Table of Contents

7.10a Tau Tagging

Complete: 🟨🟨⬜
Detailed Review status

N.B. All references to B tagging in this page should be ignored. They are historic and have been superceded by the ***b tag chapter of the Workbook****.*

Goals of this page

This page is intended to familiarize you with running the Tau tagging algorithms, accessing information about the tagged jets, and looking at their CMS.MonteCarlo flavour. In particular:

- Which algorithms are available, and how to run them
- What is the `JetTag object`
- What is an "Extended" collection, or `xxxTagInfoCollection`
- How to get the information you want from those
- How to associate to each jet a CMS.MonteCarlo flavour

Contents

NOTE:

Since CMSSW_1_7_0 and further versions the tau tagging based on the DataFormats/BTau package (explained in this section of the workbook) is **deprecated** for offline analysis. A new CMS.DataFormat/TauReco has been created with two different classes: CaloTau and PFTau, the relative Workbook can be found here.

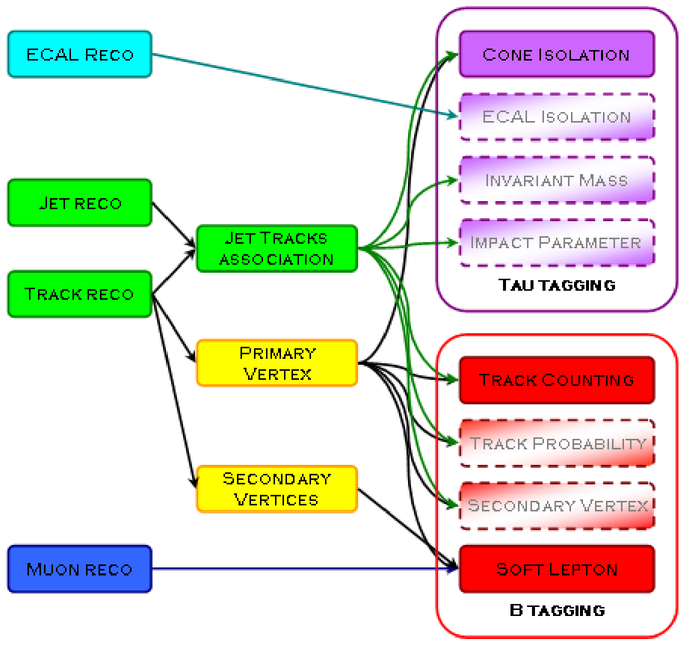
Introduction

The Tau objects are essentially made by a jet, a collection of tracks associated (in some way) to the jet, and a discriminator which is the actual result of the tagging. Each algorithm may also produce some "extended" information relative to the algorithm used to produce the object.

The reconstruction chain is very simple:

- Jets and tracks are reconstructed by standard producers
- A `JetTrackAssociator` associates tracks to jets (at the moment there is only one cone-based implementation, and a slightly smarter one taking into account effective tracks propagation is underway)

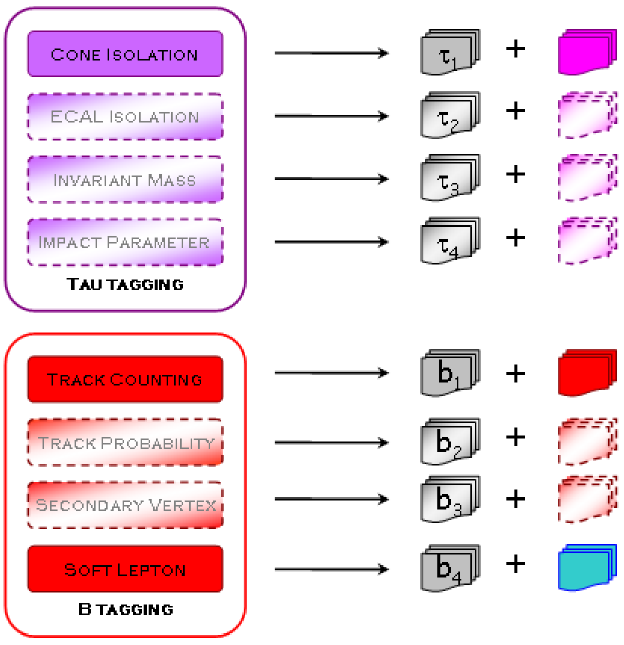
- The output of the associator is used as input for the producers
- If needed, the primary vertex is used
- Some algorithms require additional inputs which are in turn read from the Event.



Several different producers exist or are being written for Tau tagging:

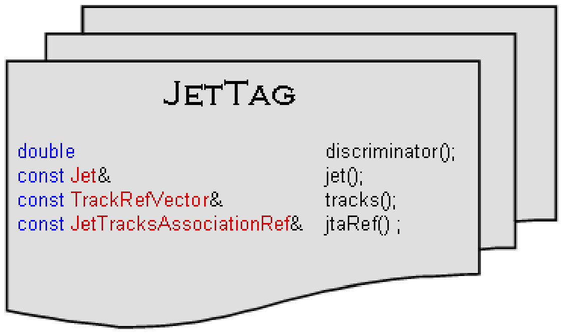
- Isolation Cone
- Impact Parameter
- Mass Tag (work in progress)

Each producer creates two collections: a `JetTagCollection` and an extended `xxxTagInfoCollection`.



JetTag

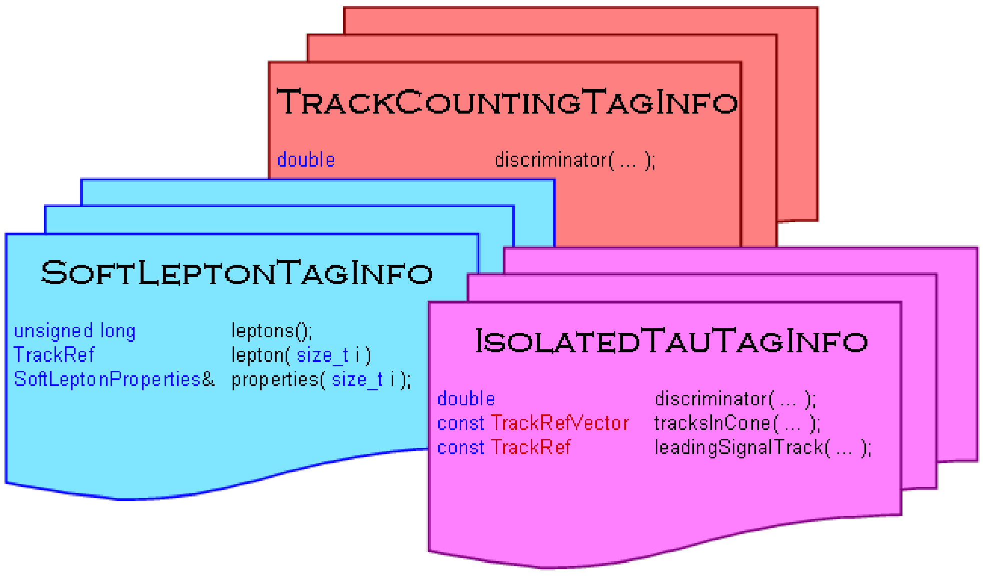
JetTag contains the basic result (discriminator) of the tagging algorithm and a link (edm::Ref) to the input object. All tagging algorithms produce a collection of `JetTag` objects.


JetTag
double discriminator();
const Jet& jet();
const TrackRefVector& tracks();
const JetTracksAssociationRef& jtaRef() ;

ExtendedCollections (xxxTagInfo)

This is a collection of classes which have:

- a link to the `JetTag` (to access jets and tracks used as input)
- algorithm specific data members and quantities used in computation of the discriminator
- methods to recompute the discriminator with different parameters (NB: not all parameters can be changed out of the producer!)


TrackCountingTagInfo
double discriminator(...);
SoftLeptonTagInfo
unsigned long leptons();
TrackRef lepton(size_t i)
SoftLeptonProperties& properties(size_t i);
IsolatedTauTagInfo
double discriminator(...);
const TrackRefVector tracksInCone(...);
const TrackRef leadingSignalTrack(...);

Existing extended collections:

- `IsolatedTauTagInfo` (produced by `RecoTauTag/ConeIsolation`)

Set up your Environment

This assumes you'll be using CMSSW 1.2.0 (and bash - for [t]csh use the -csh option on the scram command line):

```
scramv1 project CMSSW CMSSW_1_2_0
cd CMSSW_1_2_0/src
eval `scramv1 runtime -sh`

project CMSSW
```

How to run the code and create a ROOT file with the RECO object

As of CMSSW release 1.2.0, some B and Tau tagging algorithms are already available in the default distribution. Thus, there is no need to compile them yourself: you can just write the appropriate .cfg file to run them. Moreover, our input files will already have all the required Reco objetcs (jets, tracks, vertices, ...).

⚠ **NOTE:** In fact, the tau tagging algorithms described in this page are run with their default settings on all the CMSSW 1.2.0 data samples, so you don't even need to run them unless you wish to tune the algorithm parameters to better suite your needs. Since we'll run on these files anyway, most of the tau tag objects will come out duplicated.

For this tutorial, we will use a configuration file for Tau tagging In your working area you can do:

```
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookBTauTagging/test_tautag.cfg
cmsRun test_tautag.cfg
```

The relevant fragments of the configuration file is:

```
    # Tracks to jets association
    include "RecoBTau/JetTracksAssociator/data/jetTracksAssociator.cfi"

    # Tau tagging
    include "RecoTauTag/ConeIsolation/data/coneIsolationTauJetTags.cfi"

    # module execution
    path tautag = { jetTracksAssociator, coneIsolationTauJetTags }
```

All such configuration files have at least two modules

- a jet-tracks associator
- a tau tagging algorithm, for example `coneIsolationTauJetTags`

The default configuration for these producers is defined in their respective .cfi files, such as

```
    module jetTracksAssociator = JetTracksAssociator {
      InputTag tracks = ctfWithMaterialTracks
      InputTag jets   = iterativeCone5CMS.CaloJets
      double coneSize = 0.5
    }
```

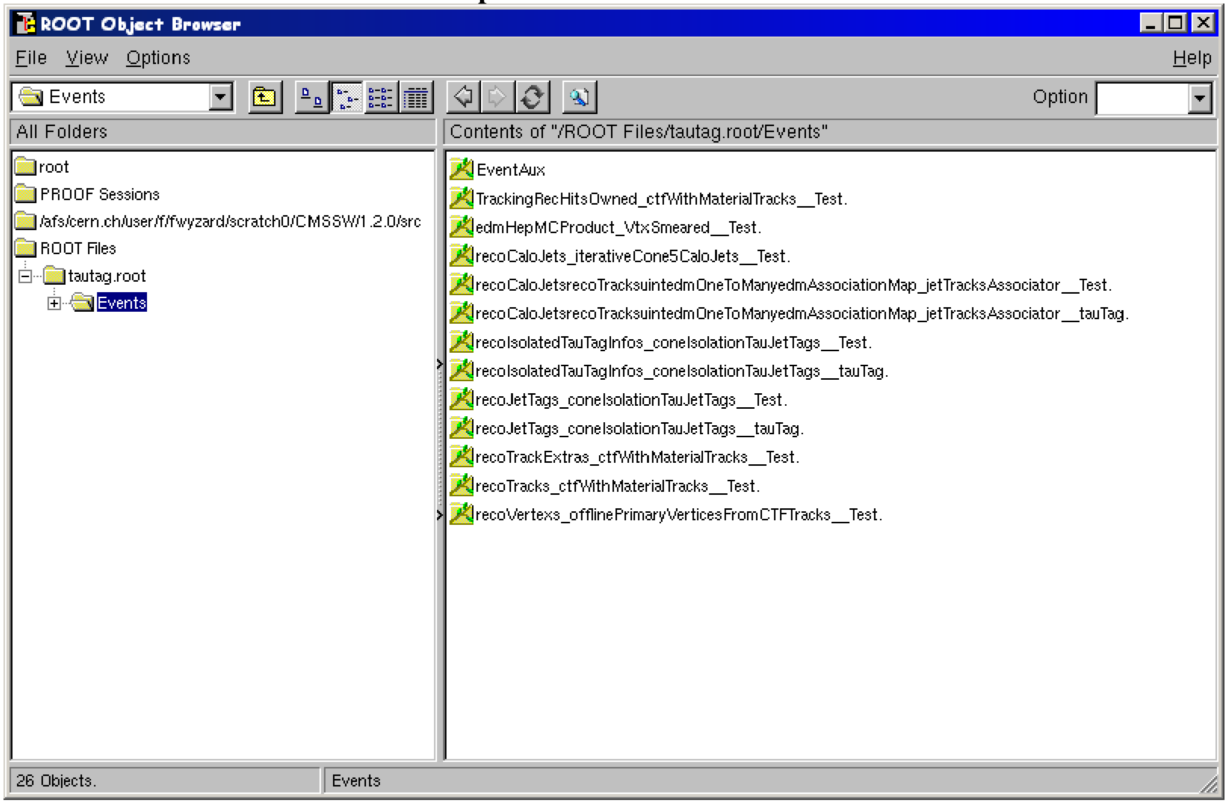
Accessing JetTag with bare ROOT

Bare ROOT functionality is described in Analysis with Bare ROOT. To access a `JetTag` collection in this manner, you can simply open the root files produced above with ROOT:

```
root tautag.root
new TBrowser;
```

and then navigate in the Browser ...

Screenshot of the TBrowser of the output file of the `ConeIsolation` module:



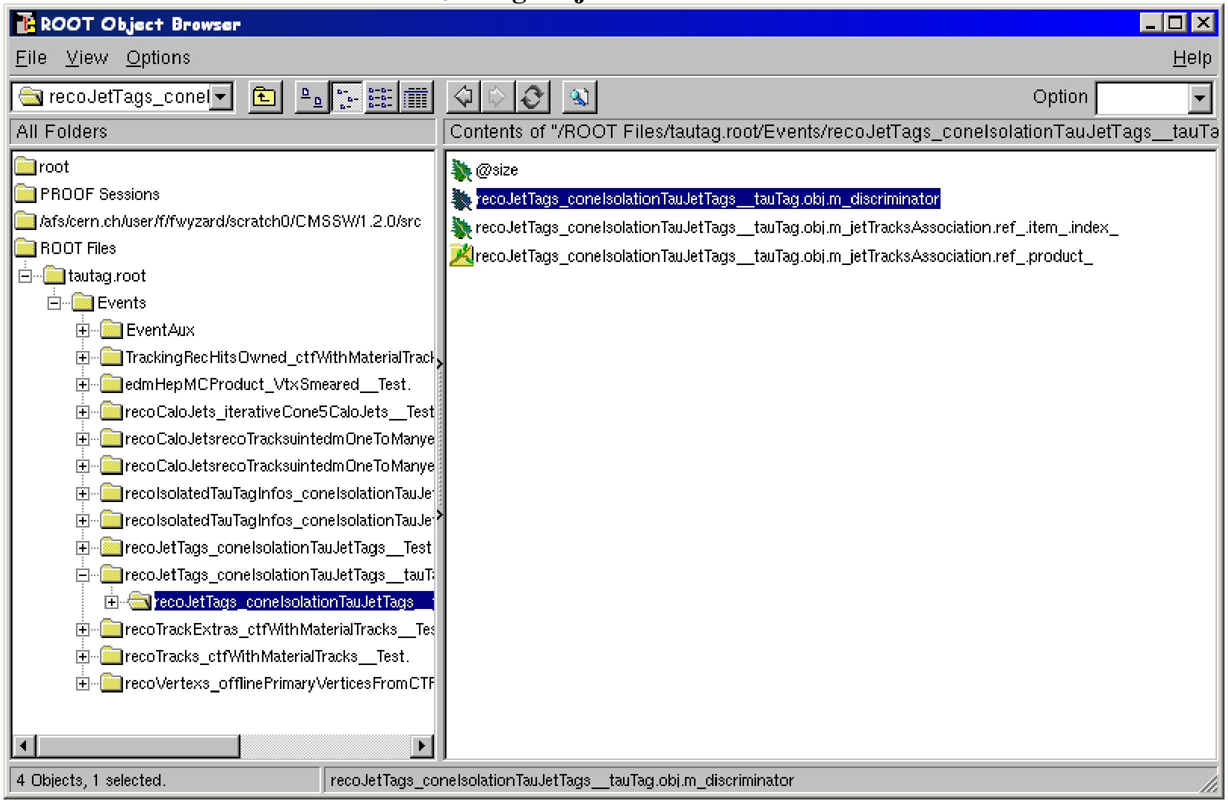
Contents of "/ROOT Files/tautag.root/Events"

```
root
PROOF Sessions
/afs/cern.ch/user/f/fwyzard/scratch0/CMSSW/1.2.0/src
ROOT Files
tautag.root
Events
```

```
EventAux
TrackingRecHitsOwned_ctfWithMaterialTracks__Test.
edmHepMCProduct_VtxSmeared__Test.
recoCaloJets_iterativeCone5CaloJets__Test.
recoCaloJetsrecoTracksuintedmOneToManyedmAssociationMap_jetTracksAssociator__Test.
recoCaloJetsrecoTracksuintedmOneToManyedmAssociationMap_jetTracksAssociator__tauTag.
recoIsolatedTauTagInfos_coneIsolationTauJetTags__Test.
recoIsolatedTauTagInfos_coneIsolationTauJetTags__tauTag.
recoJetTags_coneIsolationTauJetTags__Test.
recoJetTags_coneIsolationTauJetTags__tauTag.
recoTrackExtras_ctfWithMaterialTracks__Test.
recoTracks_ctfWithMaterialTracks__Test.
recoVertexs_offlinePrimaryVerticesFromCTFTracks__Test.
```

26 Objects. Events

Screenshot of the leaves of the recoJetTags objects:



Contents of "/ROOT Files/tautag.root/Events/recoJetTags_coneIsolationTauJetTags__tauTa

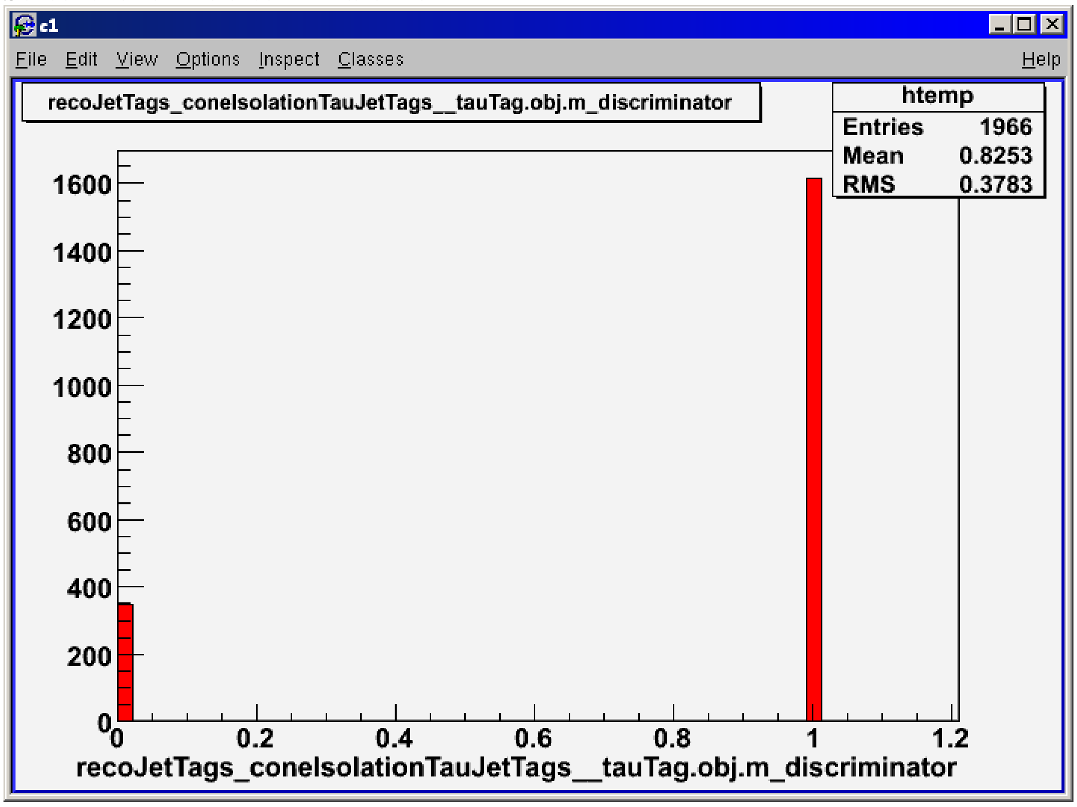
```
root
PROOF Sessions
/afs/cern.ch/user/f/fwyzard/scratch0/CMSSW/1.2.0/src
ROOT Files
tautag.root
  Events
    EventAux
    TrackingRecHitsOwned_ctfWithMaterialTrack
    edmHepMCProduct_VtxSmeared__Test.
    recoCaloJets_iterativeCone5CaloJets__Test
    recoCaloJetsrecoTracksuintedmOneToManye
    recoCaloJetsrecoTracksuintedmOneToManye
    recoIsolatedTauTagInfos_coneIsolationTauJe
    recoIsolatedTauTagInfos_coneIsolationTauJe
    recoJetTags_coneIsolationTauJetTags__Test
    recoJetTags_coneIsolationTauJetTags__tauT
      recoJetTags_coneIsolationTauJetTags__
    recoTrackExtras_ctfWithMaterialTracks__Te
    recoTracks_ctfWithMaterialTracks__Test.
    recoVertexs_offlinePrimaryVerticesFromCTF
```

```
@size
recoJetTags_coneIsolationTauJetTags__tauTag.obj.m_discriminator
recoJetTags_coneIsolationTauJetTags__tauTag.obj.m_jetTracksAssociation.ref_.item_.index_
recoJetTags_coneIsolationTauJetTags__tauTag.obj.m_jetTracksAssociation.ref_.product_
```

4 Objects, 1 selected. recoJetTags_coneIsolationTauJetTags__tauTag.obj.m_discriminator

Double clicking on the discriminator leaf, the plot with the values of the discriminator will appear ("0" means not tagged, "1" means tagged).

Screenshot of the discriminator for `ConeIsolation`:



fwLite analysis

First you should launch ROOT and set it up:

```
root

gSystem->Load("libFWCoreFWLite.so")
AutoLibraryLoader::enable()
```

Then open one of the files produced above and have a look at it:

```
TFile* file = new TFile("tautag.root");
new TBrowser;

Events->SetAlias("coneIsolBase", "recoJetTags_coneIsolationTauJetTags__tauTag.obj")
Events->SetAlias("coneIsolExt",  "recoIsolatedTauTagInfos_coneIsolationTauJetTags__tauTag.obj")
Events->Draw("coneIsolExt.whatever()");
```

Analysis with an EDAnalyzer

This approach is described in Using the full CMSSW framework with an EDAnalyzer.

In this section we show how to access collections and methods from an EDAnalyzer, using as an example TauTagTest.cc. This analyzer shows how to access the main method of the IsolateTauTagInfo object, defined in IsolatedTauTagInfo.h

First, we have to build the analyzer. From your working area, do:

```
cvs co -r CMSSW_1_2_0 RecoTauTag/ConeIsolation/test
cd RecoTauTag/ConeIsolation/test
scramv1 build
```

Then we need an appropriate .cfg file. We can modify the one used above to run the Cone Isolation algorithm, adding

```
    module tauTagTest = TauTagTest {
        string JetTagProd = "coneIsolationTauJetTags"
    }

    path runEDAna = { tauTagTest }
```

And finally run it:

```
cmsRun test_tautag.cfg
```

The extended collection (IsolatedTauTagInfoCollection) has all the methods needed to do the analysis. This is very useful when creating efficiency plots, for example:

```
IsolatedTauTagInfoCollection::const_iterator i = tauTagInfo.begin();
int it=0;
for (; i != tauTagInfo.end(); ++i)
{
  //To compute the efficiency as a function of the isolation cone (Riso)
  if (it == 0)
  {
    for(int ii=0;ii<6;ii++)
    {
      nEventsUsed[ii]++;
      float Riso = ii*0.05 + 0.2;
      float Rmatch = 0.1;
      float Rsig = 0.07;
      float pT_LT = 6.;
      float pT_min =1.;
      if (i->discriminator(Rmatch, Rsig, Riso, pT_LT, pT_min) > 0) nEventsRiso[ii]++;
    }
  }
  // [...]
}
```

And it is useful in debugging the algorithm:

```
    cout << "Discriminator from jetTag = " << i->discriminator() << endl;
    const TrackRef leadTk = (i->leadingSignalTrack(0.45, 1.));
    if (! leadTk) {
        cout <<"No Leading Track "<<endl;
    } else {
        cout <<"Leading Track pt "<<(*leadTk).pt()<<endl;
        math::XYZVector momentum = (*leadTk).momentum();

        //0.07 = SignalCone, 1. = minimum pT of tracks
        cout << "Number of SignalTracks = " <<
    (i->tracksInCone(momentum, 0.07,  1.)).size() << endl;
       // [...]
    }
```

You can book the histogram and fill it inside the code:

```
void TauTagTest::endJob(){
  TH1F effVsRiso("eff","Eff",6,0.2,0.5);
```

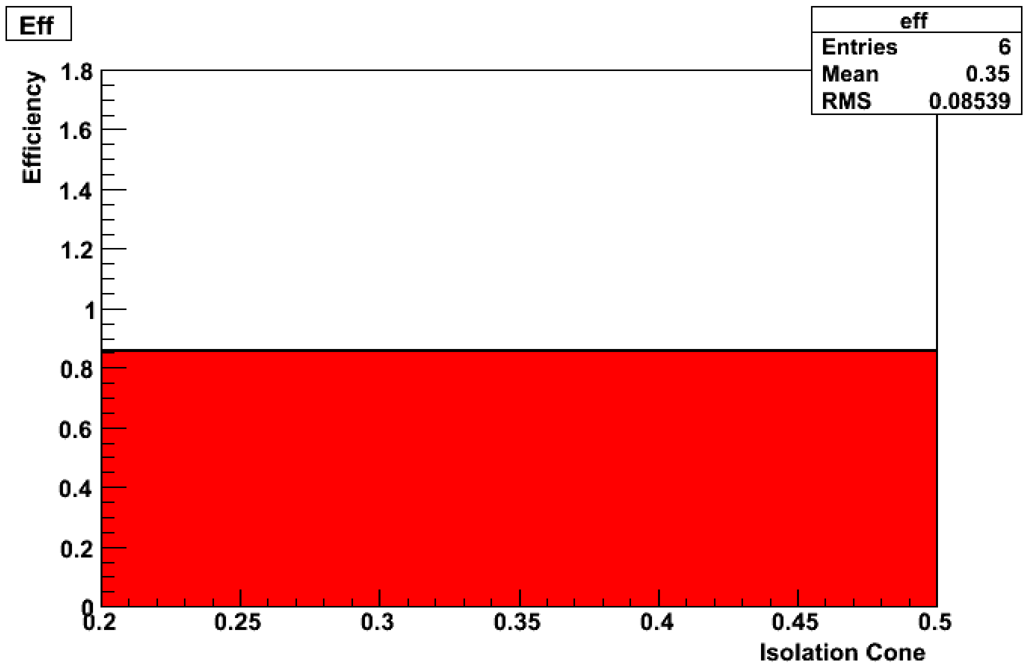
```
    int ibin;

  for (int ii = 0; ii < 6; ii++){
    if (nEventsUsed[ii] > 0)
      ibin= ii+1;
    float eff = nEventsRiso[ii]/nEventsUsed[ii];
    effVsRiso.SetBinContent(ibin,eff);
  }

  //Write the histos in a rootfile
  TFile hFile( "histo1.root", "RECREATE" );
  effVsRiso.Write();

  // Make a plot and save it as png
  gROOT->SetBatch();
  gROOT->SetStyle("Plain");
  TCanvas c;
  effVsRiso.GetXaxis()->SetTitle( "Isolation Cone" );
  effVsRiso.GetYaxis()->SetTitle( "Efficiency" );
  effVsRiso.SetFillColor( kRed );
  effVsRiso.SetLineWidth( 2 );
  effVsRiso.Draw();
  c.SaveAs( "eff.png" );
};
```

And here is the result:



Further Information

Related hypernews forums, group pages and software guide pages:

- Energy/Particle Flow and TauId: Energy/Particle Flow and TauId hypernews, group homepage

Review status

Reviewer/Editor and Date (copy from screen)	Comments

IanTomalin - 13 Nov 2007	Deleted historic b tag documentation from this page
AnneHeavey - 19 Jul 2006	Reviewed after created by Simone and Andrea , prior to tutorial session.

Responsible: SimoneGennai
Last reviewed by: RickCavanaugh - 27 Feb 2008

This topic: CMSPublic > WorkBookTauTagging
Topic revision: r73 - 2008-04-08 - SimoneGennai