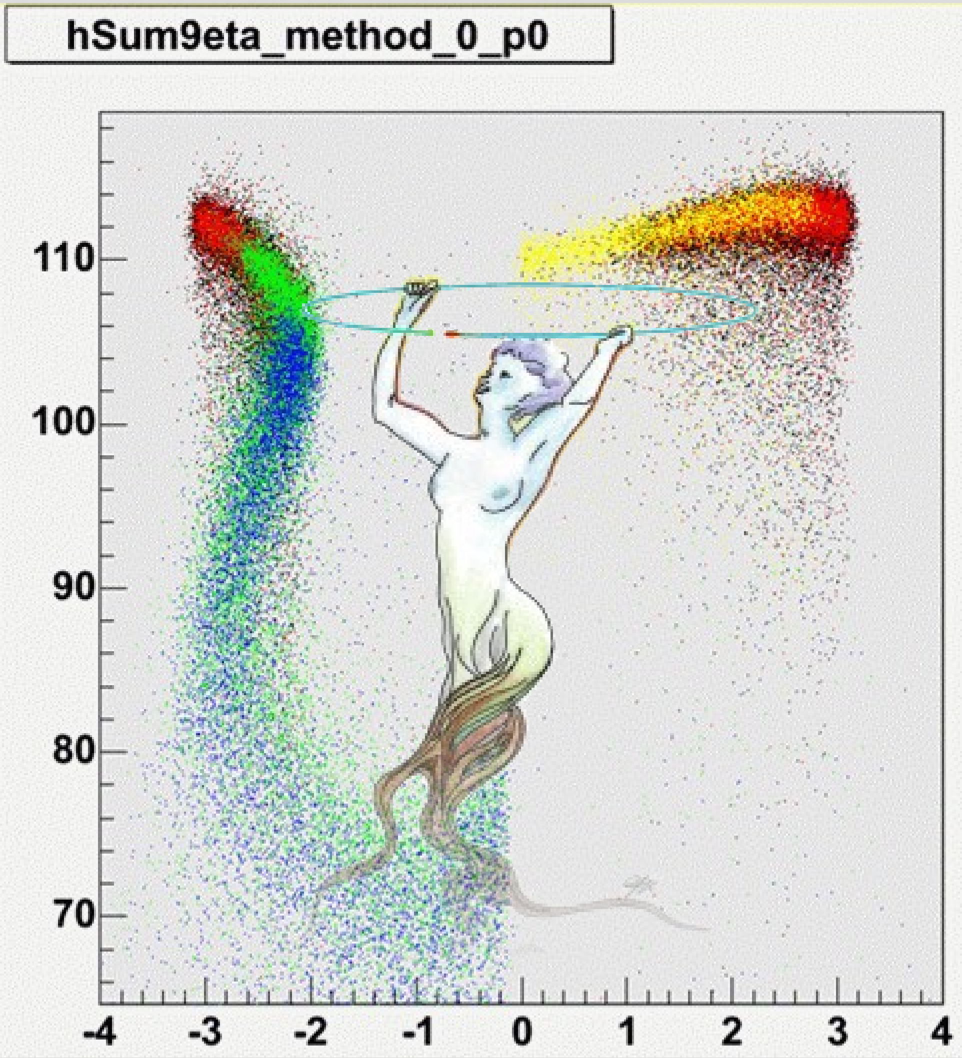
Introduction to ROOT



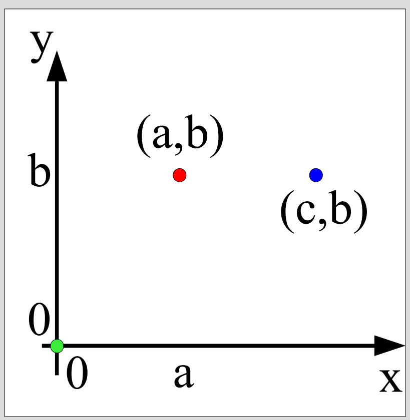
Sami Kama

Outline

- A little C++
- Starting ROOT - CINT
- ROOT Documentation
- Histograms and Graphs
- Trees and makeClass
- Filling in generated class.

A Little C++

01101000011001010110
11000110110001101111=hello




```
float Point_x=a;
float Point_y=b;

struct Point{
  float x;
  float y;
};

Point p1,p2,origin;
p1.x=a;
p1.y=b;
p2.x=c;
p2.y=b;
origin.x=0.;
origin.y=0.;
```

- Computers understand binary.
- Most languages provide support for integers, real and complex values, some has support for strings.
- But computers don't attach any meaning to these values.
- We assign meaning to them.
- And we combine basic types to build complex types.
- We can combine two real numbers to represent a point in 2D space
- But since all 2D points are represented in the same way, we can define a new data type called ***Point***.
- What about lines?

Classes

```
class Point{
public:
  Point(float xValue,float yValue);
  float x;
  float y;
};
class Line{
public:
  Line(Point s,Point e){start=s;end=e;};
  Line(float sx,float sy,float ex,float ey);
  //change end points of the line
  setPoints(Point s,Point e){start=s;end=e;};
//calculate length of the line
  float length();
//calculate the shortest distance to point p.
  float distance(Point p);
private:
  Point start;
  Point end;
};
```

- A class defines a new type of data structure which contains members.
- Members can be variables or functions(methods).
- Classes can be derived from other classes and inherit the methods and variables of base class.
- Classes support access restrictions. They have three different levels of access defined by three keywords.
  - ***private:*** Only class itself can access
  - ***protected:***Class and its derived classes can access.
  - ***public:*** Everybody can access.

Declaring a class: .h file

```
#ifndef __MY_TEST_CLASS_H
#define __MY_TEST_CLASS_H

class MyTestClass{
public:
  //constructor
  MyTestClass();
  //destructor
  ~MyTestClass();
  void myPublicMethod();
  double myPublicVariable;
protected:
  int myProtectedMethod();
  float myProtectedVariable;
private:
  void myPrivateMethod();
  int myPrivateVariable;
};

#endif
```

Multiple inclusion protection

Checks whether **__MY_TEST_CLASS_H** keyword is defined or not. Evaluates **true** if not defined and continues with the next line. Otherwise jumps to **#endif**.

Class declaration. Tells compiler that I am declaring ***MyTestClass***.

Defines **__MY_TEST_CLASS_H**. If same header file (.h file) is included a second time **#ifndef** will evaluate false and skip second declaration

Constructor declaration for initializing the class. A class might have multiple constructors

Class destructor for doing the cleanup and finalization. A class can have only one destructor and it can not take any parameters.

Member methods

Member variables. Typically it is not a good idea to have public member variables.

Finish class declaration. Don't forget **;** to put after **}**.

#endif closes **#ifndef** block

Defining a class: .cpp file

Include class declaration

Constructor definition. Here you initialize variables and prepare object for use

Destructor definition. Here you do final tasks such as closing files, releasing allocated memory, deleting created objects.

MyTestClass:: in front of the methods tell the compiler that the methods defined here belong to MyTestClass.

```
#include "MyTestClass.h"
//include iostream for output
#include <iostream>
//define constructor
MyTestClass::MyTestClass(){
  //here I initialize the class
  myPrivateVariable=-2;
  myProtectedVariable=3;
  myPrivateMethod();
  myPublicMethod();
  std::cout<<"Initialized MyTestClass"
           <<std::endl;
};
//define destructor
MyTestClass::~MyTestClass(){
  std::cout<<"Destroying MyTestClass"
           <<std::endl;
  myPublicMethod();
}
void MyTestClass::myPublicMethod(){
  std::cout<<"private Variable="
           <<myPrivateVariable
           <<"protected Variable="
           <<myProtectedVariable
           <<"public Variable="
           <<myPublicVariable<<std::endl;
}
int MyTestClass::myProtectedMethod(){
  myProtectedVariable+=10.;
  return 0;
}
void MyTestClass::myPrivateMethod(){
  myPublicVariable=2.0;
  myProtectedVariable+=100;
  myPrivateVariable+=1000;
}
```

Include external library definitions

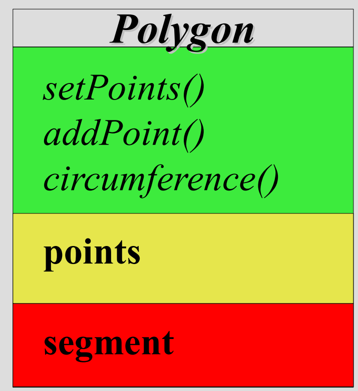
Unlike methods, constructors or destructors do not have return types.

Member method definitions. Each method can be put in different files.

Inheritance

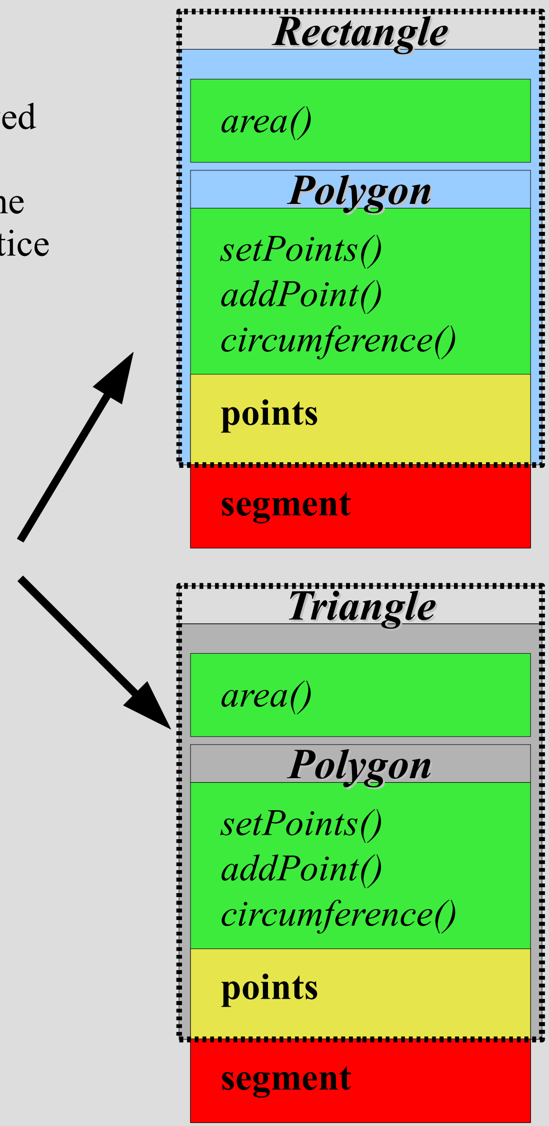
```
class Polygon{
public:
  void setPoints(std::vector<Point> p){points=p;}
//add p to points
  void addPoint(Point p){points.push_back(p);}
//calculate circumference();
  float circumference(){
    float sum=0.;
    for(size_t t=0;t<points.size()-1;t++){
      l.setPoints(points.at(t),points.at(t+1));
      sum+=l.length();
    }
    return sum;
  }
protected:
  std::vector<Point> points;
private:
  Line l;
};
class Rectangle:public Polygon{
public:
  float area(){
    Line side1(points.at(0),points.at(1));
    Line side2(points.at(1),points.at(2));
    return side1.length()*side2.length();
  };
};
class Triangle:public Polygon{
  float area(){
    Line l(points.at(0),points.at(1));
    return l.length()*l.distance(points.at(2))/2.0;
  }x
};
```

Tells that Rectangle is derived from Polygon.
In other words Polygon is the base class of Rectangle. Notice ":" after Rectangle.



Base Class

Derived Classes



Derived classes automatically inherit all public and protected methods and members of their base classes.

STL

- Standard Template Library of C++ composed of several libraries:
  - Strings library defines std::string and string manipulation methods.
  - Containers library defines vector, map, set, list and such containers.
  - Algorithms library defines several useful functions such as sorting, searching and more.
  - I/O library defines basic input/output streams and operations such as cin, cout, fstream, stringstream
  - Numeric library defines complex, valarray and several other mathematical functions.
  - Iterators library defines iterators, pointers to objects in the containers.
  - Utilities library defines various useful functions and classes such as numeric_limits, RTTI, exceptions, assert, date/time, system and environment access etc.

Common Containers

- Vectors are arrays that can change their size dynamically.
  Pros:
  - They are good at random or sequential access
  - Fast at removals from the end.
  - Fast at appending to end if enough space allocated beforehand.

  Cons:
  - Slow at insertions to or removals from the beginning and middle.
  - Slow to search.
- Maps are associative containers that have key-value pairs.
  Pros:
  - Fast to search, they are good lookup tables.

  Cons:
  - Slow at insertions since they are sorted.

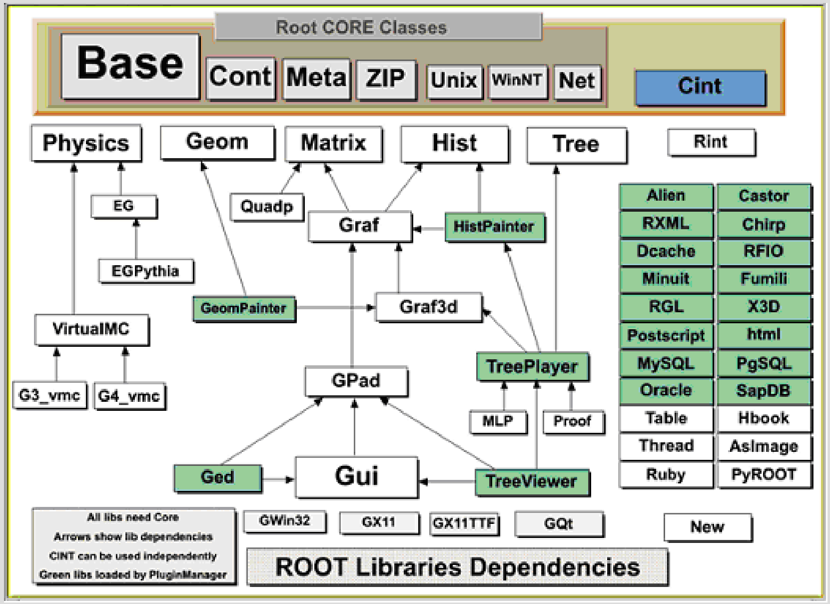
Other notable classes

- <string> defines std::string class.Strings are human readable sequence of characters. They are meant to be used for human interaction. Try to avoid using them unless they are for human interactions such as preparing something for humans to read or getting input from humans.
- <iostream> defines std::cin, std::cout, and std::cerr streams. They provide access to stdin, stdout, stderr.
- <fstream> defines std::fstream, std::ifstream and std::ofstream. fstream can read and write files while ifstream can only read and ofstream can only write.
- <sstream> defines stringstream, istringstream and ostringstream. istringstream class is used for reading from a string buffer while ostringstream is used for writing into it. stringstream can both read from and write to a string.

Some references

- Bjarne Stroustrup, *The C++ Programming Language*
- http://www.cppreference.com
- http://www.cplusplus.com
- Google! (or your favorite search engine)

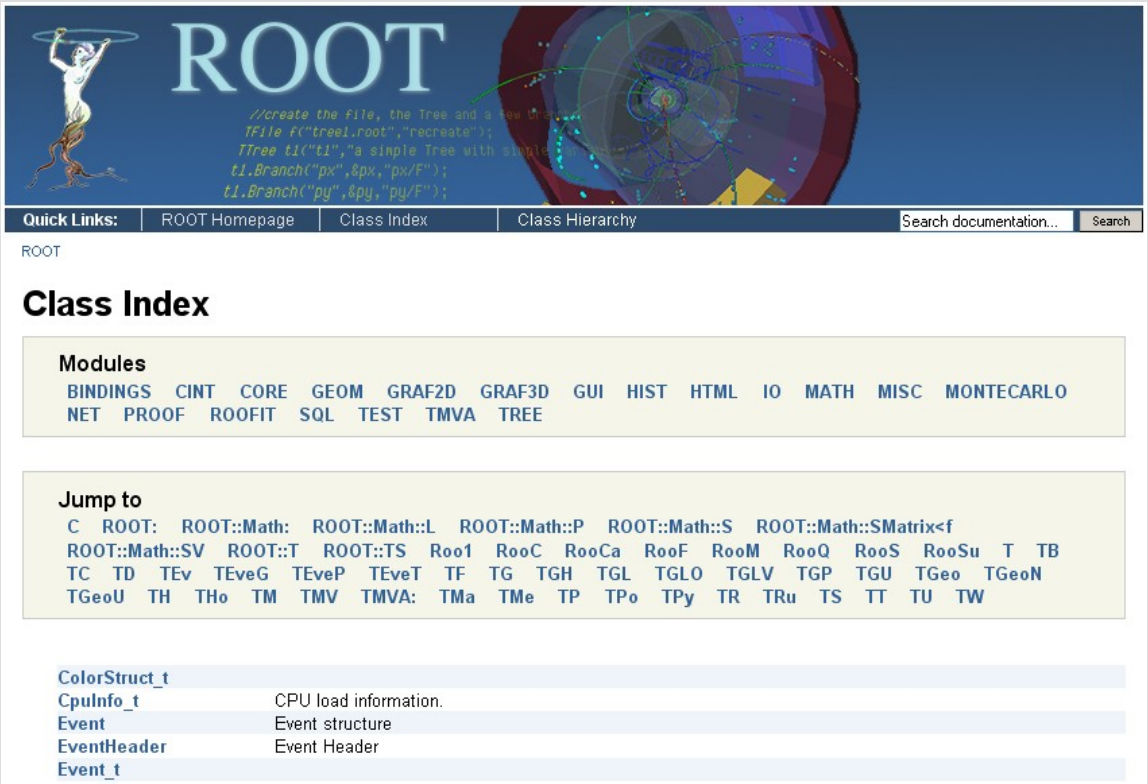
ROOT



- *"ROOT is an object-oriented framework aimed at solving the data analysis challenges of high-energy physics. There are two key words in this definition, object oriented and framework"*
- A framework is an infrastructure that defines tools and services. You need to use these tools and services to build your applications (or analysis).
- ROOT provides an extensive list of classes for graphics, statistics, I/O, and mathematics.
- These classes separated in several libraries and rather well documented in http://root.cern.ch/root/html528/ClassIndex.html

Reference Guide

This page will probably be your most frequently visited page about ROOT



TGuiBldHintsEditor	layout hints editor
TGuiBldNameFrame	frame name editor
TGuiBuilder	ABC for gui builder
TGuiFactory	Abstract factory for GUI components
TH1	1-Dim histogram base class
TH1C	1-Dim histograms (one char per channel)
TH1D	1-Dim histograms (one double per channel)
TH1Editor	TH1 editor
TH1F	1-Dim histograms (one float per channel)
TH1I	1-Dim histograms (one 32 bits integer per channel)
TH1K	1-Dim Nearest Kth neighbour method
TH1S	1-Dim histograms (one short per channel)
TH2	2-Dim histogram base class
TH2C	2-Dim histograms (one char per channel)
TH2D	2-Dim histograms (one double per channel)
TH2Editor	TH2 editor
TH2F	2-Dim histograms (one float per channel)
TH2GL	GL renderer for TH2.
TH2I	2-Dim histograms (one 32 bits integer per channel)
TH2Poly	2-Dim histogram with polygon bins
TH2PolyBin	2-Dim polygon bins
TH2S	2-Dim histograms (one short per channel)
TH3	3-Dim histogram base class
TH3C	3-Dim histograms (one char per channel)
TH3D	3-Dim histograms (one double per channel)
TH3F	3-Dim histograms (one float per channel)
TH3GL	GL renderer class for TH3.
TH3I	3-Dim histograms (one 32 bits integer per channel)
TH3S	3-Dim histograms (one short per channel)
THLimitsFinder	Class to find best axis limits
THStack	A collection of histograms
THYPE	HYPE shape
THaarMatrixT<double>	Template of Haar Matrix class
THaarMatrixT<float>	Template of Haar Matrix class
THashList	Doubly linked list with hashtable for lookup

Proper use of reference guide is essential!

http://root.cern.ch/root/html528/ClassIndex.html

Make sure that you are using correct reference version!

Function Members (Methods)

public:

```
                TH2F ()
                TH2F (const TMatrixFBase& m)
                TH2F (const TH2F& h2f)
                TH2F (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins, Int_t nbinsy, const
                TH2F (const char* name, const char* title, Int_t nbinsx, const Float_t* xbins, Int_t nbinsy, const Fl
                TH2F (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins, Int_t nbinsy, Double
                     yup)
                TH2F (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup, Int_t nbinsy, const Double_t*
                     ybins)
                TH2F (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup, Int_t nbinsy, Double_t ylow,
                     Double_t yup)
        virtual ~TH2F ()
   virtual void AddBinContent (Int_t bin)
   virtual void AddBinContent (Int_t bin, Double_t w)
 static TClass* Class ()
   virtual void Copy (TObject& hnew) const
  virtual TH1* DrawCopy (Option_t* option = "") const
virtual Double_t GetBinContent (Int_t bin) const
virtual Double_t GetBinContent (Int_t binx, Int_t biny) const
virtual Double_t GetBinContent (Int_t binx, Int_t biny, Int_t) const
virtual TClass* IsA () const
          TH2F& operator= (const TH2F& h1)
   virtual void Reset (Option_t* option = "")
   virtual void SetBinContent (Int_t bin, Double_t content)
   virtual void SetBinContent (Int_t binx, Int_t biny, Double_t content)
   virtual void SetBinContent (Int_t binx, Int_t biny, Int_t, Double_t content)
   virtual void SetBinsLength (Int_t n = -1)
   virtual void ShowMembers (TMemberInspector& insp)
   virtual void Streamer (TBuffer& b)
           void StreamerNVirtual (TBuffer& b)
```

class TH2F –

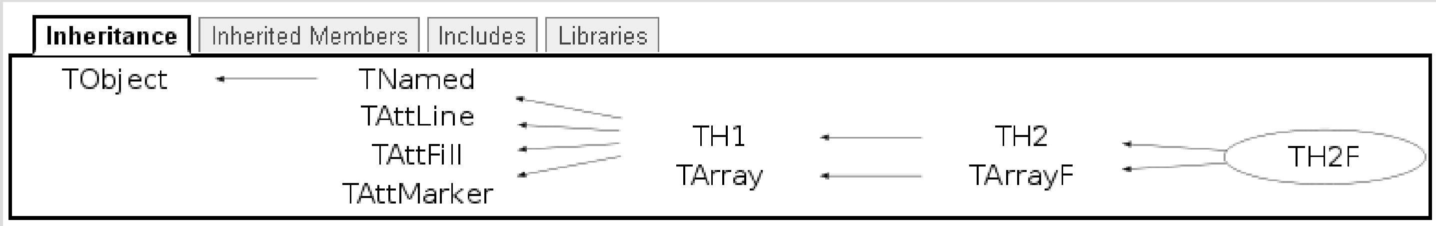
library: libHist
#include "TH2.h"

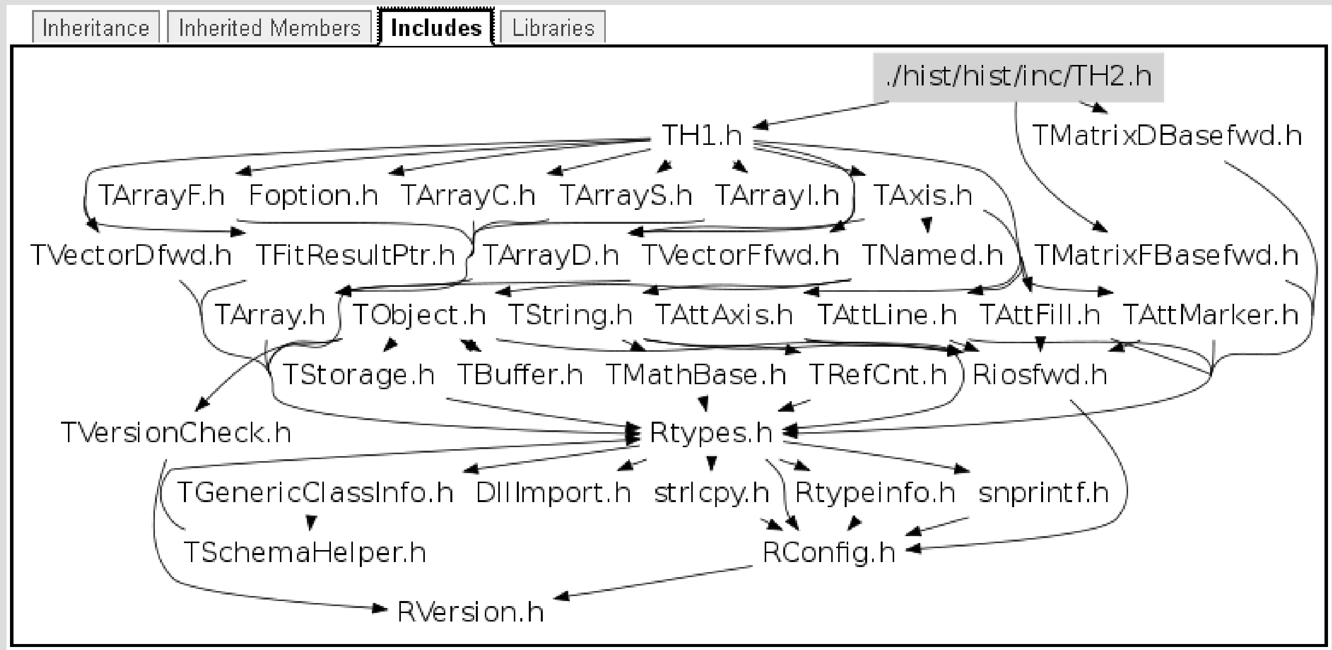
Display options:
☐ Show inherited
☐ Show non-public

[Top] | [? Help]

Rather small list of methods?!
You must enable "Show inherited"
to see full list of available methods!

Inheritance tree for TH2F





Header files that are included automatically when you include TH2.h in your code

This list is less than half of available methods!

ROOT » HIST » HIST » TH1

class TH1: public TNamed, public TAttLine, public TAttFill, public TAttMarker

TH1
library: libHist
#include "TH1.h"

Display options:
☑ Show inherited
☐ Show non-public

[↑ Top] | [? Help]

The Histogram classes

ROOT supports the following histogram types:

- 1-D histograms:
  - TH1C : histograms with one byte per channel. Maximum bin content = 127
  - TH1S : histograms with one short per channel. Maximum bin content = 32767
  - TH1I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH1F : histograms with one float per channel. Maximum precision 7 digits
  - TH1D : histograms with one double per channel. Maximum precision 14 digits
- 2-D histograms:
  - TH2C : histograms with one byte per channel. Maximum bin content = 127
  - TH2S : histograms with one short per channel. Maximum bin content = 32767
  - TH2I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH2F : histograms with one float per channel. Maximum precision 7 digits
  - TH2D : histograms with one double per channel. Maximum precision 14 digits
- 3-D histograms:
  - TH3C : histograms with one byte per channel. Maximum bin content = 127
  - TH3S : histograms with one short per channel. Maximum bin content = 32767
  - TH3I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH3F : histograms with one float per channel. Maximum precision 7 digits
  - TH3D : histograms with one double per channel. Maximum precision 14 digits
- Profile histograms: See classes TProfile, TProfile2D and TProfile3D. Profile histograms are used to display the mean value of Y and its RMS for each bin in X. Profile histograms are in many cases an elegant replacement of two-dimensional histograms : the inter-relation of two measured quantities X and Y can always be visualized by a two-dimensional histogram or scatter-plot; If Y is an unknown (but single-valued) approximate function of X, this function is displayed by a profile histogram with much better precision than by a scatter-plot.

Class documentation, same as in the users guide. You should read at least ***TH1***, ***TProfile****, ***TTree***, and ***TCanvas***.

Histograms

- There are various histogram types in ROOT. They are all derived from TH1 and have names in the form TH[123][CSIFD] or TProfile([23]D)?.
- Typical constructor for histograms have the form
  Thxx(char* objName,char* histTitle,\(int nbins, lowLim, upLim\){1,3}) or
  Thxx(char* objName,char* histTitle,\(int nbins, arrPointer\){1,3})
- They support both constant and variable bin sizes as well as labelled bins. They have nbin+2 bins for each dimension, n=0 being underflow and n=nbin+1 being overflow.
- All histograms are filled with ***Fill(binNo,weight)*** function. Higher dimensional histograms will map the x,y or x,y,z values to correct bin. For example, ***TH3D.Fill(x,y,z,w)*** method will find the bin which contains the point ***(x,y,z)*** and increment its value by ***w.***
- ***Draw()*** function will draw the histogram on a canvas. ***Draw("same")*** or ***Draw("sames")*** are the most common draw options. See TH1::Draw() for more details.

I like regular expressions :)

Graphs

- ROOT also has a large set of graphs. TGraph, TGraph2D, TGraphErrors, TGraphAsymErrors are some of them.
- All graphs contain a set of points but some of them also contain errors on points.
- They can be initialized only with the number of points and then values of individual points can be altered with ***SetPoint()*** and ***SetPointError()*** methods.
- They can also be initialized by passing pointers to the arrays containing values and errors of the point.
- Unlike histograms graphs don't draw axis to the current canvas by default. If current canvas has an axis, graph uses it. Otherwise you must use ***Draw("AL").***
- See TGraphPainter class for more information about drawing graphs.

ROOT CINT

```
feynman@istapp2011:~$ root -l
root [0] ?

Note1: Cint is not aimed to be a 100% ANSI/ISO compliant C/C++ language
 processor. It rather is a portable script language environment which
 is close enough to the standard C++.

Note2: Regulary check either of /tmp /usr/tmp /temp /windows/temp directory
 and remove temp-files which are accidentally left by cint.

Note3: Cint reads source file on-the-fly from the file system. Do not change
 the active source during cint run. Use -C option or C1 command otherwise.

Note4: In source code trace mode, cint sometimes displays extra-characters.
 This is harmless. Please ignore.

CINT/ROOT C/C++ interpreter interface.
All commands must be preceded by a . (dot), except
for the evaluation statement { } and the ?.
===========================================================================
             > [file]  : output redirection to [file]
             2> [file] : error redirection to [file]
             >& [file] : output&error redirection to [file]
Help:        ?         : help
-- Press return for more -- (input [number] of lines, Cont,Step,More)
             help      : help
             /[keyword] : search keyword in help information
Shell:       ![shell]  : execute shell command
Source:      v <[line]>: view source code <around [line]>
             V [stack] : view source code in function call stack
             t         : show function call stack
             f [file]  : select file to debug
             T         : turn on/off trace mode for all source
             J [stat]  : Set warning level [0-5]
             A [1|0]   : allowing automatic variable on/off
```

- ROOT can be started with command ***root -l*** (-l is for skipping splash screen). If root is not installed on your system, see http://root.cern.ch/drupal/content/downloading-root and http://root.cern.ch/drupal/content/installing-root-source
- ROOT has a built-in C interpreter called CINT.
- It can evaluate C/C++ expressions and has an auto-complete feature similar to bash.
- You can get help by typing ***?*** in CINT prompt
- All commands listed in help must be preceded with a **.**(dot)

Interacting with CINT

Instantiate a TFile object ***f***, which creates/overwrites “MyTestRootFile”

```
root [1] TFile f("MyTestRootFile","RECREATE")
root [2] TH1F myHist(
TH1F TH1F()
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)
TH1F TH1F(const TVectorF& v)
TH1F TH1F(const TH1F& h1f)
root [2] TH1F myHist("myHist","Test Histogram from Command-Line",1000,-2.0,2.0)
```

I forgot TH1F constructor here. Pressing tab shows me possible constructors.

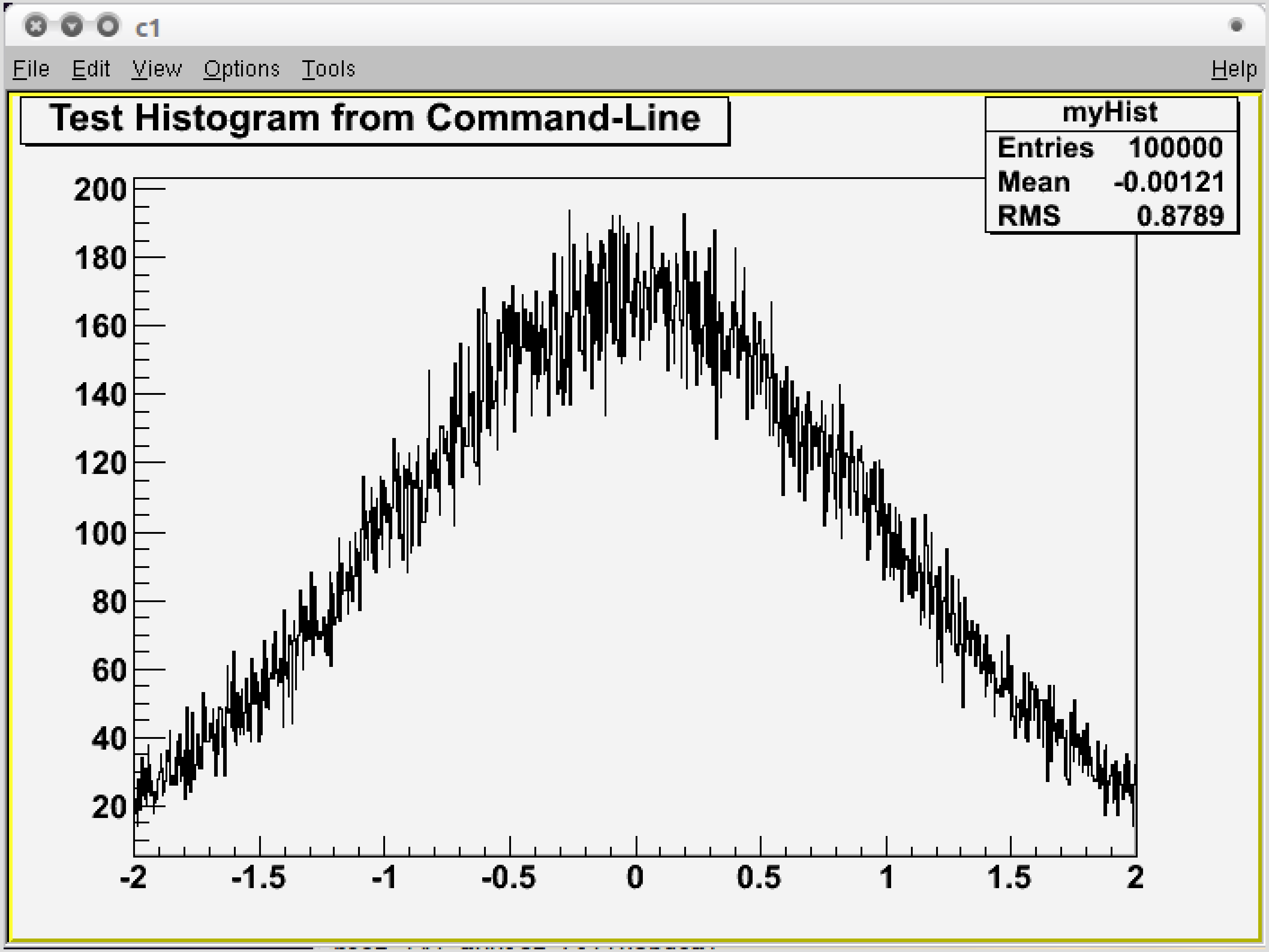
I instantiate a TH1F object myHist, a 1000 bin 1D histogram with limits -2.0 to 2.0

I am filling my histogram with 100000 random entries, distributed according to ***“gaus”*** function

```
root [5] myHist.FillRandom("gaus",100000)
root [6] myHist.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

Drawing my histogram

Since I didn't have any TCanvas instances before, ROOT created one named c1 for me.



Output of the commands that I typed-in to CINT

Executing a macro

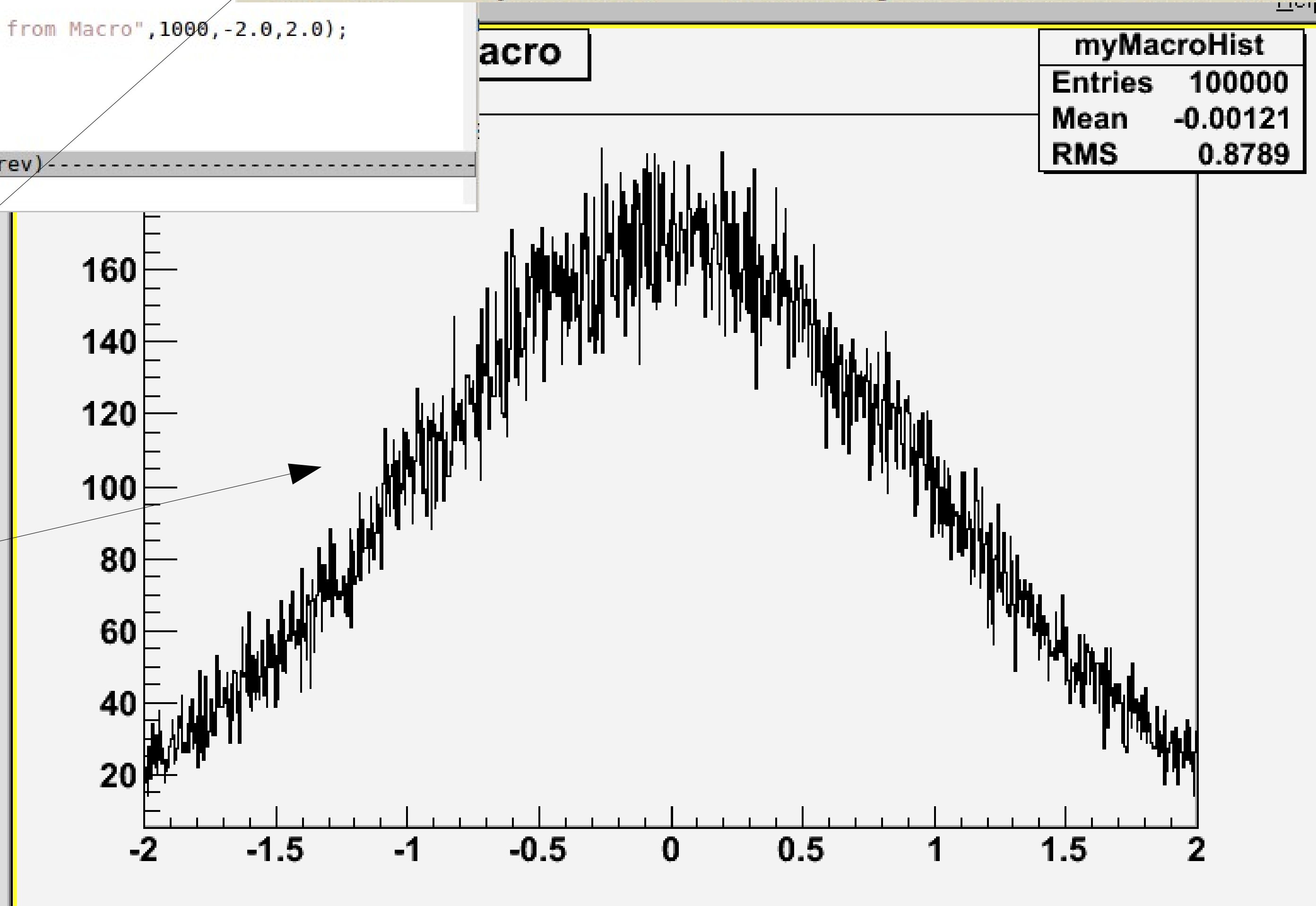
emacs@istapp2011

File Edit Options Buffers Tools C++ Help

```
{
  // I can write commands that I type in a .C file
  // Then I can execute it with .x "myTestMacro1.C"
  TFile f("MyTestRootFile","RECREATE");
  TH1F myHist("myMacroHist","Test Histogram from Macro",1000,-2.0,2.0);
  myHist.FillRandom("gaus",100000);
  myHist.Draw();
}
```

-U:--- myTestMacro1.C All L9 (C++/l Abbrev)----------------------------------

```
root [0] .x myTestMacro1.C
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [1] .ls
TFile**         MyTestRootFile
 TFile*         MyTestRootFile
  OBJ: TH1F     myMacroHist     Test Histogram from Macro : 0 at: 0xa160768
```



I write everything in myTestMacro1.C file

Then I execute macro.

I get exactly the same histogram

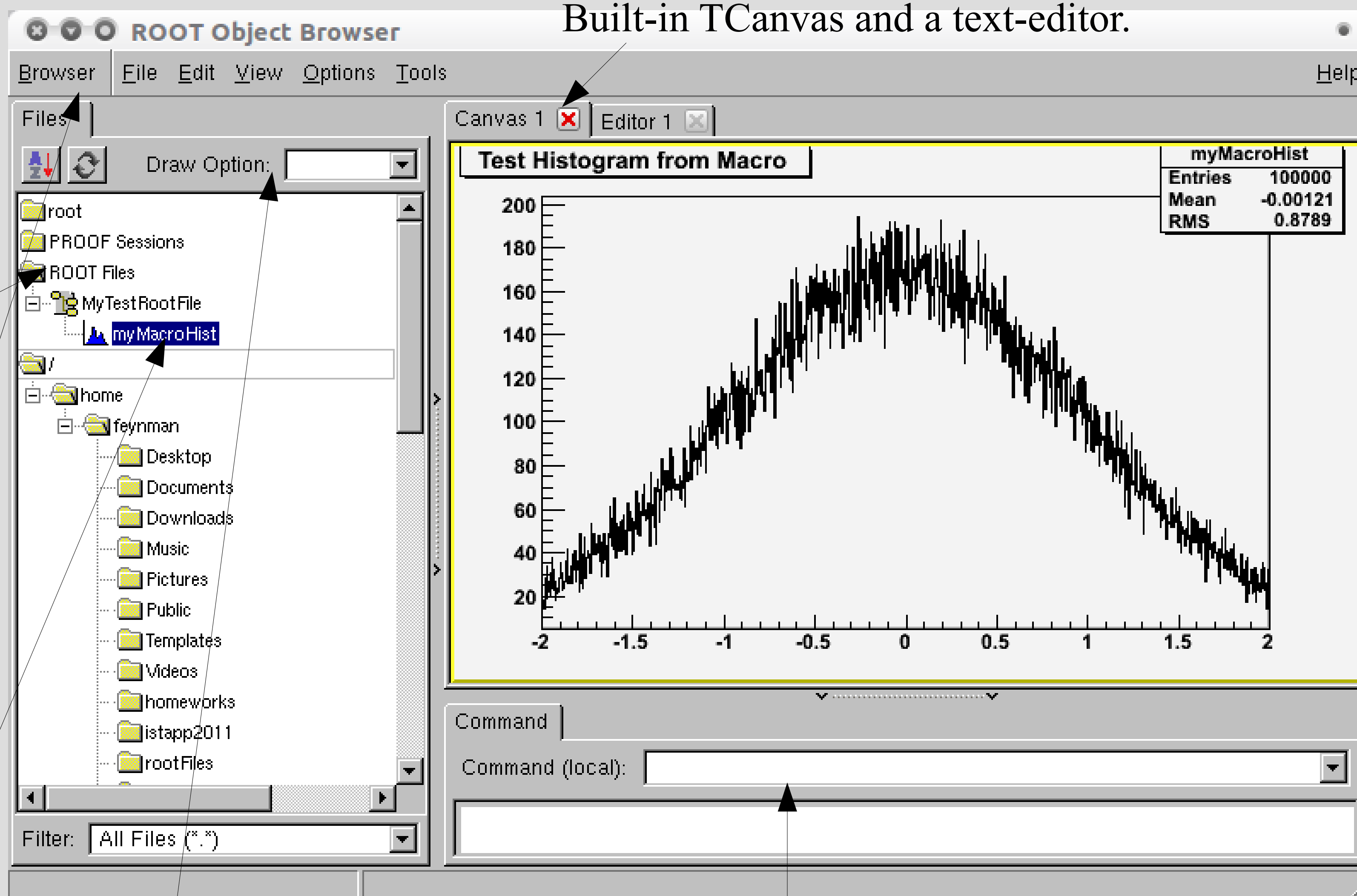
You can also have named macros which you can execute many times by typing its name in CINT.

TBrowser

```
root [1] TBrowser b
root [2]
```

Starts new ROOT Browser

Built-in TCanvas and a text-editor.



Point-and-click access to open root files and local files.

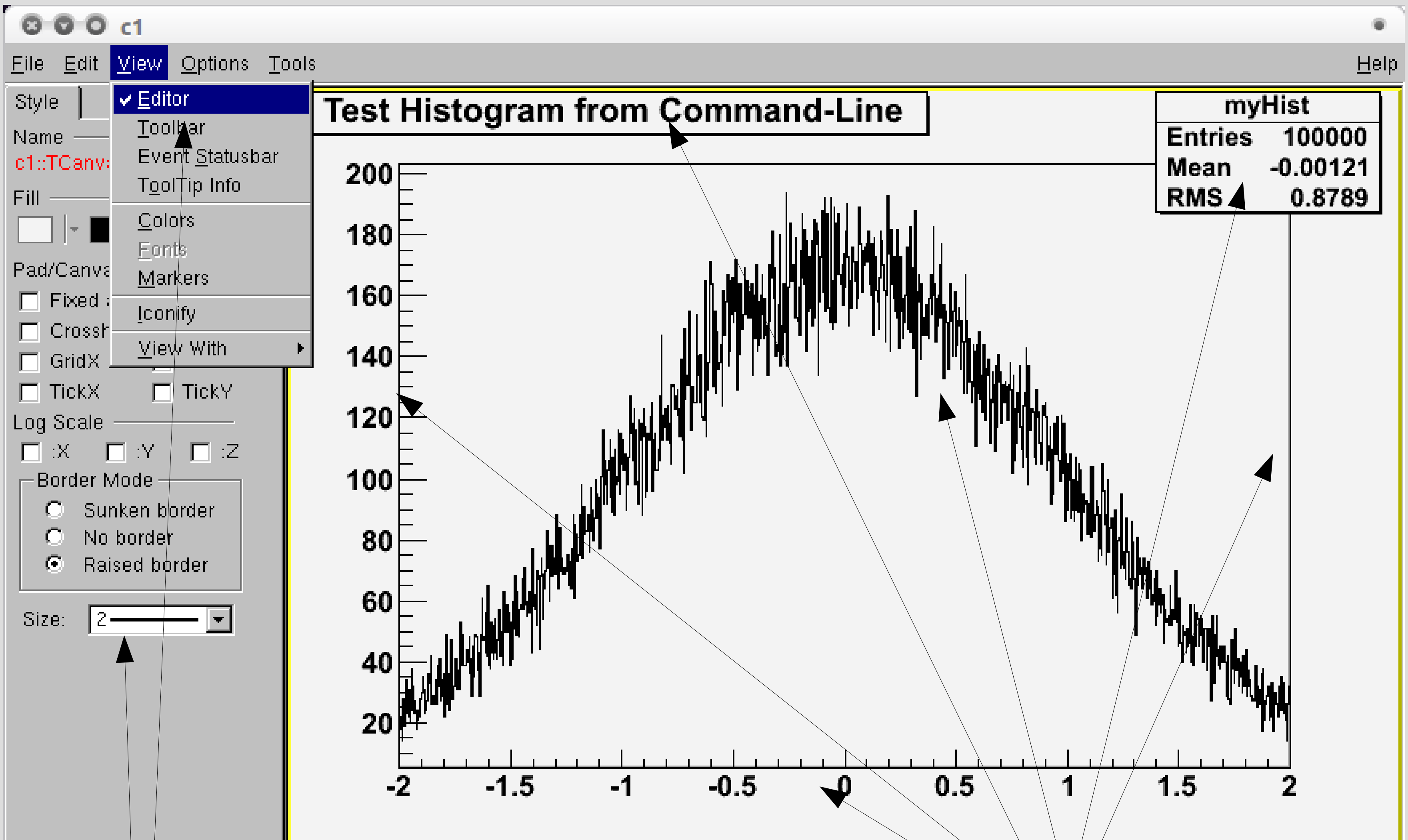
File menu allows you to save current canvas in different formats such as .ps, .pdf, .png, .eps even as a root macro.

Double-clicking on a histogram will draw it on the built-in canvas

You can set options for Draw() method of the selected object.

You can type-in your commands like in the shell.

TCanvas



TCanvas not only displays but also can interact and edit histograms and other objects in it. Enable "Editor" from the "View" menu.

All these are editable. Editor automatically changes its layout to show most common options. Sometimes you can learn how to modify these parameters in your macros if you save your canvas as a root macro. (.C file)

Trees

Trees provide an easy way to store large amounts of data into files. They can store basic data types, such as floats, doubles, booleans, integers etc, or complex data types such as classes. At every call to Ttree::Fill(), values of the variables are stored into file as an ***Entry***. When reading the tree, TTree::GetEntry() method sets the values of the variables to the values at the time of call to TTree::Fill() for the respective ***Entry***.

```
void genTree() {
   const Int_t kMaxTrack = 500;
   Int_t ntrack;
   Int_t stat[kMaxTrack];
   Int_t sign[kMaxTrack];
   Float_t px[kMaxTrack];
   Float_t py[kMaxTrack];
   Float_t pz[kMaxTrack];
   Float_t pt[kMaxTrack];
   Float_t zv[kMaxTrack];
   Float_t chi2[kMaxTrack];
   TFile f("testtree.root","recreate");
   TTree *t3 = new TTree("test","Reconst ntuple");
   t3->Branch("ntrack",&ntrack,"ntrack/I");
   t3->Branch("stat",stat,"stat[ntrack]/I");
   t3->Branch("sign",sign,"sign[ntrack]/I");
   t3->Branch("px",px,"px[ntrack]/F");
   t3->Branch("py",py,"py[ntrack]/F");
   t3->Branch("pz",pz,"pz[ntrack]/F");
   t3->Branch("zv",zv,"zv[ntrack]/F");
   t3->Branch("chi2",chi2,"chi2[ntrack]/F");
   for (Int_t i=0;i<1000000;i++) {
      Int_t nt = gRandom->Rndm()*(kMaxTrack-1);
      for (Int_t n=0;n<nt;n++) {
         stat[n] = n%3;
         sign[n] = i%2;
         px[n]   = gRandom->Gaus(0,1);
         py[n]   = gRandom->Gaus(0,2);
         pz[n]   = gRandom->Gaus(10,5);
         zv[n]   = gRandom->Gaus(100,2);
         chi2[n] = gRandom->Gaus(0,.01);
         pt[n]   = TMath::Sqrt(px[n]*px[n] + py[n]*py[n]);
      }
      t3->Fill();
   }
   t3->Print();
   f.cd();
   t3->Write();
}
```

- Variable declarations: Setup our containers for the information. We will store the information in these arrays into tree on every call to Fill() method
- `TFile f(...)`: Create a file
- `TTree *t3 = new TTree(...)`: Create a tree named “test”
- `t3->Branch(...)` lines: Adding our arrays to tree. All arrays contain “[ntrack]” to tell root that we will save *ntrack* items from the array on each Fill() call.Notice “/I” and “/F” at the end. They tell ROOT the type of the arrays that we add.
- Loop: Fill our arrays with random variables. Normally these should be filled with meaningful information from event.
- `t3->Fill();`: Copy ***ntrack*** items from the arrays into file as a new ***Entry***.
- `f.cd();`: cd to file we opened to make sure that our tree is written in it
- `t3->Write();`: Write tree to file.

MakeClass()

Filling MakeClass() output

Flat ntuples

HANDS-ON
(see hands-on slides)