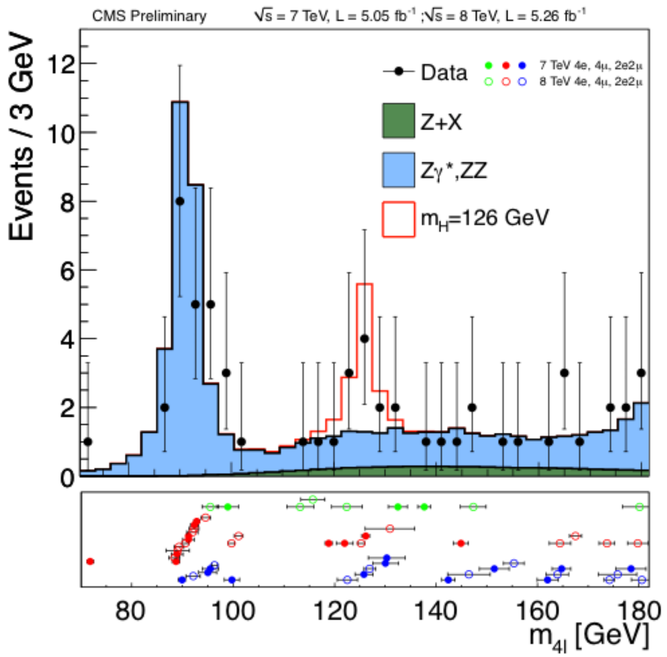
ROOT and its Applications

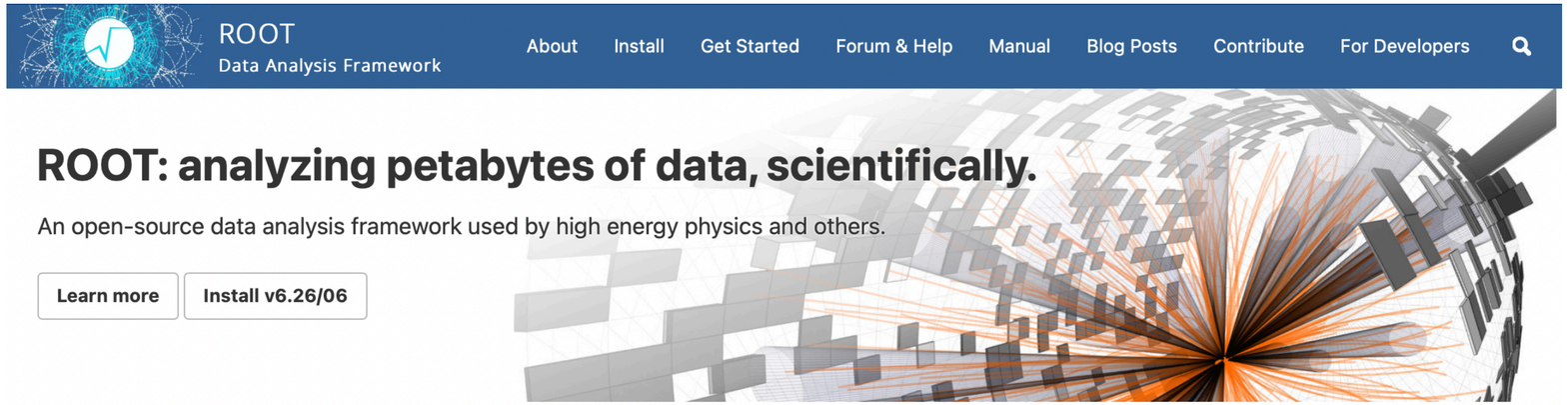
Data Storage and I/O, Generating Random Numbers, Histograms

ROOT: An Object-Oriented Data Analysis Framework

https://root.cern

Reference guide is all you need!
You are a user, exploiting code written by others





Installing ROOT

https://root.cern/install/

Installing ROOT

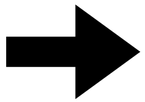
ROOT is available on Linux, Mac, and (as a beta release) on Windows.
The latest stable ROOT release is 6.26/06 (about ROOT versioning scheme).

There are several ways to install ROOT on your computer: they are all listed in the table of content on the right. Which one is best for you depends on your operating system and usage requirements. In all cases, make sure to always use the most recent ROOT release possible to get the latest bug fixes, features and quick user support.

Download a pre-compiled binary distribution

We distribute pre-compiled ROOT for several major Linux distributions as well as MacOS and (as a beta) Windows. The steps to install a pre-compiled binary are simple:

1. Install all required dependencies with the system package manager
2. Download the release for the desired platform and ROOT version
3. Unpack the archive
4. Add the ROOT libraries and executables to your environment by sourcing the appropriate `thisroot.*` script. These setup scripts can be found in the ROOT binary release, in the `bin` directory.



→ On this page

Using ROOT Libraries and Header Files

Based on `examples/09/RootApp1.cpp`

```
csh> setenv ROOTSYS /Users/francesco/Library/root/v6.24.00/

bash> export ROOTSYS=/Users/francesco/Library/root/v6.24.00/


csh> setenv LD_LIBRARY_PATH $ROOTSYS/lib

bash> export LD_LIBRARY_PATH=$ROOTSYS/lib


$ g++ -o RootApp1 RootApp1.cpp `$ROOTSYS/bin/root-config --cflags --libs`
$ ./RootApp1
TH1.Print Name  = , Entries= 0, Total sum= 0
```

Needed at runtime to find libraries

Provide path to header files and libraries

```
#include  "TH1F.h"

int main() {

  TH1F h1;
  h1.Print();

  return 0;
}
```

root-config

```
> $ROOTSYS/bin/root-config --cflags --libs
-pthread -stdlib=libc++ -std=c++11 -m64 -I/Users/francesco/Library/root/v6.24.00/include
-L/Users/francesco/Library/root/v6.24.00/lib -lCore -lImt -lRIO -lNet -lHist -lGraf
-lGraf3d -lGpad -lROOTDataFrame -lROOTVecOps -lTree -lTreePlayer -lRint -lPostscript
-lMatrix -lPhysics -lMathCore -lThread -lMultiProc -lpthread -stdlib=libc++ -lm -ldl
```

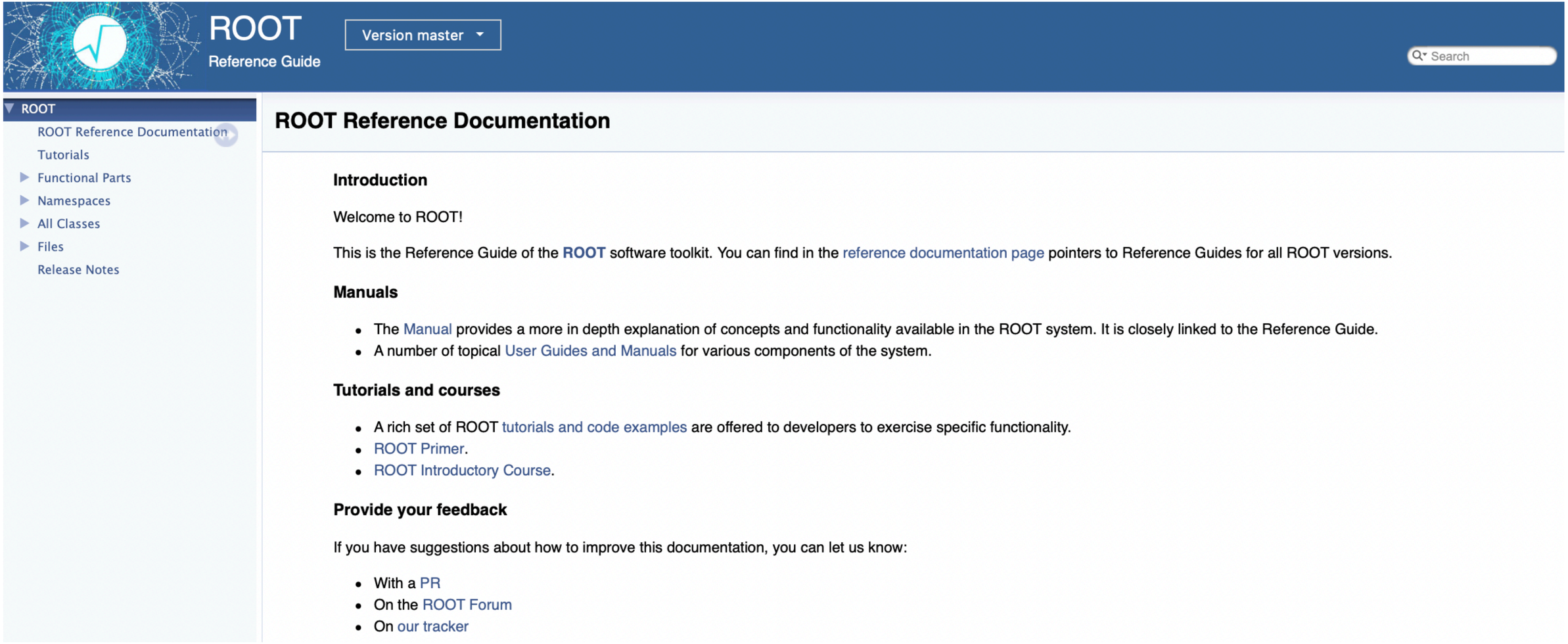
- Provides you with all options needed to compile and/or link your application
- Use on command line with `` quotes instead of writing manually

Using ROOT

- Use classes from ROOT in your application
  - 1D histograms to plot your data
  - Provided canvas to store the histogram as output in a file (.eps or .gif)
  - Functionalities to make your plot nicer: e.g., change color, labels, names, fonts, …
- **Become familiar with using external libraries without access to source files**
  - Look at the reference guide to find out what is provided by the interface
  - Start by simply creating new objects and testing them before making fancy use of many different classes

ROOT Reference Guide

https://root.cern/doc/master/



First Classes to Use

- `TH1F`: 1D histogram
  - look at constructors
  - public methods to add data to histogram
  - public methods to add comments or change labels of axes

- `TCanvas`: canvas to draw your histogram
  - how to make one
  - changing properties such as color
  - drawing 1D histogram on a canvas
  - storing the canvas as a graphic file, e.g. eps or gif

- All ROOT class names start a 'T' with the notable exception of `RooFit` (in this context all class names are of the form `Roo*`)

`TH1F`: 1D Histograms of Floats

https://root.cern/doc/master/classTH1F.html

ROOT Reference Guide — Version master — Search

- ROOT Reference Documentation
- Tutorials
- Functional Parts
  - Core ROOT classes
  - The Geometry Package
  - Graphics
  - GUI
  - Web Widgets
  - Web Display
  - Histogram Library
    - Painting classes
    - Histogram classes.
      - TAxis
      - TH1
      - TH1C
      - TH1D
      - **TH1F**
      - TH1I
      - TH1S
      - TH2C
      - TH2D
      - TH2F
      - TH2I
      - TH2Poly
      - TH2PolyBin
      - TH2S
      - TH3
      - TH3C
      - TH3D
      - TH3F
      - TH3I
      - TH3S

List of all members | Public Member Functions | Protected Member Functions | Friends | List of all members

TH1F Class Reference

Histogram Library » Histogram classes.

1-D histogram with a float per channel (see TH1 documentation)}

Definition at line 575 of file TH1.h.

Public Member Functions

	TH1F () Constructor. More...
	TH1F (const char *name, const char *title, Int_t nbinsx, const Double_t *xbins) Create a 1-Dim histogram with variable bins of type float (see TH1::TH1 for explanation of parameters) More...
	TH1F (const char *name, const char *title, Int_t nbinsx, const Float_t *xbins) Create a 1-Dim histogram with variable bins of type float (see TH1::TH1 for explanation of parameters) More...
	TH1F (const char *name, const char *title, Int_t nbinsx, Double_t xlow, Double_t xup) Create a 1-Dim histogram with fix bins of type float (see TH1::TH1 for explanation of parameters) More...
	TH1F (const TH1F &h1f) Copy Constructor. More...
	TH1F (const TVectorF &v) Create a histogram from a TVectorF by default the histogram name is "TV
virtual	**~TH1F** () Destructor. More...
virtual void	**AddBinContent** (Int_t bin) Increment bin content by 1. More...
virtual void	**AddBinContent** (Int_t bin, Double_t w) Increment bin content by a weight w. More...
virtual void	**Copy** (TObject &hnew) const

◆ TH1F() [2/6]

Interface and libraries are all you need

```
TH1F::TH1F ( const char *  name,
             const char *  title,
             Int_t         nbinsx,
             Double_t      xlow,
             Double_t      xup
           )
```

Create a 1-Dim histogram with fix bins of type float (see TH1::TH1 for explanation of parameters)

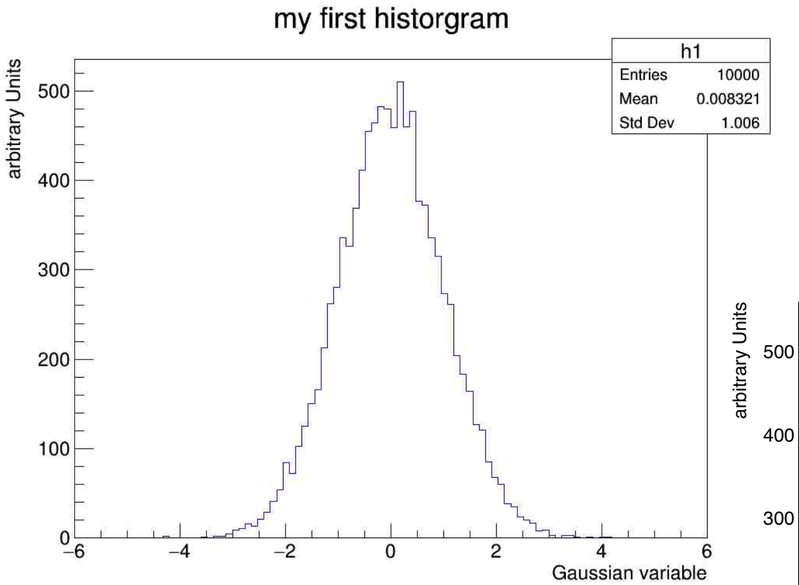
Definition at line 9876 of file TH1.cxx.

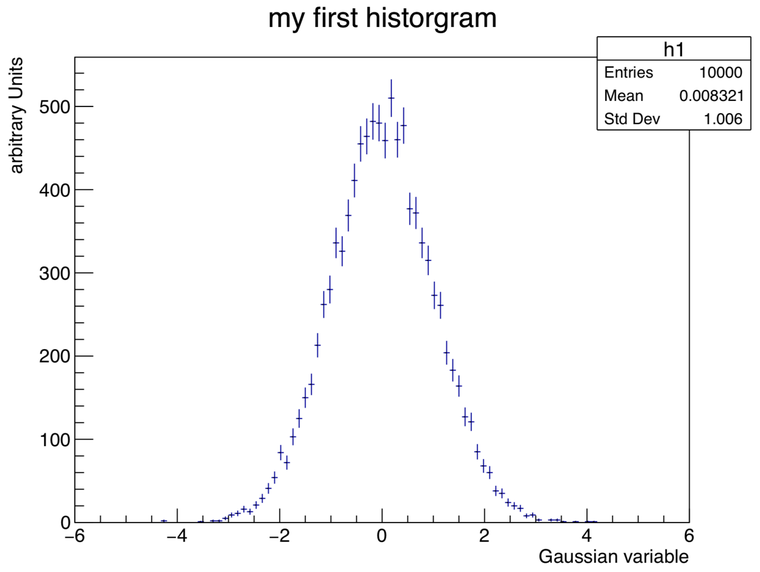
```
9874 ////////////////////////////////////////////////////////////////////////////
9875 /// Create a 1-Dim histogram with fix bins of type float
9876 /// (see TH1::TH1 for explanation of parameters)
9877
9878 TH1F::TH1F(const char *name,const char *title,Int_t nbins,Double_t xlow,Double_t xup)
9879 : TH1(name,title,nbins,xlow,xup)
9880 {
9881    fDimension = 1;
9882    TArrayF::Set(fNcells);
9883
9884    if (xlow >= xup) SetBuffer(fgBufferSize);
9885    if (fgDefaultSumw2) Sumw2();
9886 }
```

Simple Example with TH1

Based on `examples/09/RootApp2.cpp`

```
$ g++ -Wall -o RootApp2 RootApp2.cpp `root-config --libs --cflags`
$ ./RootApp2
```






```
#include "TH1F.h"
#include "TCanvas.h"

int main() {

  // create histogram
  TH1F h1("h1","my first historgram",100,-6.0,6.0);

  // fill histogram with 10000 random gaussian numbers
  h1.FillRandom("gaus",10000);

  // add labels to axis
  h1.GetXaxis()->SetTitle("Gaussian variable");
  h1.GetYaxis()->SetTitle("arbitrary Units");

  // create a canvas to draw tower histogram
  TCanvas c1("c1","my canvas",1024,800);

  // draw the histogram
  h1.Draw();

  // save canvas a JPG file
  c1.SaveAs("canvas.jpg");

  // change fill color to blue
  h1.SetFillColor(kBlue);

  // draw again the histogram
  h1.Draw();
  c1.SaveAs("canvas-blue.jpg");

  // draw histograms as points with errors
  h1.Draw("pe");

  // save canvas a PDF file
  c1.SaveAs("canvas-points.pdf");

  return 0;
}
```

I/O: Data Storage with `TFile`

https://root.cern.ch/doc/master/classTFile.html

- Text files are not a viable solution for input/output on the long run
  - No scaling when a large sample of data is involved
  - How does one store objects of custom types instead of built-in types?

- **Binary files** with dedicated storage containers provide an efficient solution
  - `TFile` is the File class provided in ROOT for storage of data (see user guide)
    - `TFile` is similar to a UNIX directory
    - It can contain directories and objects
    - Objects are stored in machine independent format
    - To use a `TFile` object you need to include its header

```
// ... some code...
#include “TFile.h"
// ... more code...
TFile f("outputfile.root");
```

Generation of Random Numbers with `TRandom`

https://root.cern.ch/doc/master/classTRandom.html

- ROOT has several random number generators: we will use the `TRandom3` class
- `TRandom` provides generators for a number of commonly used functions
  - Uniform
  - Gaussian
  - Exponential
  - Breit-Wigner
  - Binomial
  - Poisson
  - Landau
- `TRandom` also has functions to generate custom random numbers
  - from an existing histogram
  - from a custom function

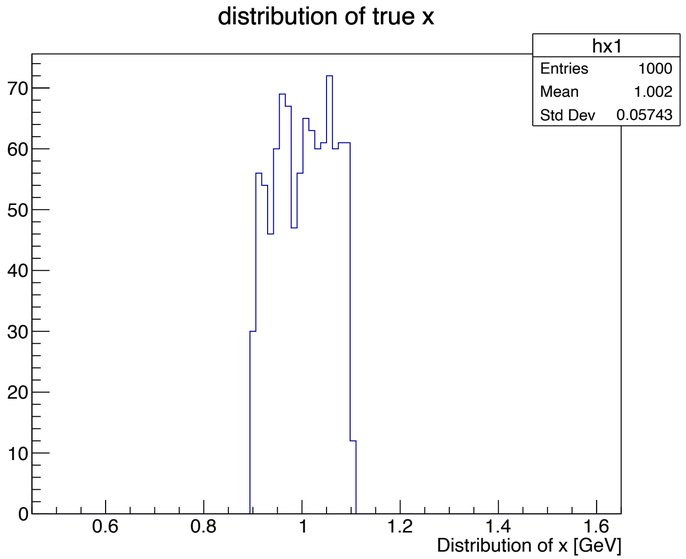
Example: Filling Histograms and Storage to `TFile`

- Based on `examples/09/01-HistoStorage.cpp`
- This relies on `TRandom3.h`, `TH1F.h`, `TCanvas.h`, `TFile.h`
- The code is properly commented so we will go through the file itself.

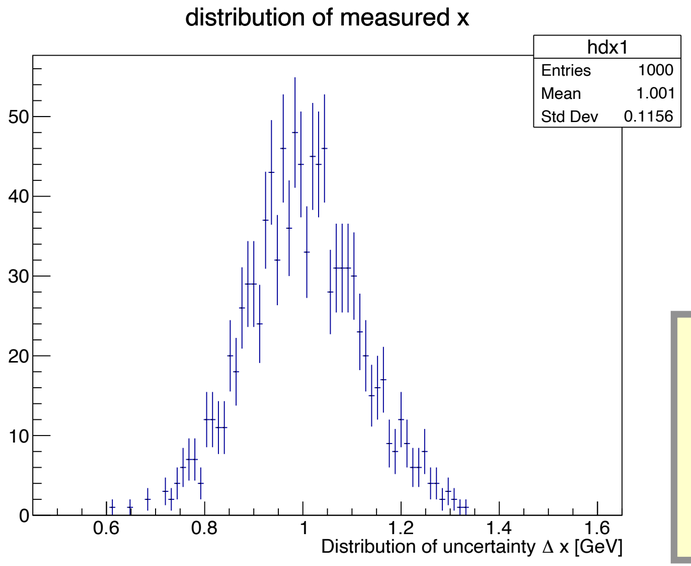
```
$ export ROOTSYS=/opt/homebrew/Cellar/root/6.24.06/
$ export LD_LIBRARY_PATH=$ROOTSYS/lib

$ g++ -o 01-HistoStorage 01-HistoStorage.cpp `$ROOTSYS/bin/root-config --cflags --libs`
$ ./01-HistoStorage
# bins: 100        bin width: 0.012
Info in <TCanvas::Print>: pdf file ./x.pdf has been created
Info in <TCanvas::Print>: jpg file ./x.jpg has been created
Info in <TCanvas::Print>: pdf file ./dx.pdf has been created
Info in <TCanvas::Print>: jpg file ./dx.jpg has been created
 FCN=35.7155 FROM MIGRAD    STATUS=CONVERGED      62 CALLS           63 TOTAL
                     EDM=5.73921e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     4.01120e+01   1.69528e+00   3.82139e-03  -1.40032e-04
   2  Mean         1.00055e+00   3.83783e-03   1.13365e-05  -6.30094e-02
   3  Sigma        1.15889e-01   3.34364e-03   2.17068e-05   1.50102e-03
Info in <TCanvas::Print>: pdf file ./dxfit.pdf has been created
Info in <TCanvas::Print>: jpg file ./dxfit.jpg has been created
```

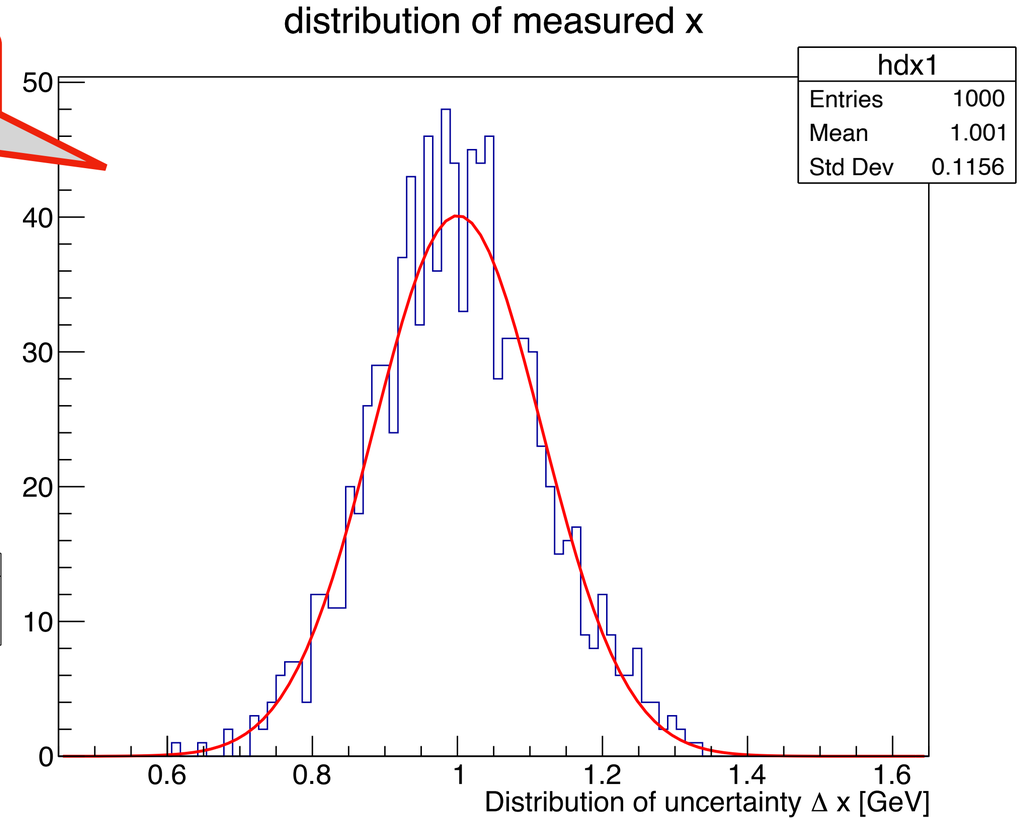
Syntax is shell-dependant: for the virtual machine you do not need to worry about this

Example: Filling Histograms and Storage to `TFile`



User guide on fitting histograms

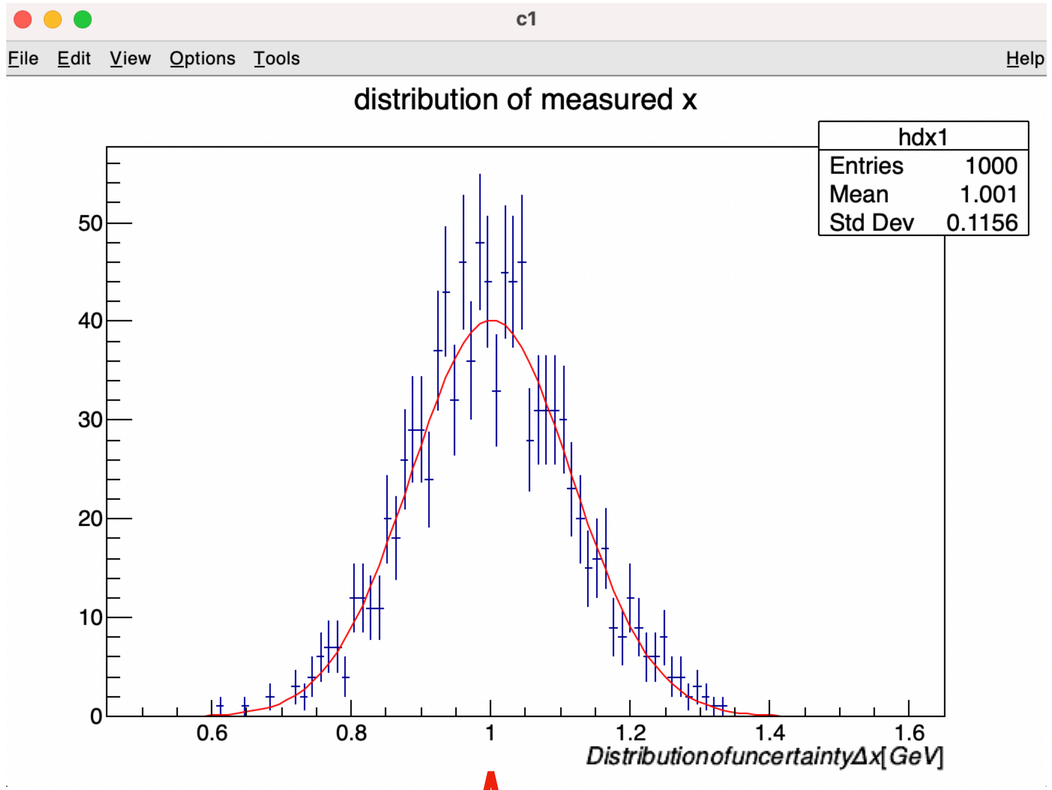





```
EXT PARAMETER
 NO.   NAME      VALUE            ERROR
   1  Constant     4.01120e+01   1.69528e+00
   2  Mean         1.00055e+00   3.83783e-03
   3  Sigma        1.15889e-01   3.34364e-03
```

Inspecting a `TFile` Storage File

```
$ root ./output.root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.24/04                          https://root.cern |
  | (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |
  | Built for macosxarm64 on Aug 25 2021, 12:59:28                   |
  | From tags/v6-24-04@v6-24-04                                      |
  | With Apple clang version 12.0.5 (clang-1205.0.22.9)              |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
   ------------------------------------------------------------------

root [0]
Attaching file ./output.root as _file0...
(TFile *) 0x131f5be20
root [1] .ls
TFile**                 ./output.root
 TFile*                 ./output.root
  KEY: TH1F    hx1;1     distribution of true x
  KEY: TH1F    hdx1;1    distribution of measured x
root [2] hdx1->Draw("pe")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```



Interactive object

`TTree` for Data Storage

https://root.cern.ch/doc/master/classTTree.html

- The `TTree` class is designed to store a large number of objects of the same class into a file (detailed user guide)

- The **tree** is the prototype of the data structure to be stored
  - It consists of **branches**, each containing a distinct collection of objects of the same class: in data analysis, variable ↔ branch
    - Each branch can have as many **leaves** as your data points

- A number of examples is provided to write and read simple types of trees:
  - Fill `TTree` and store in `TFile` (uses `Datum`): `examples/09/02-WriteTree.cpp`
  - Read `TTree` from file and fill histograms (uses `Datum`): `examples/09/03-ReadTree.cpp`
  - Fill `TTree` with variable size branch and store in `TFile`: `examples/09/04-WriteObjects.cpp`
  - Read `TTree` from file and fill 1D and 2D histograms (uses `Datum`): `examples/09/05-ReadTree.cpp`

Writing Trees Containing Custom Objects

https://root.cern.ch/doc/master/classTTree.html

- To store custom objects, we must tell ROOT about the custom class and its interface
  - This requires generating a Dictionary for the class, e.g., suppose you store `Datum` objects (as leaves) in `TTree` branches
    - Generate dictionary for `Datum`:

```
$ROOTSYS/bin/rootcint -f MyDict.cxx Datum.h
```

    - This creates a new file `MyDict.cxx` that can be checked

```
$ g++ -c `$ROOTSYS/bin/root-config --cflags` MyDict.cxx
$ ls MyDict*
MyDict.cxx        MyDict.o          MyDict_rdict.pcm
```

- Example program to store `Datum` in branches: `examples/09/06-WriteCustomObject.cpp`

```
$ g++ -o 06-WriteCustomObject 06-WriteCustomObject.cpp Datum.cc MyDict.cxx `$ROOTSYS/bin/root-config --libs --cflags`
```

Reading Trees Containing Custom Objects

https://root.cern.ch/doc/master/classTTree.html

- Example program to read branches containing `Datum` instances: `examples/09/07-ReadCustomObject.cpp`

```
$ g++ -o 07-ReadCustomObject 07-ReadCustomObject.cpp Datum.cc MyDict.cxx `$ROOTSYS/bin/root-config --libs --cflags`
$ ./07-ReadCustomObject
Reading data from root file ./data.root
******************************************************************************
*Tree    :datatree  : tree containing our data                                 *
*Entries :      100 : Total =            3757 bytes  File  Size =       2379 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Branch  :datum                                                               *
*Entries :      100 : BranchElement (see below)                               *
*.............................................................................*
*Br    0 :value_    : Double_t                                                *
*Entries :      100 : Total  Size=       1370 bytes  File Size  =        877 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.00      *
*.............................................................................*
*Br    1 :error_    : Double_t                                                *
*Entries :      100 : Total  Size=       1370 bytes  File Size  =        877 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.00      *
*.............................................................................*
Info in <TCanvas::Print>: pdf file ./newplots.pdf has been created
```

Using `TTrees` Created by Others

https://root.cern.ch/doc/master/classTTree.html

- Typically one needs to analyse data stored in a `TTree` created by someone else
- In principle, writing an application to read the `TTree` requires the list and types of all branches in order to `BranchSetAddress` for each one
  - This can be painful, but luckily ROOT can automatically generated this part
- Assume you have `data.root` created by `examples/09/04-WriteObjects.cpp`
  - Let's inspect it and create a skeleton for a class based on the tree object

Using `TTrees` Created by Others

https://root.cern.ch/doc/master/classTTree.html

```
$ root ./data.root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.24/04                          https://root.cern |
  | (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |
  | Built for macosxarm64 on Aug 25 2021, 12:59:28                   |
  | From tags/v6-24-04@v6-24-04                                      |
  | With Apple clang version 12.0.5 (clang-1205.0.22.9)              |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
   ------------------------------------------------------------------

root [0]
Attaching file ./data.root as _file0...
Warning in <TClass::Init>: no dictionary for class Datum is available
(TFile *) 0x12f289ff0
root [1] .ls
TFile**                         ./data.root
 TFile*                         ./data.root
  KEY: TTree     datatree;1      tree containing our data
```

```
root [2] datatree->Print()
******************************************************************************
*Tree    :datatree  : tree containing our data                                *
*Entries :     1000 : Total =          812989 bytes  File  Size =     755669 *
*        :          : Tree compression factor =   1.07                       *
******************************************************************************
*Br    0 :nmeas     : nmeas/I                                                 *
*Entries :     1000 : Total  Size=       4561 bytes  File Size  =       1250 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   3.26     *
*.............................................................................*
*Br    1 :value     : value[nmeas]/D                                          *
*Entries :     1000 : Total  Size=     404092 bytes  File Size  =     376651 *
*Baskets :       13 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
*Br    2 :error     : error[nmeas]/D                                          *
*Entries :     1000 : Total  Size=     404092 bytes  File Size  =     376943 *
*Baskets :       13 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
root [3] datatree->MakeClass("DataTree");
Info in <TTreePlayer::MakeClass>: Files: DataTree.h and DataTree.C generated
from TTree: datatree
(int) 0
```

Let's inspect `DataTree.h` and `DataTree.C`

Using `TTrees` Created by Others: `DataTree.h`

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Mon Nov  1 21:48:30 2021 by ROOT version 6.24/04
// from TTree datatree/tree containg our data
// found on file: ./data.root
//////////////////////////////////////////////////////////

#ifndef DataTree_h
#define DataTree_h

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

// Header file for the classes stored in the TTree if any.

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

// Fixed size dimensions of array or collections stored in the TTree if any.

   // Declaration of leaf types
   Int_t           nmeas;
   Double_t        value[78];   //[nmeas]
   Double_t        error[78];   //[nmeas]

   // List of branches
   TBranch        *b_nmeas;   //!
   TBranch        *b_value;   //!
   TBranch        *b_error;   //!

   DataTree(TTree *tree=0);
   virtual ~DataTree();
   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
```

- 78 is a non-typical value
  - It is the maximum length of the array in the tree
  - You need to modify this by hand and set it to a large value based on information provided to you by whoever created the root file (change this to 200)
  - This is yet another unfortunate example of static C arrays

Using `TTrees` Created by Others: `DataTree.h`

```
#ifdef DataTree_cxx
DataTree::DataTree(TTree *tree) : fChain(0)
{
// 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("./data.root");
      if (!f || !f->IsOpen()) {
         f = new TFile("./data.root");
      }
      f->GetObject("datatree",tree);

   }
   Init(tree);
}
```

```
#ifdef DataTree_cxx
DataTree::DataTree(TTree *tree) : fChain(0)
{
// if parameter tree is not specified (or zero), connect the file
// used to generate this class and read the Tree.
   if (tree == 0) {
        std::cout << "No tree has been provided. exiting!" << std::endl;
        exit(-1);
   }



   Init(tree);
}
```

- `DataTree.h` implements also most of the class member functions
- The constructor needs a pointer to a `TTree` in order to build a new `DataTree` object
  - Could be better to modify default behaviour to avoid problems in the future
    - Requires `#include <iostream>` in the preamble

Using `TTrees` Created by Others: `DataTree.h`

```
DataTree::~DataTree()
{
   if (!fChain) return;
   delete fChain->GetCurrentFile();
}

Int_t DataTree::GetEntry(Long64_t entry)
{
// Read contents of entry.
   if (!fChain) return 0;
   return fChain->GetEntry(entry);
}
Long64_t DataTree::LoadTree(Long64_t entry)
{
// Set the environment to read one entry
   if (!fChain) return -5;
   Long64_t centry = fChain->LoadTree(entry);
   if (centry < 0) return centry;
   if (fChain->GetTreeNumber() != fCurrent) {
      fCurrent = fChain->GetTreeNumber();
      Notify();
   }
   return centry;
}
```

- The rest of `DataTree.h` can be left unchanged
- In particular, the `Init()` function sets the Branch Address correctly for all branches in the tree

```
void DataTree::Init(TTree *tree)
{
   // The Init() function is called when the selector needs to initialize
   // a new tree or chain. Typically here the branch addresses and branch
   // pointers of the tree will be set.
   // It is normally not necessary to make changes to the generated
   // code, but the routine can be extended by the user if needed.
   // Init() will be called many times when running on PROOF
   // (once per file to be processed).

   // Set branch addresses and branch pointers
   if (!tree) return;
   fChain = tree;
   fCurrent = -1;
   fChain->SetMakeClass(1);

   fChain->SetBranchAddress("nmeas", &nmeas, &b_nmeas);
   fChain->SetBranchAddress("value", value, &b_value);
   fChain->SetBranchAddress("error", error, &b_error);
   Notify();
}
```

Using `TTrees` Created by Others: `DataTree.h`

```
Bool_t DataTree::Notify()
{
   // The Notify() function is called when a new file is opened. This
   // can be either for a new TTree in a TChain or when when a new TTree
   // is started when using PROOF. It is normally not necessary to make changes
   // to the generated code, but the routine can be extended by the
   // user if needed. The return value is currently not used.

   return kTRUE;
}

void DataTree::Show(Long64_t entry)
{
// Print contents of entry.
// If entry is not specified, print current entry
   if (!fChain) return;
   fChain->Show(entry);
}
Int_t DataTree::Cut(Long64_t entry)
{
// This function may be called from Loop.
// returns  1 if entry is accepted.
// returns -1 otherwise.
   return 1;
}
#endif // #ifdef DataTree_cxx
```

Using `TTrees` Created by Others: `DataTree.C`

```
#define DataTree_cxx
#include "DataTree.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void DataTree::Loop()
{
//   In a ROOT session, you can do:
//      root> .L DataTree.C
//      root> DataTree 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
   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;
   }
}
```

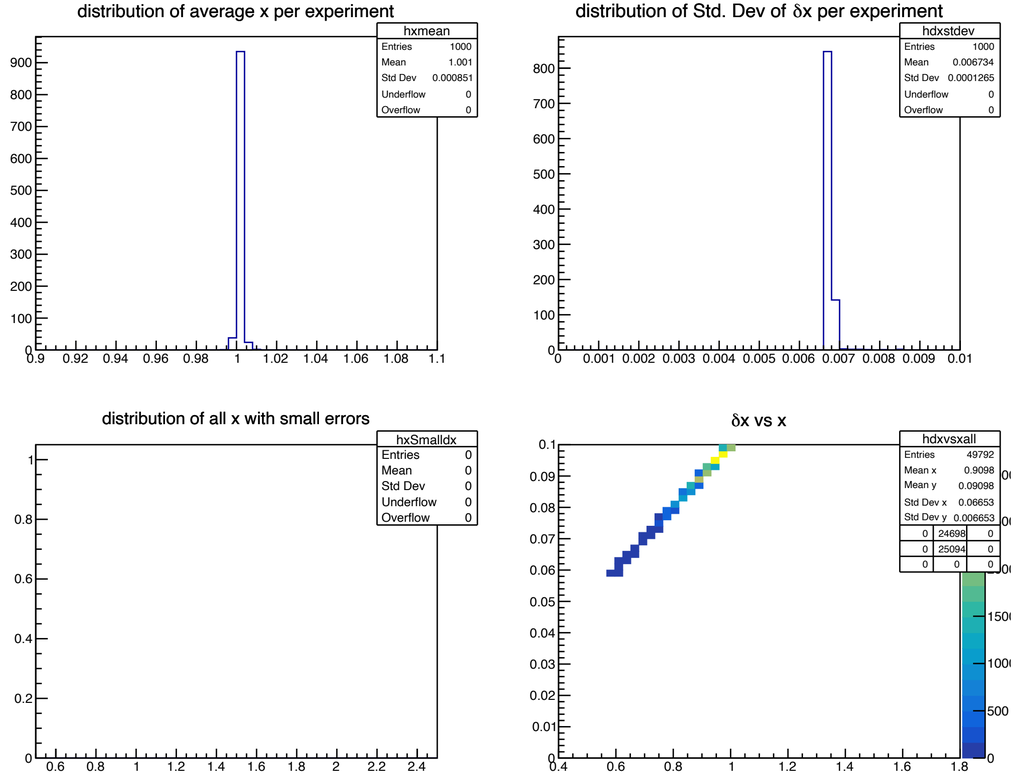
- `DataTree.C` implements the most important function `DataTree::Loop()` which is used to analyse the data in the `TTree`
- This is the file to be modify to add custom analysis code
- `Examples/09/DataTreeMod.C` has an example of a modified `Loop()` function that
  - opens a new output ROOT file
  - books histograms
  - loops over events
  - does some calculations
  - plots histograms and saves them in a pdf file
  - stores objects in the output ROOT file

Processing `TTrees` Created by Others

- Example program to use the customised Loop() method: `examples/09/08-AnalysisApp.cpp`

```
$ g++ -o 08-AnalysisApp 08-AnalysisApp.cpp DataTreeMod.C
`$ROOTSYS/bin/root-config --cflags --libs`
$ ./08-AnalysisApp
Reading data from root file ./data.root
processing event: 0
...
processing event: 995
Info in <TCanvas::Print>: pdf file ./analysisPlots.pdf
has been created
Done writing to ./analysis.root
```

Let's inspect this ROOT file



Kinematic Calculations with `TLorentzVector`

https://root.cern.ch/doc/master/classTLorentzVector.html

- `Examples/09/09-Boost.cpp` is an example of kinematic calculations and applying the relativistic boost

- It relies on the ROOT class `TLorentzVector`
  - handles 4-position and 4-momentum
  - provides (among other things) `Beta()` and `Gamma()` methods
  - calculates boost vector between reference frames (`BoostVector()`)
  - method to boosts particles `Boost(BoostVector)`

Kinematic Calculations with `TLorentzVector`

https://root.cern.ch/doc/master/classTLorentzVector.html

```
$ g++ -o 09-Boost 09-Boost.cpp `$ROOTSYS/bin/root-config --libs --cflags`
$ ./09-Boost
--> LAB p4 B:
(x,y,z,t)=(0.300000,0.000000,0.000000,5.287517) (P,eta,phi,E)=(0.300000,-0.000000,0.000000,5.287517)
--> CoM p4 pi*:
(x,y,z,t)=(0.000000,0.100000,0.000000,0.172047) (P,eta,phi,E)=(0.100000,-0.000000,1.570796,0.172047)
--> boost parameters of B reference frame
             beta: 0.0567374                    p/E:0.0567374
             gamma: 1.00161                     E/m: 1.00161
             beta*gamma: 0.0568289              p/m: 0.0568289
--> boost vector of the B meson
TVector3 A 3D physics vector (x,y,z)=(0.056737,0.000000,0.000000)
(rho,theta,phi)=(0.056737,90.000000,0.000000)
--> now boost the pion to LAB
--> LAB p4 pi:
(x,y,z,t)=(0.009777,0.100000,0.000000,0.172324) (P,eta,phi,E)=(0.100477,-0.000000,1.473334,0.172324)
```

With a boost vector along x, the pion p_y should remain unchanged in the LAB (it is perpendicular to the boost). A p_x component change should instead be manifest.