Table of Contents

3.3 How to Make an Analysis

Complete:
Detailed Review status

THIS PAGE IS OBSOLETE AND PLEASE IGNORE ADVICE ON THIS PAGE
This page is superseded and not part of the workbook any more. The content of this page has been moved to SWGuideAnalysisPatterns

Goals of this page:

Introduce the different analysis schemes available for CMSSW.

Contents

Introduction

All CMS RECO data or AOD (analysis objects data) are stored in ROOT files. This tutorial first gives a short introduction to the file format and then explains the available approaches to doing analysis with the data files. The different analysis approaches include:

- Bare ROOT
- FWLite mode
- Full Framework (with EDAnalyzer)
- candidate tools (four-vector analysis)

This tutorial assumes that you know the basic commands like scram and cvs.

The RECO data format - short reminder

All data is stored in so-called RECO objects. The RECO classes are designed to provide a simple and uniform interface. For instance, the same method-naming convention is used to access the same quantities throughout

the various classes:

- everywhere: `pt()`, `eta()`, `phi()`, `energy()`
- not inconsistent conventions: `E()`, `Energy()`, `getE()`, `getEnergy()`, etc.

RECO reference documentation is collected in the CMS Offline Guide .

Set up your Environment

Set your runtime environment (shown for release %WBRELEASENEW%):

```
cd CMSSW_%WBRELEASENEW%/src
cmsenv
```

- Full instructions
- Summary for every login session
- Access modules from the CVS repository.

The sample data file

NOTE: The method describe to look at the data files below in DBS may not be confined to CMSSW_2_1_12 release. Sometimes the release is deprecated and you may not find the files. So use this for guidance and look for similar sample in other releases. The data file used for instruction here is in CMSSW_2_1_9.

For this tutorial we will work with the TTbar release validation sample available for grid analysis and stored in CERN Advanced STORage manager (CASTOR). You can find this sample corresponding to this release following these instructions:

1. Go the the Data discovery page. (Here you select both the event type, and the data tier (RAW, RECO, AOD, etc) to run your analysis on.)
2. Choose a release close to `CMSSW_2_1_12` as the Software Release from the scroll-down menu
3. Choose `RelValTTbar` as the Primary Dataset/ MC generators from the scroll-down menu (not all releases have them, if the release of your choice does not have them, choose another release in the same release series).
4. Choose the data tier RECO from the scroll-down menu
5. Click on the Find button.

If you click on **show** for one of the samples on the list you get, you will get a table with the location of files and you will see that at CERN (srm.cern.ch) there are 33 files with 10050 events (32.4 GB in total).

If you want to copy the entire file, you can use `rfcp` to copy one file of the example files to our current directory. You find the Logical File Name (LFN) by clicking on the link in the LFNs column following the srm.cern.ch. To copy the file, we use the Physical File Name (PFN), so you will need to add `/castor/cern.ch/cms` to the beginning of the name. Change to the target directory of your choice, and run the following command (cut and paste so that the command is all on one line). Do this ONLY if you really want to copy the entire file, if not see the instructions in the following paragraph.

```
rfcp /store/relval/CMSSW_2_1_9/RelValTTbar/GEN-SIM-DIGI-RAW-HLTDEBUG-RECO/IDEAL_V9_v2
/0000/1A0FD639-1B86-DD11-A3C0-000423D99614.root reco.root

# edit the above two lines to be a single line
```

This may take some time, and it may well be that it does not fit to your disk quota. (If you're really determined, and have disk quota problems, see Workbook: Setting Up Your Computing Environment: Working with Large Files to store and access the file on CASTOR.)

If, instead, you only want to copy few events, you can use a simple configuration file to copy some events of the file to your local area. Save the following in `copy_cfg.py`

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("COPY")

process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(10) )

process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring("/store/relval/CMSSW_2_1_9/RelValTTbar/GEN-SIM-DIGI-RAW-HLT
)

process.copyAll = cms.OutputModule("PoolOutputModule", fileName = cms.untracked.string("reco.root
process.printEventNumber = cms.EDAnalyzer("AsciiOutputModule")
process.out = cms.EndPath(process.copyAll + process.printEventNumber)
```

and copy the events by typing `cmsRun copy_cfg.py`. Note that this may take some time, if the file is not "staged" (i.e. already stored in a disk area which allows fast access). At the end of the process, you will find a file `reco.root` in your local area.

Analysis with Bare ROOT

In the Bare ROOT access, only the output file is used in ROOT. No shared libraries are loaded. Bare ROOT access is useful for doing very simple validations of the file and its contents. In order for this analysis mode to be possible, the data within the Event must be kept very simple, for example, C style structs.

Bare ROOT access doesn't allow you to access the object methods, but you will be able to access objects' private attributes. Let's start with an example:

To do this exercise, make sure you do NOT have ROOT set up for loading libraries! Start ROOT:

```
root.exe
```

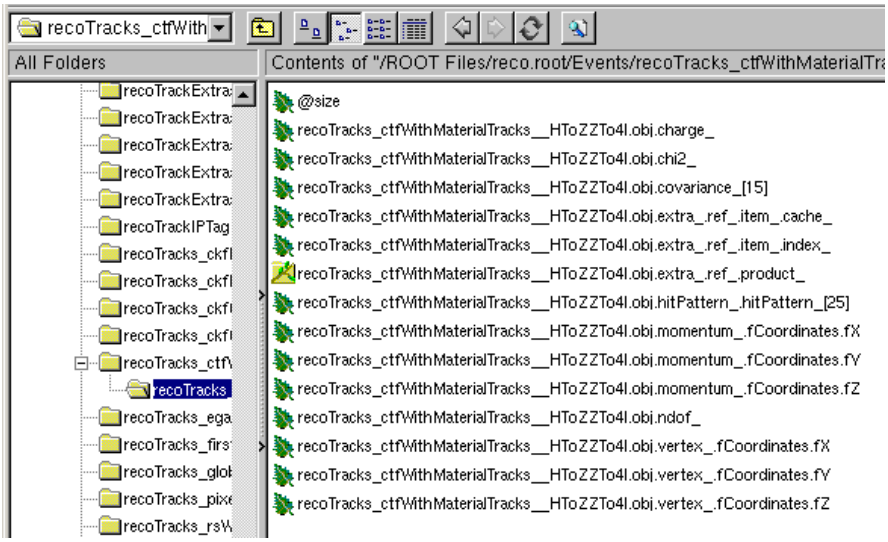
and open the `reco.root` file. You will find a number of warning messages due to missing dictionaries in the file. This is expected.

```
root[0] TFile f("reco.root");
```

By opening a browser you can explore the file content:

```
root[1] new TBrowser
```

Drill down through ROOT files, reco.root, Events, `recoTracks_ctfWithMaterialTracks_Rec1`, `recoTracks_ctfWithMaterialTracks_Rec1.obj` to see the contents:



What you see here are the private attributes of the recoTracks. Note, that in you will see `Rec1` instead of `HToZZTo4l` which is shown in the figure.

The object names you see are composed as follows:

```
DataType_ModuleName_InstanceName_ProcessName
```

where

- DataType is a more or less compact name for of the C++ data stored (recoTracks in this example)
- ModuleName is the name of the configuration module that produced the data (ctfWithMaterialTracks in this example)
- InstanceName is optional (hence the double '_' in many names), and is set by the producer, for example in case it makes more than one collection of the same type in a single module (empty in this example)
- ProcessName is the name of the full process (Rec1 in this example)

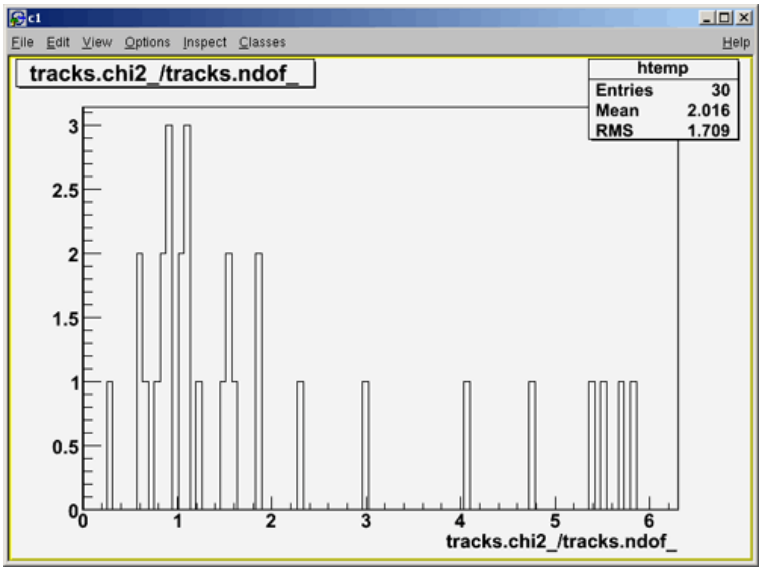
Instead of going through the TBrowser each time, you can set an alias for a branch you want to access. Use the branch name visible in the TBrowser, adding `obj` at the end. E.g., here we set the alias for recoTracks_ctfWithMaterialTracks__Rec1.obj to "tracks":

```
root[2] Events->SetAlias("tracks","recoTracks_ctfWithMaterialTracks__Rec1.obj");
```

Thus the "leaf" object tracks.chi2_ corresponds to the actual leaf object recoTracks_ctfWithMaterialTracks__Rec1.obj.chi2_ as shown in the ROOT browser, above.

Using this alias, you can plot a track's normalized chi-squared (ndof = number of degrees of freedom):

```
root[3] Events.Draw("tracks.chi2_/tracks.ndof_");
```



(This plot was made using the reco.root file with only a small number of tracks.)

Tools: Event ID in ROOT mode

It is useful to identify the event number of particularly interesting events - for example to examine them in the CMS event display. To access the event number when working in ROOT mode (possible also in FwLite/ROOT), you can issue the command:

```
Events.Scan("EventAuxiliary.id_.event_");
```

Analysis in Framework-Lite (FWLite) mode

FWLite is the automatic library loading system. FWLite allows you to call any object method interactively without loading the needed libraries by hand. You don't have to know which libraries contain each C++ class. FWLite automatically loads the correct shared library the first time ROOT needs to use the dictionary for the class. With this, you have access to all methods of the RECO objects (as in the full framework) in the ROOT file. In comparison in bare ROOT mode you could only access the object's variables.

For this analysis mode, you need to make sure the the FWLite autoloader is enabled in ROOT. To do so, see Workbook: Setting up your Computing Environment: Set up ROOT. The commands listed there may be executed either manually at the ROOT prompt, or via your `$HOME/rootlogon.C` file.

Next, start ROOT (and execute the "set up ROOT" commands manually if necessary). It's quickest to use `root -l` (lower case `L`) which just skips the colourful welcome screen:

```
root -l
```

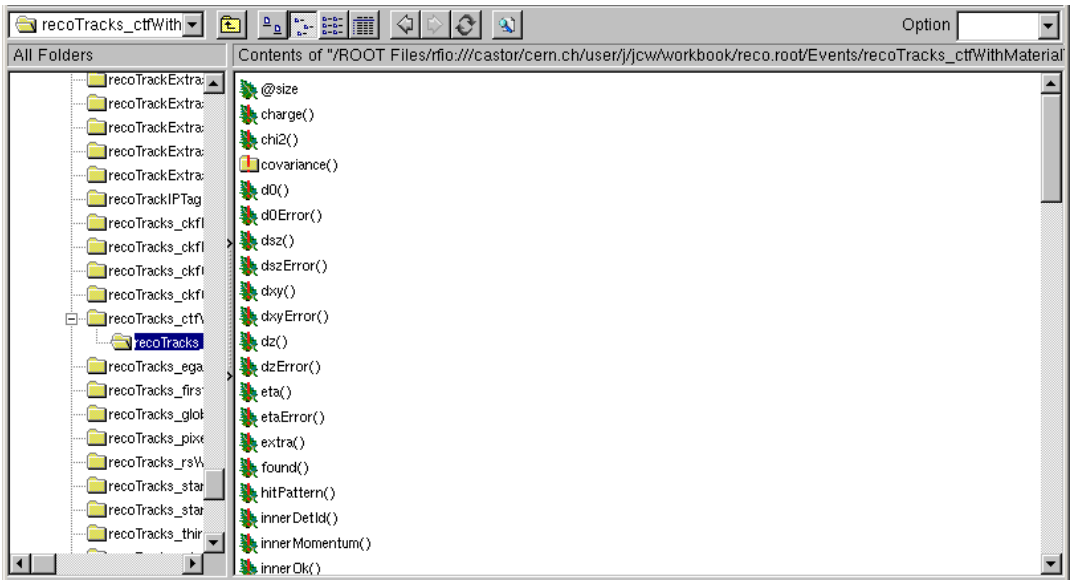
From the ROOT prompt, open the `reco.root` file that you copied earlier. It contains a collection of RECO objects. Opening the file may take some time.

```
root[0] TFile f("reco.root");
```

Look at the ROOT TBrowser to see which collections are in the file.

```
root[1] new TBrowser;
```

Double-click on *ROOT Files*, then *reco.root*, then *Events* (the event tree). Drill down through `recoTracks_ctfWithMaterialTracks__Rec1` to `recoTracks_ctfWithMaterialTracks__Rec1.obj`. We show only an abbreviated list of its contents; enough to see that it's different from what we found for Bare ROOT:



What you see here are the methods of the stored recoTracks. Since **`covariance()`** returns a class, the icon shows a folder.

While you still have the ROOT browser displayed, go back to your ROOT session and set an alias for each of the branches you want to access (we'll choose `recoTracks` and `recoTrackExtras`). Use the branch name that's visible in the TBrowser, and add `obj` at the end, as follows:

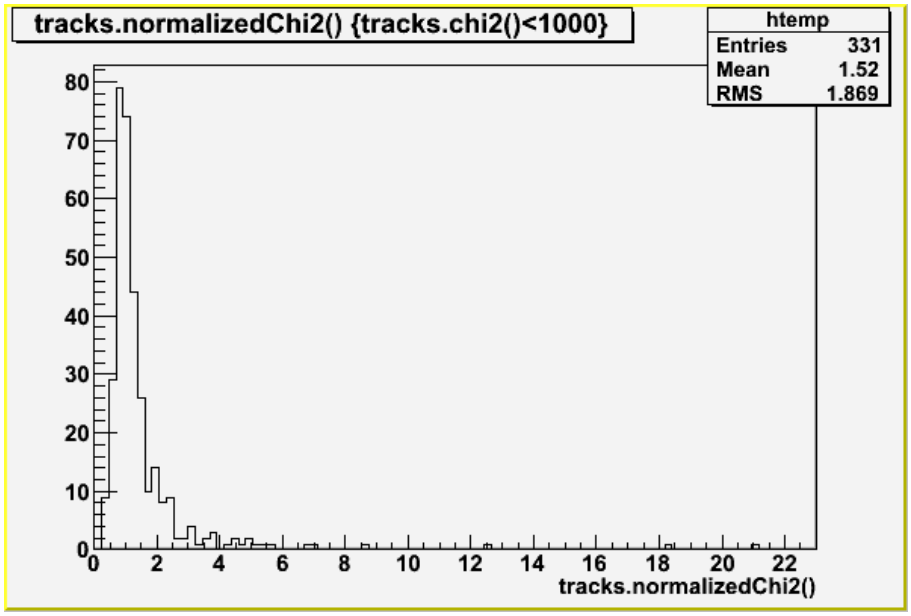
```
root[2] Events->SetAlias("tracks","recoTracks_ctfWithMaterialTracks__Rec1.obj");
root[3] Events->SetAlias("trackExtras","recoTrackExtras_ctfWithMaterialTracks__Rec1.obj");
```

Note: SetAlias commands work only if the ROOT browser is still up.

Now, look at the examples contained in the file `CMS.PhysicsTools/RecoExamples/test/trackPlots.C`, and copy some of the examples into the ROOT prompt to see how they work. For instance:

// plot chi-squared divided by n. degrees of freedom

```
root[4] Events.Draw("tracks.normalizedChi2()", "tracks.chi2()<1000")
```



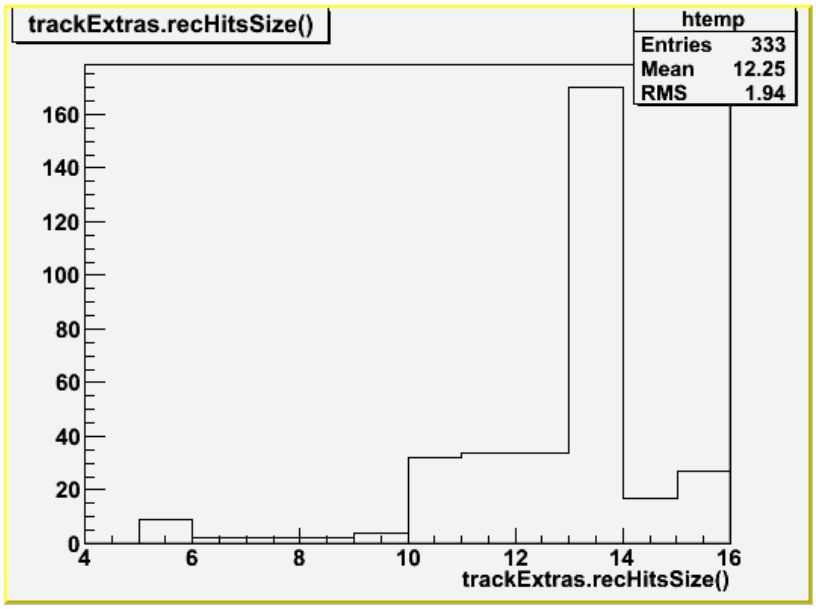
(This image is shown for 100 events, each containing several tracks, from the full reco file.)

Now if you run this same "Events.Draw" command in Bare ROOT (after setting the alias), you'll get the message

```
Part of the Variable "tracks.chi2()" exists but some of it is not accessible or useable.
```

Now run:

```
root[5] Events.Draw("trackExtras.recHitsSize()");
```



Of course you can prepare a ROOT script and run it in batch mode, if you like.

FWLite in Python

If you prefer Python to CINT (the C interpreter in ROOT) you can use PyROOT instead. The commands for startup are quite close to the CINT version.

Invoke python with:

```
python
```

Then load the libraries:

```
>>> from ROOT import *
>>> gSystem.Load("libFWCoreFWLite.so")
>>> AutoLibraryLoader.enable()
```

To load a file and to open a browser, type:

```
>>> myFile = TFile("reco.root")
>>> myBrowser = TBrowser()
```

This brings up the standard ROOT Object browser.

Now you can use some of the advantages of the Python interpreter. If you want to know something more about a class, use the `help` command. For example (command followed by output is shown):

```
>>> help(HepMC.GenParticle)

Help on class HepMC::GenParticle in module __main__:

class HepMC::GenParticle(ROOT.ObjectProxy)
 |  Method resolution order:
 |      HepMC::GenParticle
 |      ROOT.ObjectProxy
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  BeginDaughters(...)
 |      int GenParticle::BeginDaughters()
 |
 |  CollisionNumber(...)
 |      int GenParticle::CollisionNumber()
 |
 |  CreationVertex(...)
 |      CLHEP::HepLorentzVector GenParticle::CreationVertex()
 |
 |  DecayVertex(...)
 |      CLHEP::HepLorentzVector GenParticle::DecayVertex()

...
```

This way you can explore all objects inside the ROOT file and learn step-by-step what information you can get. Just take an object, use `help` and get used to the interface of the class.

To quit the interactive python session, press **Ctrl-D** or **.q** as in ROOT.

#CINTEvent

CINT event loop

If you want to use FWLite for looping through all events and with more complicated analysis of the data stored, you can write your own event loop in FWLite mode. First you 'connect' your defined variables to the according branches via `SetAddress`. Then you simply loop over all events and for every branch you want to access you do `GetEntry` inside the loop. Here is an example showing how this would work for generator

information:

```
TFile theFile("reco.root");

TTree *events = (TTree*)theFile.Get("Events");
events->GetEntry();

// connection of products and branches
edm::HepMCProduct prod;
TBranch *sourceBranch = events->GetBranch(events->GetAlias("VtxSmeared"));
sourceBranch->SetAddress(&prod);

HepMC::GenEvent *genEvent;
for( unsigned int index = 0; index < events->GetEntries(); index++)
{
    sourceBranch->GetEntry(index);
    events->GetEntry(index,0);
    genEvent = prod.GetEvent();
    ...
```

The . . . shows where your analysis code should go.

Event loop in Python

You could use a similar syntax as above for a Python event loop. But there is a much smarter way. First you create an `EventTree` and then access the products inside the event by `EventTree.branch`. Inside the loop there is no `GetEntry` necessary. The point here is that you don't need to define the variable beforehand and thus you don't have to know the type of the object you want to access. This makes the coding much faster. Here an example showing how it works:

```
from ROOT import *
from CMS.PhysicsTools.PythonAnalysis.cmstools import *

### prepare the FWLite autoloading mechanism
gSystem.Load("libFWCoreFWLite.so")
AutoLibraryLoader.enable()

# open the file and access the event tree
events = EventTree("reco.root")

# loop over the events
for event in events:
    prod = event.getProduct(productName)
    source = event.source
    ...
```

Compiling a FWLite macro in ROOT "on the fly"

You may want to compile your FWLite macros because ROOT C++ interpreter CINT is not capable of dealing with many standard c++ features. Here is how to compile your macro "on the fly" within a ROOT interactive session.

Using the full CMSSW Framework

The previously described approaches involve only accessing existing event data in a ROOT file. The full framework adds to this the steps by which event data are placed in the ROOT file. This includes pulling in

analysis code modules by way of parameter sets (in the form of configuration files), and running the cmsRun executable on the given parameter set thereby creating or modifying event data (in one data format or another, depending on the stage of analysis). This process exploits the full framework.

In this tutorial, we'll illustrate the full framework analysis process by way of:

- writing an EDAnalyzer module
- including the EDAnalyzer in the cfg file
- starting cmsRun to run with the cfg file
- looking at the output

Write an EDAnalyzer

The important parts of an EDAnalyzer are the following three methods:

```
void beginJob( const edm::EventSetup& );
void analyze( const edm::Event& , const edm::EventSetup& );
void endJob();
```

In these methods, respectively, you have to put in code that runs:

- at the beginning of the job (to book histograms)
- for every event (to fill histograms)
- at the end of the job (to save and plot histograms)

Now you can start writing your own C++ code. Writing a module is beyond the scope of this page. More information is available from the Introductory Framework Tutorial, and examples of more complete EDAnalyzer modules are given in high-level object tutorials and in WorkBookAnalysisStarterKit page.

Candidate Tools (Four-vector analysis)

In order to make a fast combinatorial analysis you can build simple objects containing 4-vectors and references to their RECO counterparts ("candidates"). In the following candidate example we will take all charged tracks found during reconstruction as candidates for particles.

The reconstructed tracks are already inside the `reco.root` file and called `allTracks`. This is a ready-made module providing the candidates which can be invoked by including the corresponding "cfi" file in the configuration file. There are many of them available (for tracks and clusters in the CMSSW_7_4_15 release and several others for later releases). The candidates are taken and filtered for `goodTracks` (transversal momentum higher than 3 GeV) in the `CandSelector` module. Then two tracks of opposite charge are combined to form a new 4-vector via a `CandCombiner` module . If the mass of the new 4-vector is inside a window of 70.0 - 110.0 GeV it is taken as a candidate for a `Z->l+l-` decay. Two of these Z candidates are then combined to produce a Higgs candidate. Finally the resulting Higgs candidates are plotted.

You can find an example configuration file in PhysicsTools/CandExamples/python directory. To run over your small local sample, change the `fileNames` variable to `"file:reco.root"`. You can also access directly the file in the CERN local storage, in this case, use the same syntax as in the previous `copy.cfg` file. Run the example with `h4l_cfg.py` (lower case "L") as configuration file:

```
cmsRun h4l_cfg.py
```

As a result, this job will produce a `candidates.root` file. To make some plots of your results, save the

following commands into a file `higgsPlots.C`:

```
{
TFile f("candidates.root");
TCanvas c;

TH1F higgsMass( "higgsMass", "Higgs mass", 100, 150, 250 );
Events.Project( "higgsMass", "HiggsCandidates.data_.mass()");
higgsMass.SetFillColor( kRed );
higgsMass.SetLineWidth( 2 );
higgsMass.GetXaxis()->SetTitle( "4l mass (GeV/c^{2})" );
higgsMass.Draw();
c.SaveAs("higgsMass.gif");

TH1F zMass( "zMass", "Z mass", 100, 70, 110 );
Events.Project( "zMass", "Ztoll.data_.mass()");
zMass.SetFillColor( kBlue );
zMass.SetLineWidth( 2 );
zMass.GetXaxis()->SetTitle( "l^{+}l^{-} mass (GeV/c^{2})" );
zMass.Draw();
c.SaveAs("zMass.gif");
}
```

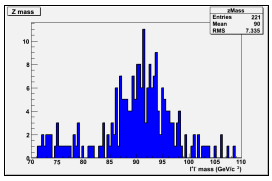
Start ROOT with

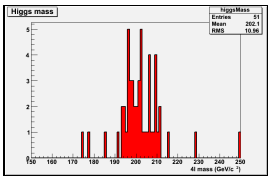
```
root
```

Run the two commands in order to load the libraries as explained in Set up ROOT, if they are not already in your `$HOME/rootlogon.C` file. Run this macro from the ROOT command line with

```
.x higgsPlots.C
```

Look at the invariant mass plots in Gif format:





Information Sources

https://twiki.cern.ch/twiki/bin/view/CMS/May06CPTweekAnalysis

Review status

Reviewer/Editor and Date (copy from screen)	Comments
JennyWilliams - 14 Nov 2007	Included Benedikt's updates for CMSSW_1_6_0
JennyWilliams - 12 Nov 2007	Updated for CMSSW_1_6_0
BenediktHegner - 20 Dec 2006	Cleaned up for print.
JennyWilliams - 4 Dec 2006	Added instructions from Ian Tomalin to get event id when working in ROOT mode.
AnneHeavey - 18 Jul 2006	Edited after Benedikt updated the tutorial

Responsible: BenediktHegner
Last reviewed by: PetarMaksimovic - 15 Feb 2008

This topic: CMSPublic > WorkBookMakeAnalysis
Topic revision: r93 - 2009-10-15 - SudhirMalik