Table of Contents

Cheatsheet: CMSSW Analysis a la Framework

An unofficial information dump page for CMSSW. In below there's older and newer information together, I tried to separate them whenever I can.

CMSSW

Accessing physics objects

Electrons (PAT/MiniAOD)

Wiki pages	WorkBookPATDataFormats#PatElectron, WorkBookElectronAnalysis	
Add to buildfile	`<use name="DataFormats/PatCandidates"/>`	Related error: `undefined reference to `pat::Electron::electronID(std::string const&) const`
Header file(s)	`#include <DataFormats/PatCandidates/interface/Electron.h>`	
Label	`slimmedElectrons`	
Collection type	`pat::ElectronCollection` (`vector<pat::Electron>`)	
Object reference	link	
Notes	To see available electron IDs in data, type a wrong ID name in the `electronID` method and framework will list them along with the error message.	

Show legacy information Hide legacy information

Generator Particles

Wiki page	WorkBookGenParticleCandidate
Add to buildfile	`<use name=SimDataFormats/HepMCProduct>`
Header file(s)	`#include "DataFormats/Candidate/interface/Candidate.h"` and `#include "DataFormats/JetReco/interface/GenJet.h"` if you need GEN jets
Label	`genParticleCandidates`
Collection type	`CandidateCollection`
Physics Object	`Candidate` GenParticle

Electrons

Wiki page	WorkBookElectronReco
Add to buildfile	`<use name=DataFormats/EgammaReco>`
Header file(s)	`#include "DataFormats/EgammaCandidates/interface/PixelMatchGsfElectron.h"`
Label	`pixelMatchGsfElectrons`
Collection type	`PixelMatchGsfElectronCollection`
Physics Object	PixelMatchGsfElectron

Photons

Wiki page	WorkBookPhotonReco
Add to buildfile	`<use name=DataFormats/EgammaReco>`
Header file(s)	`#include "DataFormats/EgammaCandidates/interface/Photon.h"`
Label	`correctedPhotons` or `photons`
Collection type	`PhotonCollection`
Physics object	reco::Photon

Jets

Wiki page	WorkBookJetAnalysis
Add to buildfile	`<use name=DataFormats/JetReco>`
Header file(s)	`#include "DataFormats/JetReco/interface/CaloJet.h"` `#include "DataFormats/JetReco/interface/GenJet.h"`
Label	Many, e.g. `iterativeCone5CaloJets` `midPointCone5GenJets`
Collection type	`CaloJetCollection` `GenJetCollection`
Physics object	reco::Jet

Muons

Wiki page	WorkBookMuonAnalysis
Add to buildfile	`<use name=DataFormats/MuonReco>` `<use name=RecoMuon/TrackingTools>`
Header file(s)	`#include "DataFormats/MuonReco/interface/Muon.h"`
Label	`muons`
Collection type	`MuonCollection`
Physics Object	reco::Muon
Example code	available

See this hypernews thread for the muons with fast simulation.

MET

Wiki page	???
Add to buildfile	`<use name=DataFormats/METReco>`
Header file(s)	#include "DataFormats/METReco/interface/CaloMETCollection.h"
Label	`met`
Collection type	`CaloMETCollection`
Physics Object	reco::CaloMET
Example code	Below, or in SusyAnalyzer

Other	`Handle<CaloMETCollection> met;`
	`iEvent.getByLabel("met", met);`
	`const CaloMET calomet = met->front();`
	`cout << "met px=" << calomet.px() << ", py=" << calomet.py() << endl;`

Vertices

Wiki page	WorkBookOfflinePrimaryVertexFinding
Add to buildfile	???
Header file(s)	`#include <DataFormats/VertexReco/interface/Vertex.h>` `#include <DataFormats/VertexReco/interface/VertexFwd.h>`
Label	`offlinePrimaryVerticesFromCTFTracks` and `offlinePrimaryVerticesFromRSTracks`
Collection type	VertexCollection
Physics object	reco::Vertex
Example code	

Tracks

Physics Object	reco::Track

CaloTowers

These are combined objects from ECAL cells and HCAL towers, they represent the whole calorimetry and are used for e.g. jet reconstruction.

Wiki page	
Header file(s)	#include "DataFormats/CaloTowers/interface/CaloTowerCollection.h"
Label	`towerMaker`
Collection type	`CaloTowerCollection`
Physics object	CaloTower
Example code	
Info	1

- More labels, collections and documentation available here.
- Namespace documentations: reco, edm.

"Handling" the data

Main article

One accesses the data in the root file thru "handles". Handles types are templated with the collection/data type in the EDM root file. `EdmDumpEvent` prints the collection types in the first column of it's output.

In the class definition, define as private the token

```
edm::EDGetTokenT< pat::ElectronCollection > tok_electrons_;
```

In the constructor assign it's value

```
std::string labelElec_("slimmedElectrons");
tok_electrons_ = consumes< pat::ElectronCollection >(edm::InputTag(labelElec_));
```

In the analysis code, handle it

```
Handle<pat::ElectronCollection> elec;
iEvent.getByToken( tok_electrons_, elec );
```

Show legacy information Hide legacy information
Taken from Benedikt's presentation:

```
# by module and default product label
Handle<TrackVector> trackPtr;
iEvent.getByLabel("tracker", trackPtr );

# by module and product label
Handle<MuonCollection>trackPtr;
string muonModule_ = "paramMuons"
string muonProduct_ = "ParamGlobalMuons"
iEvent.getByLabel(muonModule_, muonProduct_, muons);

# by type (what happens if not unique?)
vector<Handle<SimHitVector> > allPtr;
iEvent.getByType( allPtr );

# by Selector
ParameterSelector<int> coneSel("coneSize",5);
Handle<JetVector> jetPtr;
iEvent.get( coneSel, jetPtr );
```

Looping over particles

Handle types are `edm::Handle<CollectionType>` and collection types are usually `std::vector<ObjectType>`.

if your product contains objects of type `ObjectType`, contained in collection of type `CollectionType` , and handled by the *variable* `theHandle`:

```
for (size_t j = 0; j < theHandle->size(); ++ j ) {
   const ObjectType & particle = (*theHandle)[ j ];
   cout << "->   electron" << j << " pT: " << particle.pt()
        << " eta: " << particle.eta()
        << " phi: " << particle.phi() << endl;
   someHisto ->Fill( particle.pt() );
};
```

Yet another loop using an iterator

```
size_t idx = 0;
for( CollectionType::const_iterator cand = theHandle->begin();
      cand != theHandle->end(); ++ cand, ++ idx ) {
      // do something, cand is a pointer to your object
      cout << "Electron #" << idx << ": pt=" << cand->pt() << endl;
}
```

Code snippets/recipes

Filtering events according to hlt path

This goes to the cfg file:

```
include "HLTrigger/HLTfilters/data/hltHighLevel.cfi"
replace hltHighLevel.HLTPaths = {"CandHLT1SumET"}
```

Sending a string to your analysis module from cfg file

Let's send a filename to our analysis class. Your cfg file should look like zis:

```
   module somename = YourAnalysisClass {
      ...
      untracked string fname = "somefile.root"
      ...
   }
```

and the header of your analysis class:

```
class YourAnalysisClass : public edm::EDAnalyzer{
   ...
   private:
      ...
      std::string fname_;
      ...
```

finally the analysis class constructor:

```
YourAnalysisClass::YourAnalysisClass(const edm::ParameterSet& cfg):
   fname_ ( cfg.getUntrackedParameter<string>( "fname" ) ) ,
   <further parameters received here> {
... constructor code here... }
```

Alternative way to receive parameters from within constructor:

```
YourAnalysisClass::YourAnalysisClass(const edm::ParameterSet& cfg){
   fname =  cfg.getUntrackedParameter<string>( "fname" );
   ... More constructor code/further parameter receiving code here.. }
```

More information about other types is here. Info about cfg files is here.

Is a parameter provided in the cfg file?

```
MyAnalyzer::MyAnalyzer(const edm::ParameterSet &conf) {
    if (conf.exists("some_parameter")) do_something
```

Does a certain product exist in your root file?

```
  Handle<L1GlobalTriggerReadoutRecord> L1GTRR;
  try {iEvent.getByLabel(l1GTReadoutRecTag_,L1GTRR);} catch (...) {;}
  if ( L1GTRR.isValid() ) { // exists
```

or

```
edm::Handle<edm::TriggerResults> trh;
try {iEvent.getByLabel(triggerInputTag_,trh);}
catch( cms::Exception& ex ) { LogWarning("HWWTreeDumper") << "Trigger results: " << triggerInputT
if (!trh.isValid())
   throw cms::Exception("ProductNotValid") << "TriggerResults product not valid";
```

Setting the event generator random number seed

Implementation in the `cfg` file is:

```
service =  RandomNumberGeneratorService {
```

```
   # This is to initialize the random engine of the source
    untracked uint32 sourceSeed = 123456789
}
```

Arsenal

There are numerous useful tools available for analyses. Using these tools will usually result in shorter code, and more reliable results; and the most important, If some of your results are just not OK, you can blame the developers on it 🙂

- Candidate selectors allow you to preselect physics objects on the fly, right within the `cfg` file.
- Gen decay tree analysis modules, ParticleListDrawer_Utility
- Candidate combiner reconstructs decayed particles (e.g. mu+mu- -> Z).
- Various matching modules
- Isolation stuff.
- Modules for filtering events.
- Common math functions

Locating/Accessing Datasets

1. To access root files from CMSSW, use the following format:

   ```
   source = PoolSource{
      untracked vstring fileNames = {'rfio:/castor/cern.ch/path/to/someData.root', 'another f
   }
   ```

   Here the `rfio:` prefix means that the file is on CASTOR. For a local file, use the `file:` prefix.
2. Now let's locate where the data is. Browse to the DBS Dataset Discovery Page and search for the dataset of your interest.
3. Once you find your dataset, check it's location. If located at `srm.cern.ch` , then you can access it directly from `lxplus` since it's on CASTOR. Click "plain". You'll see a long list of root files in the form:

   ```
   /store/mc/year/month/day/<~dataset name>/<some number>/longnamedfile.root
   ```

   This path translates to CASTOR location `/castor/cern.ch/cms/<location>`. You can pick e.g. individual files this way. (`"rfio:/castor/cern.ch/cms/store/RelVal/.../somedata.root"` or `"/store/RelVal/.../somedata.root"`).
4. If you want to access the whoel dataset, click "cff" instead of "plain" on dataset's page. This will show you a `cff` file containing paths and names of all root files belonging to that dataset. Directly include this `cff` file in your `cfg` file.
5. Hint: While working on a single root file which is on CASTOR, copying it to `/tmp/$USER` and then accessing from there may speed up things.

Framework messages to file

```
# Keep the logging output to a nice level #
include "FWCore/MessageService/data/MessageLogger.cfi"
replace MessageLogger.destinations = {"detailedInfo.txt"}
```

Other possibly useful stuff

- Event selection using trigger-HLT bits: here.
- Skipping missing input root files (goes into the PoolSource part):

```
untracked bool skipBadFiles = true #  skip missing or inaccessible files. Defaults to fals
```

Note that this won't prevent crashes with bad root files.

- Skipping the first *n* events in a root file:

```
source = PoolSource {
   untracked vstring fileNames = { <empty or list of files> }
   untracked uint32 skipEvents  = 21
}
```

Skips the *first 21* events. Note that the `FirstEvent` / `FirstRun` options may not work the same way.

- Throwing exceptions

```
throw cms::Exception("ErrorName") << "Message";
```

FIXME: Add header

- Skeleton code generators are very useful.
- Getting the cross-section of the generated event:

```
Handle<double> genCrossSect;
evt.getByLabel( "genEventRunInfo:PreCalculatedCrossSection", genCrossSect);
```

- Useful tool: addpkg , checks out packages from CVS.
- List of candidate clone producers.
- List of clone producers immediately available as modules.
- CMSSW location on AFS (`cern.ch`):
  `/afs/cern.ch/cms/sw/slc4_ia32_gcc345/cms/cmssw/CMSSW_X_Y_Z`,
- Other interesting locations on AFS (header files):
  - `$CMSSW_ROOT/src/DataFormats/Candidate/interface`
  - `$CMSSW_ROOT/src/DataFormats/Math/interface`
- Saving a Subsample of the AOD plus the User Data
- See which physics objects a root file contains: `EdmDumpEventContent <root file>`
- `scramv1 p` is the shorthand for `scramv1 project` . Other shorthands: `r` for `runtime` , `s` for `setup`
- I don't know what exactly `scramv1 b --fast` does but I think unless you do not change your buildfiles and add/remove new source files it's safe to use this option which speeds up the compile.

Dropping Products

Put into `PoolSource` module block

```
untracked vstring outputCommands =
  {
    "keep *",
    "drop *_*_*_HLT",
    "keep FEDRawDataCollection_*_*_*"
  }
```

More info available here.

Keeping Products

```
untracked vstring outputCommands = {
     "drop *",
    # HepMC event. Famos uses it
     "keep edmHepMCProduct_source_*_*",
     "keep edmGenInfoProduct_source_*_*",

    # or
```

```
 "keep *_source_*_*",

# Old gen event format
 "keep *_genParticleCandidates_*_*",

 # New gen event format
 "keep *_genParticles_*_*",          # to be removed when no external MC ref's left

# Keep trigger results
 "keep edmTriggerResults_TriggerResults_*_*",

 # if you chose to save all particles into a  collection in PAT layer0, to make it persistent
 "keep *_*_allJets_*",

# Gen jets
 "keep *_midPointCone5GenJetsNoNuBSM_*_*",

# Calo towers
"keep *_towerMaker_*_*",

 # CSA07 Stuff
 "keep *_genEventWeight_*_*",
 "keep *_genEventScale_*_*",
 "keep *_genEventProcID_*_*",
 "keep *_csa07EventWeightProducer_*_*"}
```

Notes

- Keep your buildfiles as clean as possible. Smaller buildfiles, faster compiles.

Useful Links Dump

- Which CMSSW version to go?
- Trigger Tables, HLT, L1, Menu
- Module cfg file parameters
- CMSSW reference manual
- CMSSW CVS (source) code browser
- CMSSW CVS search facility (LXR)
- Installing CMSSW with apt-get
- Cthulhu is always there to help you with your whatever problems!

r54 - 2016-07-05 - 09:56:10 - HalilGamsizkan

This topic: Main > CMSSWCheatSheet
Topic revision: r54 - 2016-07-05 - HalilGamsizkan