Table of Contents

EventView

%PDFALL% %PDFTHIS% %COMPLETE4%

Introduction

The EventView was first developed to aid with removing overlap between different particle types within the AOD. However as the EventView matured, it became clear that the concept of "views of events" was also helpful for organizing and book-keeping analyses.

The EventView object is simply a container of:

- **Final State Particles**: pointers to physics objects (i.e. electron, muon, taus, jets) which are coherent, exhaustive, and mutually exclusive.
- **Inferred Objects**: pointers to particles constructed from the final state particles (for example composite particles such as Z or Higgs candidates).
- **User Data**: any variables or observables calculated in the course of analysis, generally calculated from the final state or inferred particles.

The EventView object contains minimal logic beyond what is necessary to easily fill and access this information. Building a meaningful EventView is the up to the user. He/she may interact with the EventView directly in their analysis code. In addition, there is a substantial library of tools, called the EventView Toolkit, which aid with filling and accessing the information inside EventView. There are more specialisied toolkits for particular analyses such as SusyView and TopView. Because there may be several ways that AOD objects can be assigned or combined, each event may have more than one EventView.

Schematic representation of how EventView fits into the ATLAS data chain.

Terms used in EventView

- **EventView** (EV): EventView object is the main component in the EventView analysis. There are many ways to interpret one event; one EventView object represents one of such "view". The construction of the EventView is thus defined by the users using "inserters" and multiple EventViews may be defined in an analysis.
- **UserData** (UD): UserData is like an NTuple and it's a container of variables which can be used from anywhere in analysis through EventViews. In case of multiple EventViews, each view has its own UD. Since UD is not copied to the daughter EVs when they are created, quantities that do not exist in the UD of daughter EV will be fetched from its parent EV. UserData can be dumped out to Ntuplea of different forms including AANtuple.
- **Inserters**: Inserters take the AOD containers and construct EventView objects defined by the user. The process includes preselection and overlap removal and FinalStateObjects are determined for the EventView.
- **Labels**: EV allow users to attach text designations (i.e. labels) to particles. Each particle can be (multiply) labelled and later retrieved via its label.
- **FinalStateObjects**: A set of particles that are inserted from AOD's. FinalStateObjects may not be modified to retain consistency of the view.
- **InferredObjects**: When composite pariticles are reconstructed from their decay daughters, they should be treated as InferredObjects to avoid double counting.

See the EventView wiki pages for more information including tutorials and documentation. The following is mostly taken from EventViewBuilderTutorialConf.

Getting Started with EventView

Create a new file called (say) `myEVOptions.py` containing the following:

```
# files necessary to setup EventView environment
include("EventViewConfiguration/EventViewFullInclude_jobOptions.py")

# Import the default modules
from EventViewConfiguration import *

# instantiate the top alg sequence for athena, only one is required
theJob = AlgSequence()

# instantiate one tool looper for reconstructed
defaultEVLooper = EVMultipleOutputToolLooper("defaultEVLooper")

# the name of the EventView (in StoreGate) created by this tool looper
defaultEVLooper.EventViewCollectionOutputName="defaultEventView"

# schedule EVMultipleOutputToolLoopers to athena top sequencer
theJob += defaultEVLooper

# define insertion order, you may want to play with this order
toInsert=["Electron", "Photon", "Muon", "TauJet", "JetTag",
          "ParticleJet", "MissingEt", "EventInfo"]

# schedule the default inserter module with the insertion order as defined above
# Leave out ",toDo=toInsert" for default configuration
defaultEVLooper += DefaultEventView("Inserters", toDo=toInsert)

# schedule screen dumper, printUD=True to see UserData contents
defaultEVLooper+=EVScreenDump("defaultScreenDumper", printUD=False)

# print the whole job schedule
print theJob

EventSelector.InputCollections = [
                                 "aod.pool.root"
                           ]

# specify the number of events you want to run
theApp.EvtMax = 50
```

For further documentation see EventViewConfiguration, RemoveOverlap and EventViewDumpers.

Change the `EventSelector.InputCollections` appropriately and run athena:

```
> athena.py myEVOptions.py > athena_ev.out
```

In the athena initialization you should see a printout of what you have requested and all the options:

```
AlgSequence AlgSequence/topSeq
 |  |-<no properties>
 |- Algorithm EVMultipleOutputToolLooper/defaultEVLooper
    |  |-ErrorMax        : <no value>
    |  |-AuditAlgorithms : <no value>
    |  |-Enable          : <no value>
    |  |-EventViewTools  : []
...
    |- AlgTool EVModule/Inserters
       |  |-EVMuonSelection : {'ContainerKey': 'MuidMuonCollection',  ...
       |  |-EVElectronSelection: {'useNN': False, 'absoluteIsolationCut': 15000.0,  ...
...
```

For each event you should see something like this (these were Z->e+e- events):

```
AthenaEventLoopMgr   INFO   ===>>>  start of event 6  <<<===
defaultEVLooper...   INFO
*****************************************************
------------------ EV Screen Dump -------------------
---------------- Final State Objects ----------------
Object 0:  p_T = 42291.9 phi = 0.779337 eta = -0.502384 type = Electron
Object 1:  p_T = 37871.7 phi = -2.44968 eta = 0.656399 type = Electron
---------------- Inferred Objects -------------------
---------------------- UserData ---------------------
eventNumber runNumber MET_Final_et MET_Final_ex MET_Final_ey MET_Final_sumet ...
EV does not have Parent
defaultEVLooper...   INFO
----------------- End EV Screen Dump ----------------
*****************************************************
AthenaEventLoopMgr   INFO   ===>>>  end of event 6    <<<===
```

Adding Labels

EventView allows you to attach string labels to the particles. The `EVInserterTools` tools allow setting of 3 different types of labels, **Selected**, **Inserted** and **Overlap**.

The labeling logic is as follows: for each particle

- Check if the particle is selected
- Check if the particle overlaps with other objects already in EventView
- If overlap and selected: label overlapping particle with Overlap and Selected labels
- If no overlap and selected: insert object and label with Selected and Inserted labels

Add the following to `myEVOptions.py` between `DefaultEventView` and `EVScreenDump`:

```
# Schedule the labeler module
defaultEVLooper += DefaultEventViewLabels("Labeler", defaultEVLooper.Inserters)
```

You should now get extra lines in the output:

```
Object 0:  p_T = 42291.9 phi = 0.779337 eta = -0.502384 type = Electron
   Labels: Electron  ElectronS  JetTagOL  ParticleJetOL  ParticleJetS  TauJetOL
Object 1:  p_T = 37871.7 phi = -2.44968 eta = 0.656399 type = Electron
   Labels: Electron  ElectronS  JetTagOL  ParticleJetOL  ParticleJetS  TauJetOL
```

which says that both objects would be selected as an Electron or a ParticleJet (but electron took precedence because of the inserter order) and both overlap with JetTag, ParticleJet and TauJet.

Accessing the Truth

You can also build an EventView of the 'Truth'. After `DefaultEventViewLabels` but before `EVScreenDump` add:

```
#Truth
truthEVLooper = EVMultipleOutputToolLooper("truthEVLooper")
truthEVLooper.EventViewCollectionOutputName="truthEventView"
theJob += truthEVLooper

truthEVLooper += TruthEventView("TruthInserters")
truthEVLooper += TruthEventViewLabels("Labeler", truthEVLooper.TruthInserters)
```

and after `EVScreenDump` add:

```
truthEVLooper+=EVScreenDump("truthScreenDumper", printUD=False)
```

You should now have additional printout:

```
truthEVLooper.t...   INFO
*****************************************************
------------------ EV Screen Dump -------------------
---------------- Final State Objects ----------------
Object 0:  p_T = 38414.9 phi = -2.44927 eta = 0.655139 type = TruthParticle
   Labels: TrueElectron  TrueJet
Object 1:  p_T = 41975.6 phi = 0.779032 eta = -0.502809 type = TruthParticle
   Labels: TrueElectron  TrueJet
```

Creating an Ntuple

Rather than look at the printout we now put all the information into an ntuple. Add the following before `EVScreenDump`:

```
# Copy particle information into UserData
defaultEVLooper += UserDataDump("UserDataDump", Labels=defaultEVLooper['Labeler']._Labels
                       + truthEVLooper['Labeler']._Labels)

# Write UserData to AANtuple
defaultEVLooper+=AANtupleFromUserData("defaultAADumper", filename="EVExample.AAN.root", sequencer
```

and you should get an ntuple `EVExample.AAN.root`. If you browse this with root you will see that there are many trees CollectionTree and EV0, EV1 etc. If some events have multiple EventViews then EV0 contains information on the first view of each event, EV1 the second etc. You will see that EV0 has entries for all events (see the eventNumber variable) but EV5 (if it exists) has only has entries for those events that have 5 or more different views. The tree EVCands contains data on all the views of an event (multiple entries per event). See EventViewNtuple and ROOTNtupleHowTo for further information.

Type

```
root -l EVExample.AAN.root
```

and then (for instance)

```
EVCands->Draw("El_p_T")
```

to get the Pt of the electrons.

Creating User Data

In order to make all kinematic variables available in the ntuple add the following before `AANtupleFromUserData`:

```
defaultEVLooper += [ anEVTool("EVUDFinalStateLooper/EVUDFSLAll"),
                     anEVTool("EVUDFinalStateLooper/EVUDFSLElectron") ]

defaultEVLooper.EVUDFSLAll+=  anEVTool("EVUDKinCalc")
defaultEVLooper.EVUDFSLElectron+=  anEVTool("EVUDKinCalc")
defaultEVLooper.EVUDFSLElectron+=  anEVTool("EVUDElectronAll")

defaultEVLooper.EVUDFSLAll.setProperties(Prefix= "All_")

defaultEVLooper.EVUDFSLElectron.setProperties( Labels= [ "Electron" ],
                                               Prefix= "El_",
                                               SortParticles= True) # Decending p_T sort
```

This adds 2 instances of `EVUDFinalStateLooper` tool then adds instances of `EVUDKinCalc` (saves kinematic info such as p_T and energy into UserData) and `EVUDElectronAll` (saves electron specific info such as isEM or hasTrack into UserData) tools to these `EVUDFinalStateLooper` tools. Then it configures `EVUDFSElectron` to only loop over electrons by adding a label requirement. It also provide different prefixes to the loopers so they don't overwrite each other's UserData. See EventViewUserData for further information.

You should see things such as `All_E` in the Root EVCands tree.

Combining Objects

You can combine your objects to make composite objects (see EventViewCombiners). For instance to combine two "Electrons" to make "ElectronPairs" add the following before `AANtupleFromUserData`.:

```
defaultEVLooper+=[ anEVTool("EVSimpleCombo/ZeeCombo") ]

defaultEVLooper["ZeeCombo"].setProperties(Labels=["Electron"],  # input particle label
                            OutputLabel= "ElectronPair",  # output particle label
                             LowMass = 0*GeV,             # mass window
                             HighMass = 250*GeV,
                             NDaughters = 2,               # Number of daughters
                             PDGID = 23,     # PDG id of Z not really necessary
                             PassOnNoCombo = True        )
```

you should get something like this in the printout:

```
---------------- Inferred Objects -------------------
Object 0:  m = 93791.3 p_T = 5636.69 phi = 1.40623 eta = 0.73567 type = CompositeParticle
   Labels: ElectronPair
```

In order to get kinematic information about the composite particle change:

```
defaultEVLooper += [ anEVTool("EVUDFinalStateLooper/EVUDFSLAll"),
                     anEVTool("EVUDFinalStateLooper/EVUDFSLElectron") ]
```

to:

```
defaultEVLooper += [ anEVTool("EVUDFinalStateLooper/EVUDFSLAll"),
                     anEVTool("EVUDFinalStateLooper/EVUDFSLElectron"),
                     anEVTool("EVUDInferredObjectLooper/ElecPairLooper") ]
```

and add:

```
defaultEVLooper.ElecPairLooper+= anEVTool("EVUDKinCalc")

defaultEVLooper.ElecPairLooper.setProperties( Labels= [ "ElectronPair" ],
                                              Prefix= "Z_",
                                              SortParticles= True) # Decending p_T sort
```

You should now see things line `Z_E` in your Root file. If you want to look at the e+e- invariant mass you should do something like:

```
> root EVExample.AAN.root
root [1] TH1F histo("histo", "", 50, 0, 250000)
root [2] EVCands->Draw("Z_m>>histo", "Z_N>0")
```

and you should see a Z peak.

If you have calculated the invariant mass as described in

WorkBookArchiveStartingAODAnalysis#InvariantMass on the same events you may notice that you don't seem to get identical results. This is because there are some default cuts in EventView. To overcome this you need to relax these cuts. After:

```
defaultEVLooper += DefaultEventView("Inserters", toDo=toInsert)
```

add:

```
defaultEVLooper.Inserters.ElectronInserter.etCut=0*GeV
defaultEVLooper.Inserters.ElectronInserter.makeEtaCuts=False
defaultEVLooper.Inserters.ElectronInserter.DoPreselection=False
defaultEVLooper.Inserters.ElectronInserter.RemoveOverlapWithSameType=False
```

and in the block

```
defaultEVLooper["ZeeCombo"].setProperties(Labels=["Electron"],  # input particle label
```

add

```
                                     CheckCharge = False,
```

which switches off the (default) requirement that the two electrons have opposite electric charge.

Originally taken from EventViewBuilderTutorialConf and various talks by Kyle Cranmer, Amir Farbin and Akira Shibata.
-- AmirFarbin - 06 Oct 2006 -- KyleCranmer - 06 Oct 2006 -- AkiraShibata - 06 Oct 2006 -- SteveLloyd - 14 Nov 2006

%RESPONSIBLE% SteveLloyd
%REVIEW%

This topic: AtlasArchive > WorkBookArchiveEventView
Topic revision: r13 - 2007-05-25 - SteveLloyd