/src directory, download the additional necessary packages and compile them:

```
wget --no-check-certificate https://twiki.cern.ch/twiki/pub/CMSPublic/CASTOROpenData2010/Commissi
tar -xvf Commissioning10_additional_packages.tar
scram b
```

Note that following directories should appear in the CMSSW_4_2_8_lowpupatch1/src directory: RecoLocalCalo, RecoJets, data. If this is not the case then something went wrong when downloading/extracting the packages.

Usage

Once the above steps are executed you are basically ready to analyse CASTOR objects. Here we will specify what configuration snippets you will need to apply inside your typical 'demoanalyzer_cfg.py' file when you are coding an analyser in CMSSW.

First **important** step is to include a 'global tag' into your configuration file. To do this, please follow the instructions on this CMS OpenData page: http://opendata.cern.ch/docs/cms-guide-for-condition-database and follow the steps in section 'For 2010 collision data' to include the 'FT_R_42_V10A::All' global tag when you run on **data**. When you want to run on a **Monte Carlo** simulation sample, please make sure to follow the proper steps to include the 'START42_V17B::All' global tag in the configuration file.

Running on data

1) Add the following lines to the cfg file:

```
# load latest ChannelQuality conditions to remove the bad channels
process.es_ascii = cms.ESSource("CastorTextCalibrations",
    input = cms.VPSet(
                cms.PSet(
                    object = cms.string('ChannelQuality'),
                    file = cms.FileInPath('./data/castor_db2013/DumpCastorCondChannelQuality_Run1
                ),
    )
)
process.es_prefer_castor = cms.ESPrefer('CastorTextCalibrations','es_ascii')

# import correct treatment of CASTOR objects
process.load('RecoLocalCalo.Castor.ReReco_data_cff')

# filter bad data
process.castorInvalidDataFilter = cms.EDFilter("CastorInvalidDataFilter")
```

2) If process.demo is the actual process that executes your analysis code, then you need to extend the cms.Path() that contains it until you have the following:

```
cms.Path(process.castorInvalidDataFilter*process.CastorReReco*process.demo)
```

It is important that process.demo is at the end of the path, and that the order of the path is respected. Note that the above 2 extensions in the cfg file of the analyser are always needed when analysing CASTOR objects in Commissioning10 data.

Running on Monte Carlo samples

1) Add the following lines to the cfg file:

```
# load latest ChannelQuality conditions to remove the bad channels
process.es_ascii = cms.ESSource("CastorTextCalibrations",
    input = cms.VPSet(
                cms.PSet(
                    object = cms.string('ChannelQuality'),
                    file = cms.FileInPath('./data/castor_db2013/DumpCastorCondChannelQuality_Run1
                ),
    )
)
process.es_prefer_castor = cms.ESPrefer('CastorTextCalibrations','es_ascii')

# import correct treatment of CASTOR objects
process.load('RecoLocalCalo.Castor.ReReco_MC_cff')
```

2) If process.demo is the actual process that executes your analysis code, then you need to extend the cms.Path() that contains it until you have the following:

```
cms.Path(process.CastorReReco*process.demo)
```

It is important that process.demo is at the end of the path, and that the order of the path is respected. Note that the above 2 extensions in the cfg file of the analyser are always needed when analysing CASTOR objects in 2010 Monte Carlo samples.

Demo analysis

With this demo analysis you'll be able to cross check that everything is behaving as intended

class

A demo analyser C++ class can be downloaded by:

```
wget --no-check-certificate https://twiki.cern.ch/twiki/pub/CMSPublic/CASTOROpenData2010/DemoAnal
```

It contains code to read out all objects and make basic validation histograms. You can run it with the following python configuration files:

```
wget --no-check-certificate https://twiki.cern.ch/twiki/pub/CMSPublic/CASTOROpenData2010/demoanal
wget --no-check-certificate https://twiki.cern.ch/twiki/pub/CMSPublic/CASTOROpenData2010/demoanal
```

The first one is used with data, the second with MC samples. Before using remove the .txt file extension.

The demo code applies first some 'online' selections in the python configuration file (physics declared, remove beam background, remove CASTOR invalid data, trigger selection) and in the C++ class a further offline event selection is included that selects nondiffractive minimum bias events by requiring: exactly one good primary vertex, HF AND activity, and CASTOR activity. This event selection is the same as applied in the FWD-11-003 analysis (arXiv:1302.2394).

Validation plots

After running the demo analyser on the 7 TeV data samples you can plot the histograms stored in the output ROOT files. In this attachment you can find validation plots, if you get the same results your framework is working as expected.

Information valid for all 2010 data

Examples to access all objects and their properties

Access

```
  // acces the CASTOR rechits
  Handle<CastorRecHitCollection> rechitcoll;
  iEvent.getByLabel("rechitcorrector",rechitcoll);

  //-- loop over the rechit collection
  if(rechitcoll.isValid()) {
       for(size_t i = 0; i < rechitcoll->size(); ++i) {
         const CastorRecHit & rh = (*rechitcoll)[i];
         HcalCastorDetId castorid = rh.id();

         hRecHit_map->Fill(castorid.sector(),castorid.module());
```

```
        hRecHit_module->Fill(castorid.module(),rh.energy());
        hRecHit_sector->Fill(castorid.sector(),rh.energy());
                       int rh_channel = 16*(castorid.module()-1) + castorid.sector();
        hRecHit_channel->Fill(rh_channel,rh.energy());
        hRecHit_energy->Fill(rh.energy());
    }
  }
```

Access

```
// access the CASTOR towers
edm::Handle<CastorTowerCollection> towercoll;
iEvent.getByLabel("CastorTowerReco",towercoll);

  if(towercoll.isValid()) {

    hTower_multi->Fill(towercoll->size());
    for(unsigned int i=0;i<towercoll->size();i++) {
      const CastorTower & castortower = (*towercoll)[i];
      hTower_energy->Fill(castortower.energy());
      hTower_phi->Fill(castortower.phi());
      hTower_fem->Fill(castortower.fem());
      hTower_eem->Fill(castortower.emEnergy());
      hTower_ehad->Fill(castortower.hadEnergy());
      hTower_depth->Fill(castortower.depth());
      hTower_fhot->Fill(castortower.fhot());
      hTower_ncell->Fill(castortower.rechitsSize());

    }
  }
```

Access

```
  // access the CASTOR jets
  edm::Handle<edm::View< reco::BasicJet > > basicjetcoll;  //-- uncorrected jets
  edm::Handle<reco::CastorJetIDValueMap> jetIdMap;

  iEvent.getByLabel("ak5BasicJets",basicjetcoll);
  iEvent.getByLabel("ak5CastorJetID",jetIdMap);

  if(basicjetcoll.isValid()) {

    for(edm::View<reco::BasicJet>::const_iterator ibegin = basicjetcoll->begin(), iend = basicjet

      unsigned int idx = ijet - ibegin;
      const BasicJet & basicjet = (*basicjetcoll)[idx];

      hJet_energy->Fill(basicjet.energy());
      hJet_phi->Fill(basicjet.phi());

      edm::RefToBase<reco::BasicJet> jetRef = basicjetcoll->refAt(idx);
      reco::CastorJetID const & jetId = (*jetIdMap)[jetRef];

      hJet_fem->Fill(jetId.fem);
      hJet_eem->Fill(jetId.emEnergy);
      hJet_ehad->Fill(jetId.hadEnergy);

      hJet_width->Fill(jetId.width);
      hJet_depth->Fill(jetId.depth);
      hJet_fhot->Fill(jetId.fhot);
      hJet_sigmaz->Fill(jetId.sigmaz);
      hJet_ntower->Fill(jetId.nTowers);

    }
  }
```

Jet calibration factors for Commissioning10 data

Below you can find the formulas to be used when calibrating jets in CASTOR that are identified as coming from a hadronic shower in the calorimeter. The important functions are:

```
// for jets in sector 5 for Commissioning10 data
Ecal = myJETenergy*(1.3 + 0.23*log(-149 + myJETenergy)); // data or MC (SL)
Ecal = myJETenergy*(0.9 + 0.25*log(-149 + myJETenergy)); // MC (FS)

// for jets in sector 6 for Commissioning10 data
Ecal = myJETenergy*(1.4 + 0.21*log(-149 + myJETenergy)); // data or MC (SL)
Ecal = myJETenergy*(-5.8 + 1.3*log(144 + myJETenergy)); // MC (FS)

// for other good sectors for Commissioning10 data
Ecal = myJETenergy*(1.8 + 0.0037*log(1 + myJETenergy)); // data or MC (SL)
Ecal = myJETenergy*(1.5 + 0.0024*log(1 + myJETenergy)); // MC (FS)
```

The functions marked to be used with Shower Library (SL) need to be applied to data and to MC samples using SL, **such as the official public ones**. Functions marked with Full Simulation (FS) can only be used with special MC samples simulated at displaced sensor locations. These latter samples are not public (for the moment).

Full code as in the demo analyser that first identifies electromagnetic or hadronic showers using the JetID variables, and then properly calibrates the hadronic-like jets:

```
// access the CASTOR jets
  edm::Handle<edm::View< reco::BasicJet > > basicjetcoll;  //-- uncorrected jets
  edm::Handle<reco::CastorJetIDValueMap> jetIdMap;

  iEvent.getByLabel("ak5BasicJets",basicjetcoll);
  iEvent.getByLabel("ak5CastorJetID",jetIdMap);

  if(basicjetcoll.isValid()) {

    for(edm::View<reco::BasicJet>::const_iterator ibegin = basicjetcoll->begin(), iend = basicjet

      unsigned int idx = ijet - ibegin;
      const BasicJet & basicjet = (*basicjetcoll)[idx];

      double myJETenergy = basicjet.energy();

      hJet_energy->Fill(myJETenergy);
      hJet_phi->Fill(basicjet.phi());

      edm::RefToBase<reco::BasicJet> jetRef = basicjetcoll->refAt(idx);
      reco::CastorJetID const & jetId = (*jetIdMap)[jetRef];

      hJet_fem->Fill(jetId.fem);
      hJet_eem->Fill(jetId.emEnergy);
      hJet_ehad->Fill(jetId.hadEnergy);

      hJet_width->Fill(jetId.width*(180/myPI)); // convert from radians to degrees
      hJet_depth->Fill(jetId.depth);
      hJet_fhot->Fill(jetId.fhot);
      hJet_sigmaz->Fill(jetId.sigmaz);
      hJet_ntower->Fill(jetId.nTowers);

      // perform jet noncompensation calibration
      // select hadronic jet
      bool HAD = true;
      if (jetId.depth > -14450 && myJETenergy < 175) HAD = false;
      if (jetId.depth > -14460 && myJETenergy > 175) HAD = false;
      if (jetId.fem > 0.95) HAD = false;
```

```
      // select EM jet
      bool EM = true;
      if (jetId.fhot < 0.45) EM = false;
      if (jetId.fem < 0.90) EM = false;
      if (jetId.sigmaz > 30 && myJETenergy < 75) EM = false;
      if (jetId.sigmaz > 40 && myJETenergy > 75) EM = false;
      if (jetId.width*(180/myPI) > 11.5) EM = false; // convert from radians to degrees
      if (jetId.depth < -14450 && myJETenergy < 125) EM = false;
      if (jetId.depth < -14460 && myJETenergy > 125) EM = false;

      // calibrate only hadronic jets
      if (HAD) {
      // apply calibration factors
      double Ecal = 0.0;
      if (basicjet.phi() > 4*(myPI/8) && basicjet.phi() < 5*(myPI/8)) {
         // include different sectors 5 for Commissioning10 data were first channels are removed
              Ecal = myJETenergy*(1.3 + 0.23*log(-149 + myJETenergy)); // data (SL)
      } else if (basicjet.phi() > 5*(myPI/8) && basicjet.phi() < 6*(myPI/8)) {
         // include different sectors 6 for Commissioning10 data were first channels are removed
          Ecal = myJETenergy*(1.4 + 0.21*log(-149 + myJETenergy)); // data (SL)
      } else {
          Ecal = myJETenergy*(1.8 + 0.0037*log(1 + myJETenergy)); // data (SL) good sectors
      }
      hJet_calenergy->Fill(Ecal); // fill with calibrated HAD jet
     } else {
      hJet_calenergy->Fill(myJETenergy); // fill with other jets
     }

    }
  }
```

-- HansVanHaevermaet - 2018-09-10

This topic: CMSPublic > CASTOROpenData2010
Topic revision: r11 - 2019-08-09 - HansVanHaevermaet