Table of Contents

ATLASWatchMan Tutorial with r15 : how to make my own steering file step by step ?

Introduction

The goal of this tutorial is to build from A to Z a non-trivial analysis with ATLASWatchMan. So we are going to select, reconstruct and dump candidates for semi-leptonic ttbar pairs events.

Installing ATLASWatchMan

- First you have to create a new directory for athena r15.2.0 (if not already done) and install ATLASWatchMan in it. For this, you should follow step by step instructions given on ATLASWatchManQuickStart wiki.

- Do not forget at the end of the installation process to try running the default steering file :

```
cd ../run
python -m ATLASWatchMan/ATLASWatchMan_Parser ATLASWatchMan_AnalysisSteeringFile_benchmarkC
athena -c "InputCollections=['myNiceFile1', 'myNiceFile2']; EvtMax=500" ../share/ATLASWatc
```

- ... and check that the output D3PD is not empty :

```
root.exe -l OutputD3PD.root
_file0->ls()
sumWeightsMC->Draw(); // Contains total number of weighted events read
benchmarkChannelsTree->Draw("channels.data()"); // channels passed
benchmarkChannelsTree->Draw("jet4mom.Pt():jet4mom.Eta()"); // jet Pt vs Eta
```

Two important points :

- ♦ there are two output trees : benchmarkChannelsTree contains event by event information, benchmarkChannelsInfoTree contains global information like maps between each object selection and its subsequent array position.
- ♦ the name of the D3PD trees are taken from your steering file name expect if you explicitly write in your steering file

```
treeName = "myTree"
infoTreeName = "myInfoTree"
```

.

Constructing basic fields

Dumping common variables

1. First you can try running on an empty steering file. In order to do this edit

   ```
   ATLASWatchMan/run/ATLASWatchMan_AnalysisSteeringFile_sltopAna.py
   ```

   and copy there :

   ```
   #!python

   # @file:    run/ATLASWatchMan_AnalysisSteeringFile_sltopAna.py
   # @author:  FirstName LastName <FirstName.LastName@cern.ch>
   # @purpose: a working example of the ATLASWatchMan Steering File used to analyze semilepto

   import GaudiKernel.SystemOfUnits as Units
   import ROOT
   ```

   and then run the usual commands :

   ```
   python -m ATLASWatchMan/ATLASWatchMan_Parser ATLASWatchMan_AnalysisSteeringFile_sltopAna
   athena -c "InputCollections=['myNiceFile1', 'myNiceFile2']; EvtMax=500" ../share/ATLASWatc
   ```

   When browsing the output D3PD, you will see that some containers branches named *electron4mom*, *jet4mom*,... *xxx4mom* is a std::vector < TLorentzVector > storing 4-momenta of particles named *xxx*. This is automatically dumped for every container in which EDM object posses a method named *.hlv()* returning HepLorentzVector.
2. The list of all predefined containers with their access key can be found in SUSYDefaultOptsLib.py under the dictionary named ***self.collections***.
3. If you want to run WatchMan on some of these containers, you have to specify a ***collections*** dictionary inside steering file like :

   ```
   #List of AOD/DPD Collections you want to use in your analysis either :
   # -> during event selection
   # -> to dump information in your output D3PD
   #Options are:
   # - Use the 'select':True flag to go through object selection
   # - The 'type' and 'name' flags are only needed if the collection has not been previously
   collections = {'electron':{'select':True},
                  'muon':{'select':True},
                  'jet':{'select':True},
                  'truth_jet':{'select': True},
                  'gen': {'select': False},
                  'met': {},
                  }
   ```

   The different keys (container ***name***, container ***type***, ***select*** to pass object selection) can be overwritten like

   ```
   'jet':{'select':True,'name':'Cone4H1TopoJets'}
   ```
4. In addition a ***dumpContainers*** dictionary should be specified to define the list of container you want to dump in the output D3PD :

   ```
   dumpContainers = {'electron':{'Author':{'type':'int','method':'.author()'}},
                     'muon':{},
                     'jet':{},
                     'truth_jet':{},
                     'met':{},
                     }
   ```

By default, only 4-momenta *4mom* and *Charge* are dumped into std::vector for each object into the collection. If, in addition, you want to dump to specific object information from which you know the method used to access it within athena, you can dump it like :

```
'electron':{'Author':{'type':'int','method':'.author()'}
```

. The resulting variable will be named ***electronAuthor*** and stored into an std::vector < int >. Via this way, you can easily make a list of additionnal variables you would like to dump in your output D3PD for each container.

Adding non-predefined containers (like clusters) will be explained below.

Event selection

Below is an example of event selection for 2 different channels : semi-leptonic top pairs and fully leptonic (with two electrons) top pairs.

```
# Event selection is based on the following two wikis :
# - https://twiki.cern.ch/twiki/bin/view/AtlasProtected/TopSingleleptonPubNote#event_selection
# - https://twiki.cern.ch/twiki/bin/view/AtlasProtected/TopDileptonPubNote#Event_Selection_and_re
channels = {'sltop4j1lep':{'channel': 'ljjjj',
                           'cuts': {1: {'label': 'electronCrackVeto'},
                                    2: {'label': 'electronPtCutsExclusive','value': [20*Units.GeV
                                    3: {'label': 'jetPtCuts',
                                        'value': [40*Units.GeV, 40*Units.GeV, 40*Units.GeV, 20*Un
                                    4: {'label': 'missingEtCut','value': 20*Units.GeV},
                                    },
                           },
            'fltop2jee':{'channel': 'lljj',
                         'cuts': {1: {'label': 'electronCrackVeto'},
                                  2: {'label': 'leptonPtCutsExclusive','value': [20*Units.GeV,20*
                                  3: {'label': 'SumChargeCut','value': [0]},
                                  4: {'label': 'InvariantMassCut',
                                      'custom': 'Y',
                                      'formula': 'InvariantMassCut'},
                                  5: {'label': 'jetPtCuts','value': [20*Units.GeV, 20*Units.GeV]}
                                  6: {'label': 'missingEtCut','value': 35*Units.GeV},
                                  },
                         },
            }

userFormula = {

# Invariant mass cut for dileptons channels
'InvariantMassCut':
r"""
pxtot = pytot = pztot = etot = 0.
for el in candidates['electron']:
    pxtot += el.px()
    pytot += el.py()
    pztot += el.pz()
    etot += el.e()
    pass
mzsquared=etot*etot - pxtot*pxtot - pytot*pytot - pztot*pztot
mz=0.
if mzsquared>0.:
         mz = math.sqrt(mzsquared)
else:
         print "Mzsquared is < 0. --->",mzsquared
if mz >= 86.*Units.GeV and mz <= 96.*Units.GeV: cutPassed = False
""",
}
```

The key points to notice are :

- When running with this new joboption and looking at the output D3PD, you can check quickly :
    - The number of events passed by every channel via :

      ```
      sltopAnaTree->Draw("channels.data()")
      sltopAnaTree->Scan("channels.data()")
      ```

    - The last cut passed by every channel. In order to do this last step, you need first to check which vector position is corresponding to which channel via the ***channelsMap*** like

      ```
      sltopAnaInfoTree->Scan("channelsMap")
      ***********************************
      *    Row   * Instance * channelsM *
      ***********************************
      *        0 *        0 * fltop2jee *
      *        0 *        1 * sltop4j1l *
      ***********************************
      sltopAnaTree->Draw("channelsLastCutPassed[0]"); // Drawing last cut passed by fltop2
      ```

- Many cuts are predefined in ATLASWatchMan_CutsLib.py. When designing your own event selection, you should first check whether the cut is existing in ATLASWatchMan_CutsLib.py.
- If the cut is not existing in ATLASWatchMan_CutsLib.py, you can design your custom cuts (like ***InvariantMassCut*** in previous example) by :
    - adding the fields '__custom__':'Y' and 'formula': 'InvariantMassCut'
    - adding in the new dictionnary ***userFormula*** a new entry explaining the code between triple quotes. By default a cut is assumed to be passed till you put the string

      ```
      cutPassed = False
      ```

      .

Playing with object selection and overlap removal

Till now, we have been using a DEFAULT object selection and overlap removal that is defined in SUSYDefinitions.cxx.

But, it is possible to add a custom object selection and overlap removal by adding a new dictionary named ***objectSelectionAndOverlap*** inside your steering file :

```
objectSelectionAndOverlap = {'slTopObjSel': {'muon':{'ptMin': 20.*Units.GeV,'etaMax': 2.5,
                                                     'etConeMax':6.*Units.GeV,'deltaRVeto_mj':0.3
                                             'electron':{'ptMin': 20.*Units.GeV,'etaMax': 2.5,
                                                         'etConeMax':6.*Units.GeV,'deltaRVeto_ej'
                                             'jet':{'ptMin': 20.*Units.GeV,'etaMax': 2.5},
                                             },
                             }
```

If you want that some of your channels are using one of the object selection you defined, you also need to specify it via e.g. :

```
'sltop4j1lep':{'channel': 'ljjjj',
              'objSelection': 'slTopObjSel',
              'cuts': {1: {'label': 'electronCrackVeto'},
                       2: {'label': 'leptonPtCutsExclusive','value': [20*Units.GeV]},
                       3: {'label': 'jetPtCuts',
```

```
                                      'value': [40*Units.GeV, 40*Units.GeV, 40*Units.GeV, 20*Units.GeV]},
                               4: {'label': 'missingEtCut','value': 20*Units.GeV},
                               },
            },
```

- Inside this dictionary every **electron**, **muon**, **photon**, **tau**, **jet** is following default definition from SUSYDefinitions.cxx. Every cut can be overwritten in a similar way as shown above.
- When defining new object selections, you need to know for every object stored in the D3PD and passing object selection (so with collection flag 'select':True) whether it was kept by your object selection. This information is stored in objects named ***xxxObjSel*** which are std::vector< std::vector< int >>. Like for ***channelsLastCutPassed***, a map named ***objSelectionMap*** is stored to tell which array position is corresponding to which object selection like :

```
sltopAnaInfoTree->Scan("objSelectionMap")
***********************************
*    Row   * Instance * objSelect *
***********************************
*        0 *        0 * objSelDEF *
*        0 *        1 * slTopObjS *
***********************************
sltopAnaTree->Draw("jet4mom[0].Pt()","@jet4mom.size() > 0 && jetObjSel[0][1] == 1"); // Dr
sltopAnaTree->Draw("electron4mom[0].Eta()","@electron4mom.size() > 0 && electronObjSel[0][
```

Decorating my D3PD with new variables

If you want, you can create new variables inside your steering file and dump them into the output D3PD. This is particularly usefull for event variables which are time consuming and that you do not want to recompute from D3PDs. For our current case, let's add two variables, the transverse mass of the W, the delta phi between the lepton candidate and the closest jet. This can be done via a new dictionnary named ***userD3PDBranchesToFill*** :

```
userD3PDBranchesToFill = {'mtw' : {'label': 'mtw', 'type': 'float', 'formula': 'transverseMass'},
                          'dphiLepJet' : {'label': 'dphiLepJet', 'type': 'float', 'formula': 'Dph
                          }
```

The first formula has already been implemented in ATLASWatchMan_CutsLib.py, whereas the second formula has to written by you and added to the ***userFormula*** dictionary :

```
# Compute delta phi between lepton and closest jet
'DphiLepJet':
r"""
## Objects can be accessed via candidates['xxx'] with xxx being the collection key name
lepCand = None
nlep = 0
for lepton in ['electron','muon']:
    for lep in candidates[lepton]:
        lepCand = lep
        nlep += 1
        pass
    pass
if lepCand == None or nlep != 1: return 999.
dphiMin = 999.
for jet in candidates['jet']:
    dphi = math.fabs(lepCand.hlv().vect().deltaPhi(jet.hlv().vect()))
    if dphi < dphiMin: dphiMin = dphi
    pass
return dphiMin
"""
```

- Inside the ***userFormula*** template, you can access all type of objects that you defined in ***collections*** dictionary via key

  ```
  candidates['collectionname']
  ```

- every new D3PD variable is defined as a std::vector of the 'type' you defined with a length equal to the number of object selections defined :

  ```
  sltopAnaInfoTree->Scan("objSelectionMap")
  ***********************************
  *    Row   * Instance * objSelect *
  ***********************************
  *        0 *        0 * objSelDEF *
  *        0 *        1 * slTopObjS *
  ***********************************
  sltopAnaTree->Draw("dphiLepJet[1]","channels.data() == \"sltop4j1lep\""); // draw dphi for
  sltopAnaTree->Draw("mtw[1]","channels.data() == \"sltop4j1lep\"")
  ```

Running an Athena Algorithm

It is possible to run a set of athena algorithms before (or after) ATLASWatchMan in order to use their results through storegate service via dictionary ***preAlgs*** and ***postAlgs***:

```
preAlgs = {"myTruthFilterAlg":{"name":"SUSYTruthFilter",
                               "pkg":"SUSYTools",
                               "doc":"Purpose: build a new thinned truth container can be dumped
                               },
          }
```

This command will run an algorithm named SUSYTruthFilter and dump a new container named FilteredSpclMC. Later you can use objects created by the algorithm within WatchMan by adding in ***collections*** :

```
'truth':{'name':'FilteredSpclMC'},
```

and in ***dumpContainers*** :

```
'truth':{},
```

If you build again the code and run athena on some sample, you will see that the list of selected truth particles is dumped in the output D3PD.

Dumping a non pre-defined container into D3PD

If you wish to dump information of a container non predefined in SUSYDefaultOptsLib.py, you need to defined it completly inside ***collections*** via :

```
'calocluster':{'name':'CaloCalTopoCluster','type':'CaloClusterContainer','select':False},
```

In addition, if you want to dump information in output D3PD, you need to update the ***dumpContainers*** :

```
'calocluster':{'Size':{'type':'int','method':'.getClusterSize()'}},
```

If you build again watchman and run on athena, you can then check e.g. calocluster eta distribution :

```
python -m ATLASWatchMan/ATLASWatchMan_Parser ATLASWatchMan_AnalysisSteeringFile_benchmarkChannels
athena -c "InputCollections=['myNiceFile1', 'myNiceFile2']; EvtMax=500" ../share/ATLASWatchMan_Ge
root.exe -l OutputD3PD.root
sltopAnaTree->Draw("calocluster4mom.Eta()")
```

The **4mom** attribute containing 4-momenta is automatically dumped for any collection which is of DataVector type.

Summary

This table summarize the main steering file dictionaries and their goal

Dictionary name	Description
channels	Event selection cuts for the different channels
collections	List of collections you are using : • to dump variables in output D3PD • compute new variables
dumpContainers	List of collections and variables you want to dump in output D3PD. Every dumped collection should have been predefined in __collections__
objectSelectionAndOverlap	Custom object selection and overlap removal cuts
preAlgs / postAlgs	List of athena algorithms run before / after ATLASWatchMan algorithms
userD3PDBranchesToFill	List of custom variables predefined or written in ***userFormula*** that you want to dump in output D3PD
userFormula	Dictionary containing definition (python code) of custom cuts or user variable

-- RenaudBruneliere - 18 Aug 2009

This topic: Main > ATLASWatchManTutorial1r15
Topic revision: r12 - 2009-10-23 - RiccardoMariaBianchi