Introduction to the CMS Trigger

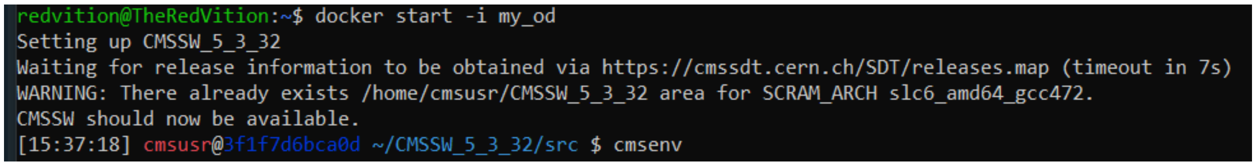
Exploring the triggers in your dataset

In order to check the trigger information we need to use the trigger tools provided in CMSSW. This record points you to some examples of the usage of such code. Let's use, for instance, some of the snippets presented in the source file of the GeneralInfoAnalyzer package to dump all the triggers in our dataset.

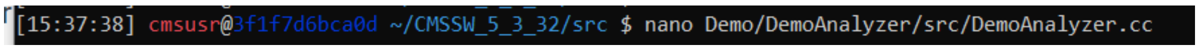
First, make sure you go to your `CMSSW_5_3_32/src` area and issue a `cmsenv` command, as you will always do.

To simplify the exercise, we will use the same source file we used for validation, the `Demo/DemoAnalyzer/src/DemoAnalyzer.cc`. Let's modify it so we can use it to dump all the triggers in our dataset. In the process, **let's comment out some of what we did for extracting the muon energy** so it does not clutter our output.

```
//classes to extract Muon information
#include "DataFormats/MuonReco/interface/Muon.h"
#include "DataFormats/MuonReco/interface/MuonFwd.h"

//for trigger configuration
#include "HLTrigger/HLTcore/interface/HLTConfigProvider.h"

//the standard c++ vector class
#include<vector>
```



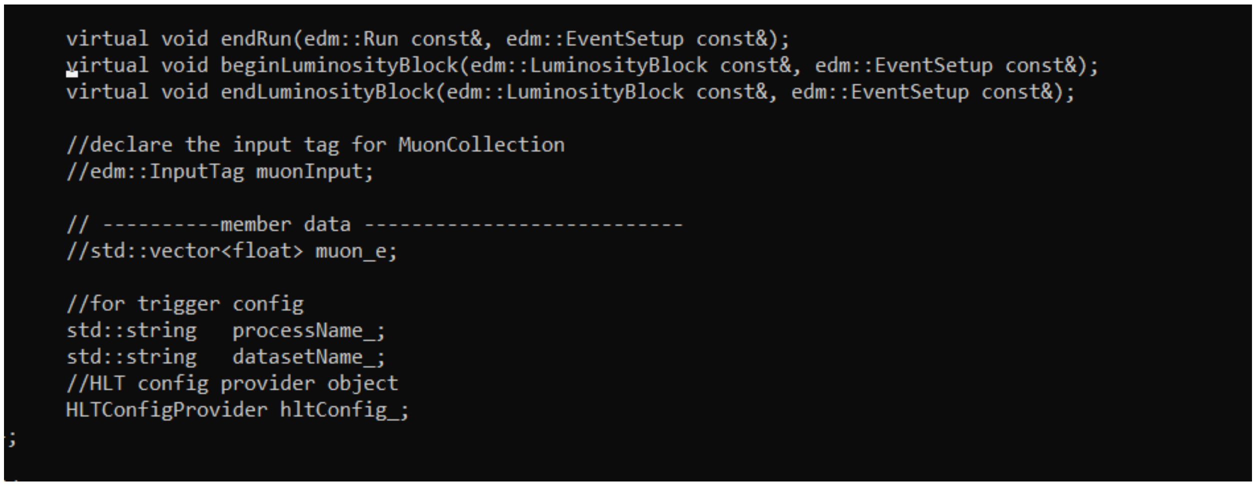
```
redvition@TheRedVition:~$ docker start -i my_od
Setting up CMSSW_5_3_32
Waiting for release information to be obtained via https://cmssdt.cern.ch/SDT/releases.map (timeout in 7s)
WARNING: There already exists /home/cmsusr/CMSSW_5_3_32 area for SCRAM_ARCH slc6_amd64_gcc472.
CMSSW should now be available.
[15:37:18] cmsusr@3f1f7d6bca0d ~/CMSSW_5_3_32/src $ cmsenv
```



```
[15:37:38] cmsusr@3f1f7d6bca0d ~/CMSSW_5_3_32/src $ nano Demo/DemoAnalyzer/src/DemoAnalyzer.cc
```

Next, insert the declaration of the variables we will need in our configuration file (process name, dataset name) and an object of class `HLTConfigProvider`, which we can use to extract the information about what the trigger configuration was for some run:

```
virtual void beginLuminosityBlock(edm::LuminosityBlock const&, edm::EventSetup const&);
  virtual void endLuminosityBlock(edm::LuminosityBlock const&, edm::EventSetup const&);

  //declare the input tag for MuonCollection
  //edm::InputTag muonInput;

  // ----------member data ---------------------------
  //std::vector<float> muon_e;

  //for trigger config
  std::string   processName_;
  std::string   datasetName_;
  //HLT config provider object
  HLTConfigProvider hltConfig_;
```


```
    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&);

    //declare the input tag for MuonCollection
    //edm::InputTag muonInput;

    // ----------member data ---------------------------
    //std::vector<float> muon_e;

    //for trigger config
    std::string   processName_;
    std::string   datasetName_;
    //HLT config provider object
    HLTConfigProvider hltConfig_;
;
```

Add the lines to read the configuration in the constructor and print it out (note the way it is done differs a bit from what we did earlier for the muons. They are, of course, equivalent):

```
// constructors and destructor
//
DemoAnalyzer::DemoAnalyzer(const edm::ParameterSet& iConfig):
processName_(iConfig.getParameter<std::string>("processName")),
datasetName_(iConfig.getParameter<std::string>("datasetName"))
{
   //now do what ever initialization is needed
   //muonInput = iConfig.getParameter<edm::InputTag>("InputCollection")
  using namespace std;
  using namespace edm;

  //Print the configuration just to check
  cout << "Here is the information passed to the constructor:" <<endl;
  cout << "Configuration: " << endl
       << "   ProcessName = " << processName_ << endl
       << "   DataSetName = " << datasetName_ << endl;


}
```

Don’t forget to comment out all the muon stuff in the analyze method so it does not bother us:

```
void
DemoAnalyzer::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
 using namespace edm;
 //clean the container
 //muon_e.clear();

 //define the handler and get by label
 //Handle<reco::MuonCollection> mymuons;
 //iEvent.getByLabel(muonInput, mymuons);

 //if collection is valid, loop over muons in event
 //if(mymuons.isValid()){
//    for (reco::MuonCollection::const_iterator itmuon=mymuons->begin(); itmuon!=mymuons->end(); ++itmuon){
//        muon_e.push_back(itmuon->energy());
//    }
 //}

 //print the vector
//  for(unsigned int i=0; i < muon_e.size(); i++){
//    std::cout <<"Muon # "<<i<<" with E = "<<muon_e.at(i)<<" GeV."<<std::endl;
 //}


#ifdef THIS_IS_AN_EVENT_EXAMPLE
   Handle<ExampleData> pIn;
   iEvent.getByLabel("example",pIn);
#endif

#ifdef THIS_IS_AN_EVENTSETUP_EXAMPLE
   ESHandle<SetupData> pSetup;
   iSetup.get<SetupRecord>().get(pSetup);
#endif
}
```

And, finally, modify the `beginRun`fuction giving a name to `iRun` and `iSetup` arguments and adding the trigger dump. Remember we have to check the triggers available at each change in runs:

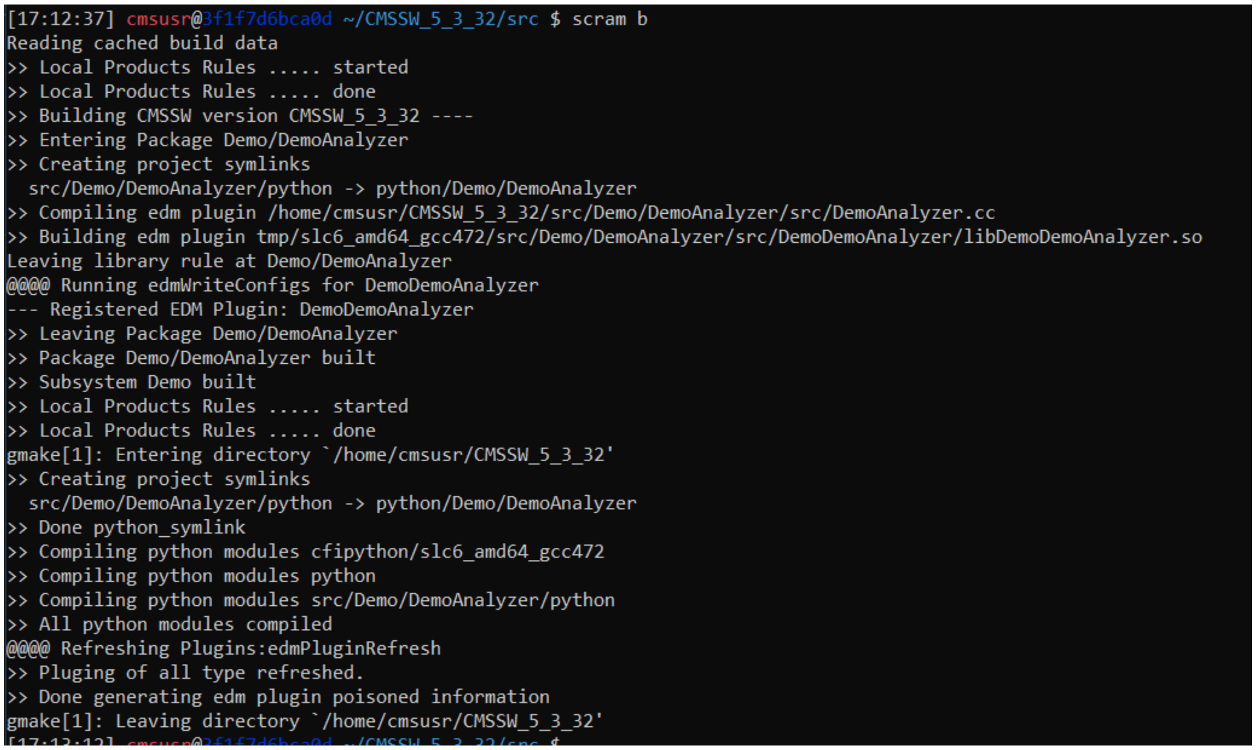
```
// ------------ method called when starting to processes a run  ------------
void
DemoAnalyzer::beginRun(edm::Run const& iRun, edm::EventSetup const& iSetup)
{

  using namespace std;
  using namespace edm;

  //If the hltConfig can be initialized, then the below is an example of
  //how to extract the config information for the trigger from the
  //so-called provenance.

  // The trigger configuration can change from
  // run to run (during the run is the same),
  // so it needs to be called here.

  ///   "init" return value indicates whether intitialisation has succeeded
  ///   "changed" parameter indicates whether the config has actually changed

  bool changed(true);
  if (hltConfig_.init(iRun,iSetup,processName_,changed)) {
      if (changed) {
        const vector<string> triggerNamesInDS = hltConfig_.datasetContent(datasetName_);
         for (unsigned i = 0; i < triggerNamesInDS.size(); i++) {
            cout<<triggerNamesInDS[i]<<endl;
          }
      }
  }

}
```

Before compiling, change your `Demo/DemoAnalyzer/BuildFile.xml` to include the `HLTrigger/HLTcore` package, where the `HLTConfigProvider` resides:

```
<use name="FWCore/Framework"/>
<use name="FWCore/PluginManager"/>
<use name="DataFormats/MuonReco"/>
<use name="FWCore/ParameterSet"/>
<use name="HLTrigger/HLTcore"/>
<flags EDM_PLUGIN="1"/>
<export>
   <lib name="1"/>
</export>
```

```
[16:00:35] cmsusr@3f1f7d6bca0d ~/CMSSW_5_3_32/src $ nano Demo/DemoAnalyzer/BuildFile.xml
```

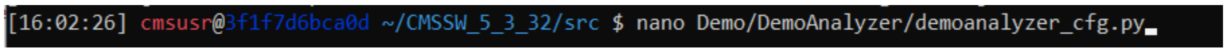
Compile the code the usual way: `scram b`



```
[17:12:37] cmsusr@3f1f7d6bca0d ~/CMSSW_5_3_32/src $ scram b
Reading cached build data
>> Local Products Rules ..... started
>> Local Products Rules ..... done
>> Building CMSSW version CMSSW_5_3_32 ----
>> Entering Package Demo/DemoAnalyzer
>> Creating project symlinks
  src/Demo/DemoAnalyzer/python -> python/Demo/DemoAnalyzer
>> Compiling edm plugin /home/cmsusr/CMSSW_5_3_32/src/Demo/DemoAnalyzer/src/DemoAnalyzer.cc
>> Building edm plugin tmp/slc6_amd64_gcc472/src/Demo/DemoAnalyzer/src/DemoDemoAnalyzer/libDemoDemoAnalyzer.so
Leaving library rule at Demo/DemoAnalyzer
@@@@ Running edmWriteConfigs for DemoDemoAnalyzer
--- Registered EDM Plugin: DemoDemoAnalyzer
>> Leaving Package Demo/DemoAnalyzer
>> Package Demo/DemoAnalyzer built
>> Subsystem Demo built
>> Local Products Rules ..... started
>> Local Products Rules ..... done
gmake[1]: Entering directory `/home/cmsusr/CMSSW_5_3_32'
>> Creating project symlinks
  src/Demo/DemoAnalyzer/python -> python/Demo/DemoAnalyzer
>> Done python_symlink
>> Compiling python modules cfipython/slc6_amd64_gcc472
>> Compiling python modules python
>> Compiling python modules src/Demo/DemoAnalyzer/python
>> All python modules compiled
@@@@ Refreshing Plugins:edmPluginRefresh
>> Pluging of all type refreshed.
>> Done generating edm plugin poisoned information
gmake[1]: Leaving directory `/home/cmsusr/CMSSW_5_3_32'
```

Now, let's modify the configuration file `Demo/DemoAnalyzer/demoanalyzer_cfg.py` to adapt it to our exercise. First, let's go back to logging for each event (and not for every 5) and change the number of events to `-1`, so we can run over all of them. Also, change the `PoolSource` file; replace it with a couple of files from our dataset selection. In addition, comment out what we had done for extracting the muon information and adding the HLTHighLevel filter, and replace it with parameters we need at configuration. Do not forget to notice that we are naming our process `mytrigger` now, and not `demo`.

Also, **make absolutely sure you have access to the conditions database information needed for 2012, which is different than that for 2011**. Here is where there is a key difference between using the **Virtual Machine** or the **Docker container**. When using the **Virtual Machine**, you have to replace the three lines that have `GlobalTag` in them with:



```
[16:02:26] cmsusr@3f1f7d6bca0d ~/CMSSW_5_3_32/src $ nano Demo/DemoAnalyzer/demoanalyzer_cfg.py
```

In this case, we are using docker, and we add this lines in the code:

```
#needed to access the conditions data from the Docker container
process.load('Configuration.StandardSequences.FrontierConditions_GlobalTag_cff')
process.GlobalTag.connect = cms.string('sqlite_file:/opt/cms-opendata-conddb/FT53_V21A_AN6_FULL_data_stripped.db')
process.GlobalTag.globaltag = 'FT53_V21A_AN6_FULL::All'
```

Something like that

These lines, with the `GlobaTag` string in them, have to do with being able to read CMS database information. We call this the **conditions data** as we may find values for calibration, alignment, trigger prescales, etc., in there . One can think of the `GlobalTag` as a label that contains a set of database snapshots that need to be adequate for a point in time in the history of the CMS detector. For the 2012 open data release, the global tag is `FT53_V21A_AN6` or `FT53_V21A_AN6_FULL` (the `::All` string is a flag that tells the frameworks to read *All* the information associated with the tag). You can find more information in this CODP guide.

The `connect` variable in one of those lines just modifies they way in which the framework is going to access these snapshots. For the VM we access them through the shared files system area at CERN (cvmfs). Read in this way, the conditions will be cached locally in your virtual machine the first time you run and so the CMSSW job will be slow. Fortunately, we already did this while setting up our VM, so our jobs will run much faster. In addition, those soft links he had to make are simply pointers to these areas.

On the other hand, in the Docker container, these database snapshots live locally in your `/opt/cms-opendata-conddb` directory. Running over them is much quicker.

Feel free to just replace the whole config file with the final version below (if using the VM, *uncomment and comment out the section in question appropriately*).

The final config file should look something like:

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("Demo")

process.load("FWCore.MessageService.MessageLogger_cfi")
process.MessageLogger.cerr.FwkReport.reportEvery = 1

process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(-1) )

process.source = cms.Source("PoolSource",
    # replace 'myfile.root' with the source file you want to use
    fileNames = cms.untracked.vstring(
    #    'file:myfile.root'
    #'root://eospublic.cern.ch//eos/opendata/cms/Run2012B/DoubleMuParked/AOD/22Jan2013-v1/10000/1EC938EF-ABEC-E211-94E0-90E6BA442F24.root'
    'root://eospublic.cern.ch//eos/opendata/cms/Run2012B/TauPlusX/AOD/22Jan2013-v1/20000/0040CF04-8E74-E211-AD0C-00266CFFA344.root',
    'root://eospublic.cern.ch//eos/opendata/cms/Run2012C/TauPlusX/AOD/22Jan2013-v1/310001/0EF85C5C-A787-E211-AFC9-003048C6942A.root'
    )
)

#uncomment to access the conditions data from the Virtual Machine (and comment out the Docker container set below)
#process.load('Configuration.StandardSequences.FrontierConditions_GlobalTag_cff')
#process.GlobalTag.connect = cms.string('sqlite_file:/cvmfs/cms-opendata-conddb.cern.ch/FT53_V21A_AN6_FULL.db')
#process.GlobalTag.globaltag = 'FT53_V21A_AN6::All'

#needed to access the conditions data from the Docker container
process.load('Configuration.StandardSequences.FrontierConditions_GlobalTag_cff')
process.GlobalTag.connect = cms.string('sqlite_file:/opt/cms-opendata-conddb/FT53_V21A_AN6_FULL_data_stripped.db')
process.GlobalTag.globaltag = 'FT53_V21A_AN6_FULL::All'

process.mytrigger = cms.EDAnalyzer('DemoAnalyzer',
  #InputCollection = cms.InputTag("muons")
    processName = cms.string("HLT"),
    datasetName = cms.string("TauPlusX"), #specific dataset example (for dumping info)
)

#process.load("HLTrigger.HLTfilters.hltHighLevel_cfi")
#process.hltHighLevel.HLTPaths = cms.vstring('HLT_Mu15*')


#process.p = cms.Path(process.hltHighLevel+process.demo)
process.p = cms.Path(process.mytrigger)
```

Note that the process name is always `HLT` for data that was processed with the online system.

Let’s run:

```
cmsRun Demo/DemoAnalyzer/demoanalyzer_cfg.py > full_triggerdump.log 2>&1 &
```