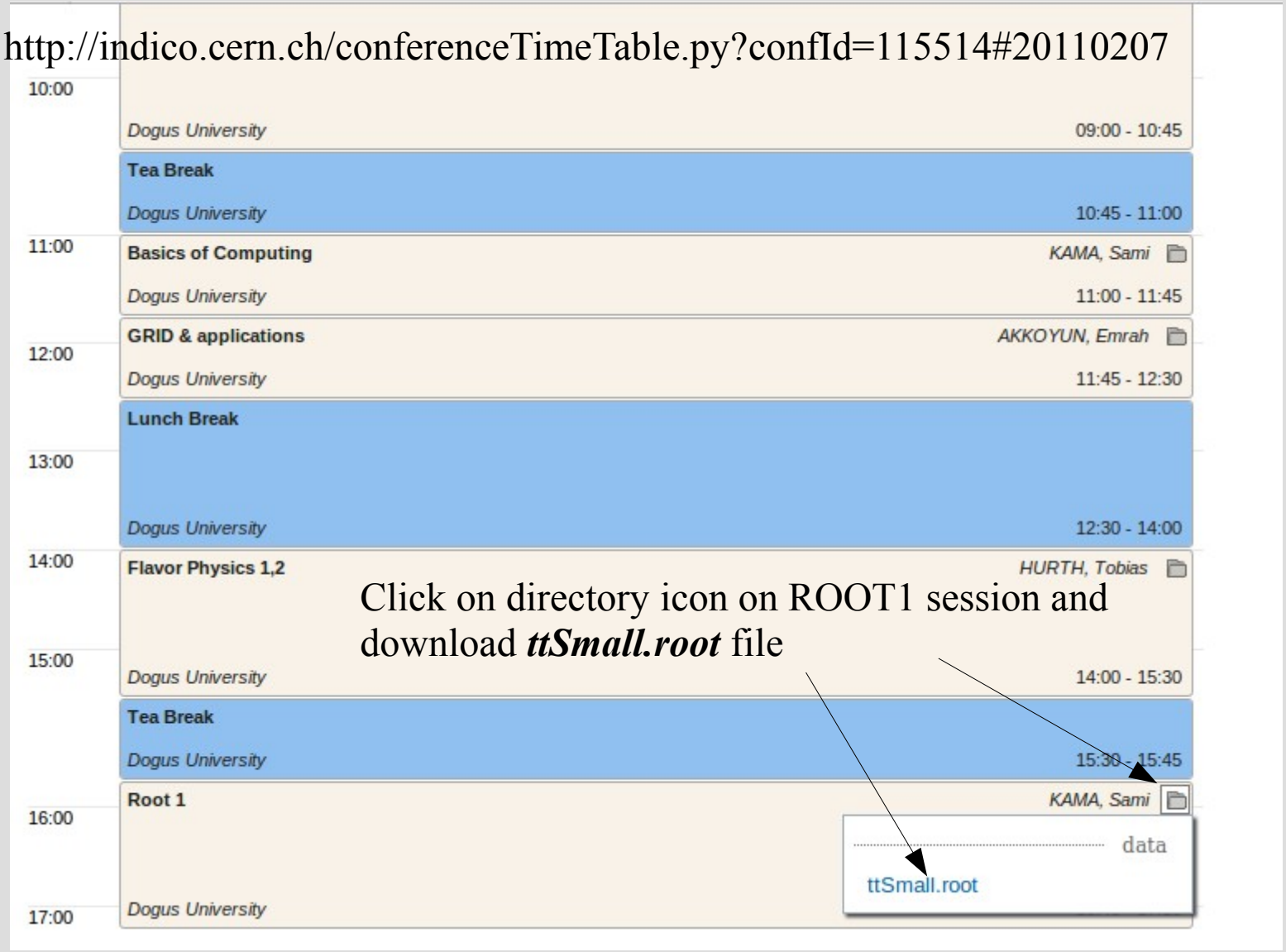
ROOT1 Hands-on

Sami Kama

Download root file from indico.

http://indico.cern.ch/conferenceTimeTable.py?confId=115514#20110207



Open the file you dowloaded

Default download location is ${HOME}/Desktop/ . cd to where your file is.

You can access the TFile object itself with **->**. This line will list the contents of the file. Notice _ (underscore) infront of the ***file0***.

If you type-in the name of a variable in CINT, it will print its value. This line tells that ***myTest*** is a ***TTree**** which is pointing to the memory address 0x8a4dbb8. It would be 0x0 if line [2] would have failed.

```
feynman@istapp2011:~$ cd Desktop/
feynman@istapp2011:~/Desktop$ root -l ttSmall.root
root [0]
Attaching file ttSmall.root as _file0...
root [1] _file0->ls()
TFile**         ttSmall.root
 TFile*         ttSmall.root
  KEY: TTree    test;2  Reconst ntuple
  KEY: TTree    test;1  Reconst ntuple
root [2] TTree* myTest=(TTree*)_file0->Get("test")
root [3] myTest
(class TTree*)0x8a4dbb8
root [4] myTest->MakeClass("MyTestTreeReaderClass")
Info in <TTreePlayer::MakeClass>: Files: MyTestTreeReaderC
lass.h and MyTestTreeReaderClass.C generated from TTree: t
est
(Int_t)0
root [5]
```

Start ROOT with file name. This will open the file and assign its memory address to ***_file0*** variable

This line says that, create a new variable of type TTree* called ***myTest*** and initialize it to value which is the result of the ***_file0->Get('test')*** call treated as a ***TTree****.

We are calling MakeClass() method of TTree class for our object myTest. This line will create two files ***MyTestTreeReaderClass.h*** and ***MyTestTree*****ReaderClass.C** which will contain skeleton code for us to read the ROOT tree pointed by myTree and implement our analysis.

MyTestTreeReaderClass.h

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Wed Feb  9 16:32:47 2011 by ROOT version 5.28/00
// from TTree test/Reconst ntuple
// found on file: ttSmall.root
//////////////////////////////////////////////////////////

#ifndef MyTestTreeReaderClass_h
#define MyTestTreeReaderClass_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>

class MyTestTreeReaderClass {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain
   Int_t           fCurrent; //!current Tree number in a TChain

   // Declaration of leaf types
   Int_t           ntrack;
   Int_t           stat[498];   //[ntrack]
   Int_t           sign[498];   //[ntrack]
   Float_t         px[498];   //[ntrack]
   Float_t         py[498];   //[ntrack]
   Float_t         pz[498];   //[ntrack]
   Float_t         zv[498];   //[ntrack]
   Float_t         chi2[498];   //[ntrack]

   // List of branches
   TBranch        *b_ntrack;   //!
   TBranch        *b_stat;   //!
---  MyTestTreeReaderClass.h   Top L16    (C/l Abbrev)----------------
```

Header file created by MakeClass() method. All necessary variables are automatically defined. Main implementation goes in .C file.

In order to keep memory consumption minimum ROOT assigns the maximum encountered number of ***ntracks*** to the length of the arrays. You need to increase this numbers if you are going to use chains and there is a probability of having more ***ntracks*** for some entries.

MyTestTreeReaderClass Constructor

```
   MyTestTreeReaderClass(TTree *tree=0);
   virtual ~MyTestTreeReaderClass();
   virtual Int_t    Cut(Long64_t entry);
   virtual Int_t    GetEntry(Long64_t entry);
   virtual Long64_t LoadTree(Long64_t entry);
   virtual void     Init(TTree *tree);
   virtual void     Loop();
   virtual Bool_t   Notify();
   virtual void     Show(Long64_t entry = -1);
};

#endif

#ifdef MyTestTreeReaderClass_cxx
MyTestTreeReaderClass::MyTestTreeReaderClass(TTree *tree)
{
// if parameter tree is not specified (or zero), connect the file
// used to generate this class and read the Tree.
   if (tree == 0) {
      TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject("ttSmall.root");
      if (!f) {
         f = new TFile("ttSmall.root");
      }
      tree = (TTree*)gDirectory->Get("test");

   }
   Init(tree);
}
```

Automatically generated minimum list of methods. You can add more methods or variables for your analyses. For example you can add pointers to your histograms as member variables and createHistograms() and saveHistograms() methods to initialize histograms before analysis and save them after it.

Class constructor takes a pointer to TTree object. You can pass a TTree* that you initialized elsewhere. Or you can leave it empty, in which case ROOT will try to open the file you used in MakeClass() process and the tree it is created from.

MyTestReaderClass.C

```
#define MyTestTreeReaderClass_cxx
#include "MyTestTreeReaderClass.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void MyTestTreeReaderClass::Loop()
{
//   In a ROOT session, you can do:
//      Root > .L MyTestTreeReaderClass.C
//      Root > MyTestTreeReaderClass t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
---   MyTestTreeReaderClass.C   Top L31    (C++/l Abbrev)------------------
```

Implementation class automatically includes the header file TCanvas, TStyle, and TH2 for you.

Loop() method is the main method that you should implement your analysis in unless you think of using PROOF. It will loop over all entries in a tree.

Comments tell you how you can use the class in CINT and leave out unwanted branches.

Loop() method.

```
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
}
```

Get the number of entries in the tree.

Loop over all entries

Select the ***jentry***'th entry

Reset the values of tree variables to the values of the ***jentry***'th entry.

Default skeleton of Loop() method just loops over all entries in the tree without doing any work. You can implement your analysis after the ***// if(Cut(ientry)<0)continue;*** line.

A basic analysis implementation

```
//by  b_branchname->GetEntry(ientry); //read only this branch
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();
   Long64_t nbytes = 0, nb = 0;

   TH2F *momDist=new TH2F("momDist","Momentum Distribution",100,-10.,10.,100,-10.,10.);
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      for(int currentParticle=0;currentParticle<ntrack;currentParticle++){
        if(zv[currentParticle]>105.)continue; //skip particles far from vertex
        momDist->Fill(px[currentParticle],py[currentParticle]);
      }
      // if (Cut(ientry) < 0) continue;
   }
   momDist->Draw();
}
```

Declare a 2D histogram to book px and py of the selected particle

Loop over all tracks in the event. ***ntrack*** will set to number of tracks in current event at every call to GetEntry(). Don't forget ***ntrack*** is specific to this example. It will be different for different trees and there may be more than one loop variable. The header file tells you which variables contain the number of entries in which arrays.

Check if the zv (z vertex) of the current particle is far away from interaction vertex. Skip particles that are further than 105. mm.

Fill the px and py of the selected particles in momDist histogram.

Draw the final histogram

Don't forget that this is a mock-up tree. You will write analysis for meaningful trees later in the data analysis lectures.

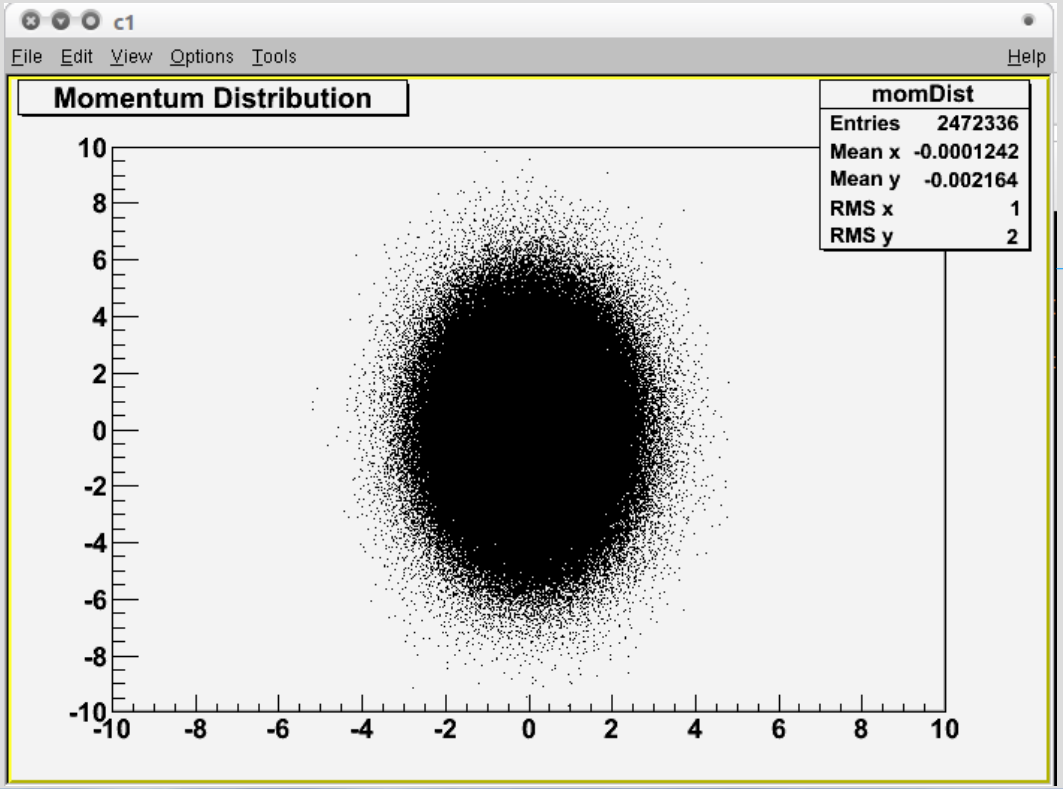
Running the analysis

cd to where the class and root file is and start ROOT.

Load your class with ***.L MyTestTreeReaderClass.C.*** Then ROOT will know your class and you can instantiate it like any other class. Unless you pass a TTree* it constructor, your class will try to open the file you used in its creation. If you get an error message, make sure that the **MyTestTreeReaderClass.C** and **ttSmall.root** file are in the current directory or pass a pointer to **„test“** tree to constructor. (lines [1] and [2] in page 3)

```
feynman@istapp2011:~$ cd Desktop/
feynman@istapp2011:~/Desktop$ root -l
root [0] .L MyTestTreeReaderClass.C
root [1] MyTestTreeReaderClass myReaderInstance
root [2] myReaderInstance.Loop()
Info in <TCanvas::MakeDefCanvas>:  created defau
root [3]
```

Then call your Loop method. It will execute the analysis code that you wrote.



Since we call Draw function at the end, Loop() method will draw momDist histogram.

You can create multiple histograms in your analyses and save them into a file by opening a new file before histogram declaration and then calling ***Write()*** method of opened file at the end.

Take a Look at

- At least *Histograms*, *Graphs* and *Trees* chapters of the ROOT users manual.
- ROOT reference manual of the classes that we used here.
- ***tutorials*** and ***tests*** directories in ROOT sources.

Thank You