Introduction to Root program:L1

Presented by

DR. MOHAMMED ATTIA MAHMOUD

-PhD, Fayoum University, Egypt and Antwerp University, Belgium.

-Researcher in ENHEP, ASRT, Fayoum Uni, and BUE.

-FSQ Gen-Contact, CMS experiment, CERN, Geneva, Switzerland.

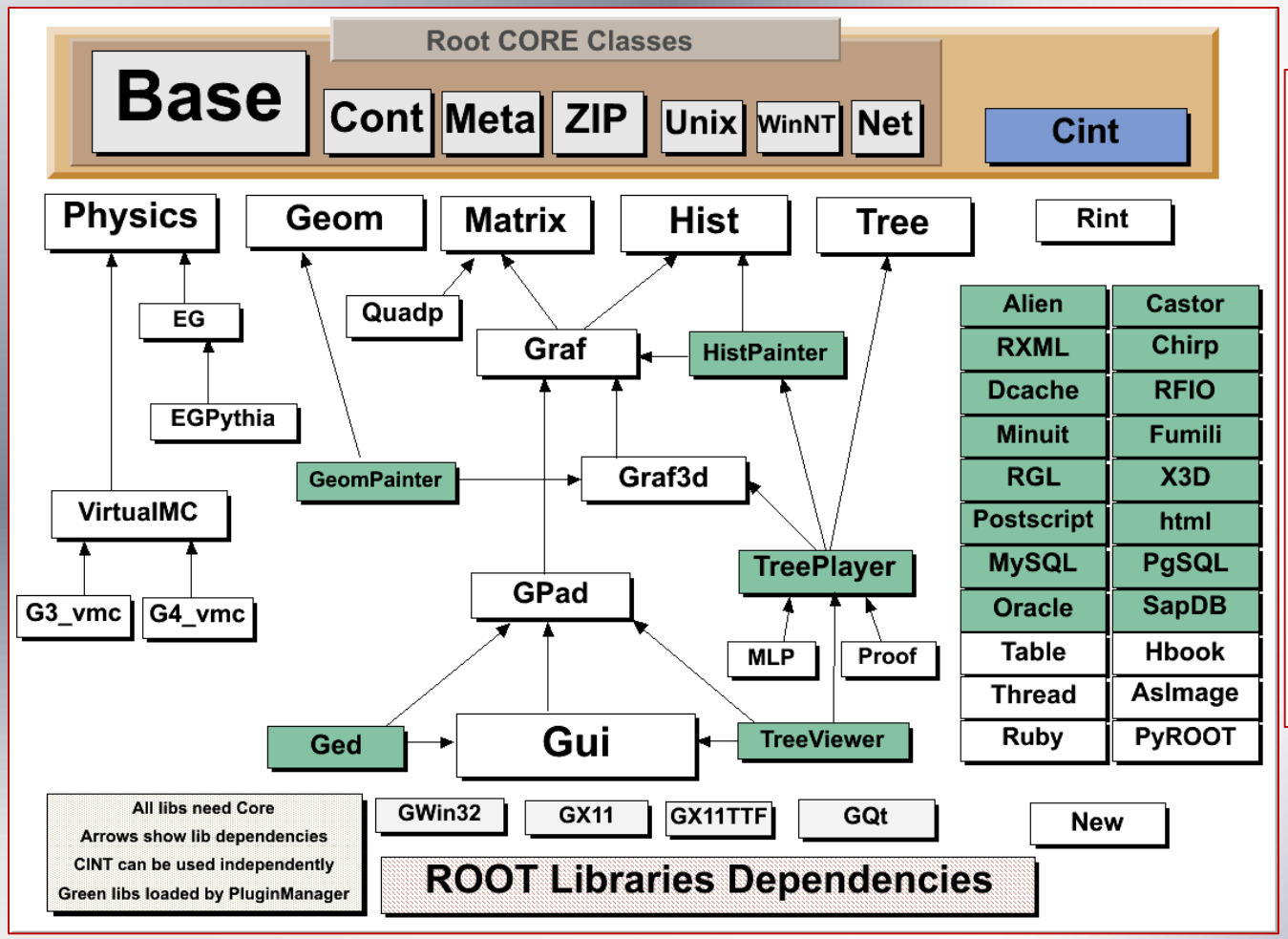
Outline

- ROOT in a Nutshell
- The ROOT Libraries
- ROOT: An Open Source Project
- ROOT: a Framework and a Library
- ROOT Application Domains
- CINT Interpreter
- Examples

ROOT in a Nutshell

- The ROOT system is an Object Oriented framework for large scale data handling applications. It is written in C++.
- Provides, among others,
  - ✓ an efficient data storage and access system designed to support structured data sets (PetaBytes)
  - ✓ a query system to extract data from these data sets
  - ✓ a C++ interpreter
  - ✓ advanced statistical analysis algorithms (multi dimensional histogramming, fitting, minimization and cluster finding)
  - ✓ scientific visualization tools with 2D and 3D graphics
  - an advanced Graphical User Interface
- The user interacts with ROOT via a graphical user interface, the command line or scripts
- The command and scripting language is C++, thanks to the embedded CINT C++ interpreter, and large scripts can be compiled and dynamically loaded.
- A Python shell is also provided.

The ROOT Libraries



- Over 1500 classes
- 1,550,000 lines of code
- CORE (8 Mbytes)
- CINT (2 Mbytes)
- Green libraries linked on demand via plug-in manager (only a subset shown)
- 100 shared libs

ROOT: An Open Source Project

- The project was started in 1995.
- The project is developed as a collaboration between:
  - **Full time developers:**
    - **11 people full time at CERN (PH/SFT)**
    - **+4 developers at Fermilab/USA, Protvino , JINR/Dubna (Russia)**
  - **Large number of part-time contributors (155 in CREDITS file)**
  - A long list of users giving feedback, comments, bug fixes and many small contributions
    - 2400 registered to RootForum
    - 10,000 posts per year
- An Open Source Project, source available under the LGPL license

ROOT: a Framework and a Library

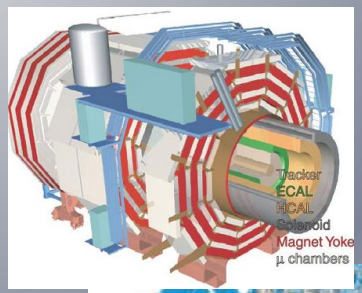
- User classes
  - User can define new classes interactively
  - Either using calling API or sub-classing API
  - These classes can inherit from ROOT classes
- Dynamic linking
  - Interpreted code can call compiled code
  - Compiled code can call interpreted code
  - Macros can be dynamically compiled & linked

This is the normal operation mode

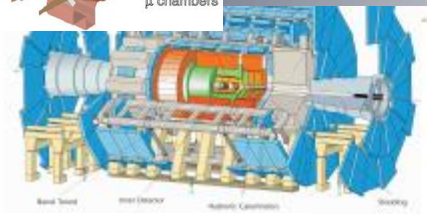
Interesting feature for GUIs & event displays

Script Compiler
root > .x file.C++

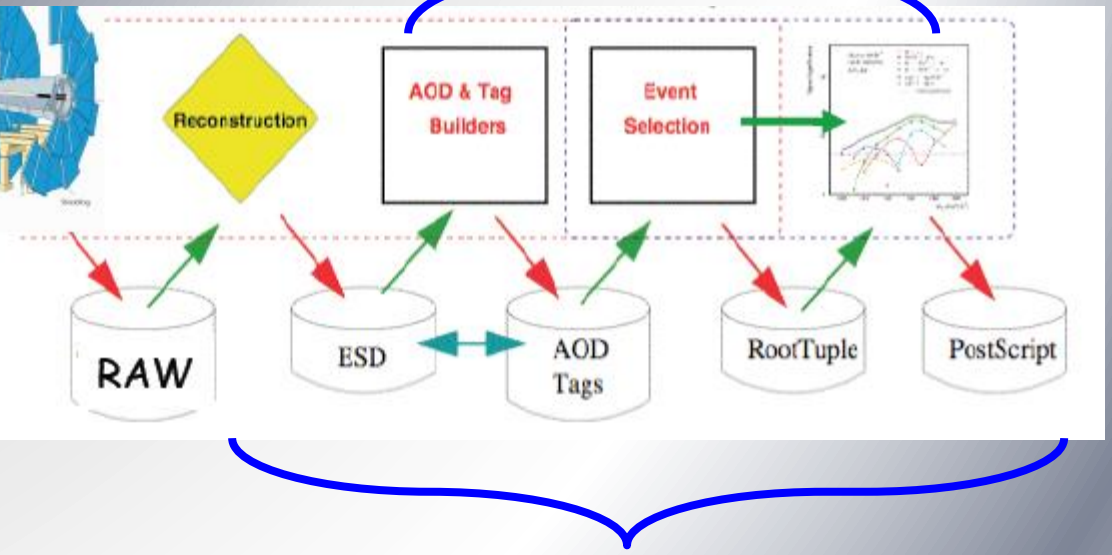
ROOT Application Domains



Data Analysis & Visualization

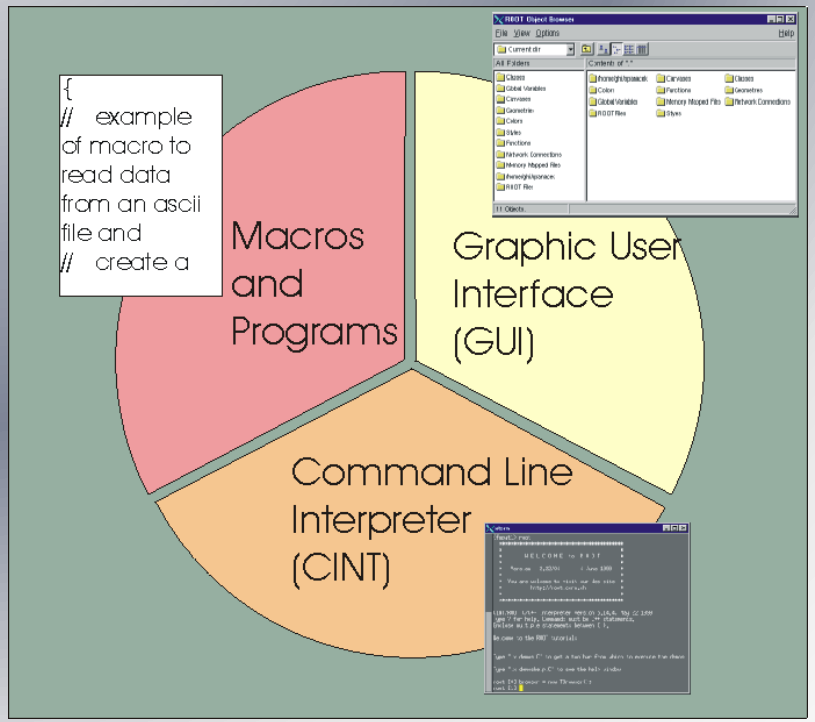


General Framework



Data Storage: Local, Network

Three User Interfaces



- GUI
  windows, buttons, menus
- Command line
  CINT (C++ interpreter)
- Macros, applications, libraries (C++ compiler and interpreter)

CINT Interpreter

CINT in ROOT

- CINT is used in ROOT:
  - As command line interpreter
  - As script interpreter
  - To generate class dictionaries
  - To generate function/method calling stubs
  - Signals/Slots with the GUI
- The command line, script and programming language become the same
- Large scripts can be compiled for optimal performance

Running Code

To run function mycode() in file mycode.C:

```
root [0] .x mycode.C
```

Equivalent: load file and run function:

```
root [1] .L mycode.C
root [2] mycode()
```

All of CINT's commands (help):

```
root [3] .h
```

Running Code

To run function mycode() in file mycode.C:

```
root [0] .x mycode.C
```

Equivalent: load file and run function:

```
root [1] .L mycode.C
root [2] mycode()
```

All of CINT's commands (help):

```
root [3] .h
```

Histograms

Making your first histogram:

- Histograms can be 1-d, 2-d and 3-d
- Declare a histogram to be filled with floating point numbers:

```
TH1F *histName = new TH1F("histName", "histTitle", num_bins,x_low,x_high)
```

- 2-d and 3-d histograms can be booked similarly…

```
TH1F *my_hist = new TH1F("my_hist", "My First Histogram", 100, 2, 200)

TH2F *myhist = new TH2F("myhist", "My Hist", 100, 2, 200, 200, 0,500)
```

Drawing Histograms

- **To draw:**

  my_hist->Draw();

- **To fill a histogram:**

  my_hist->Fill(50);

  my_hist->Fill(100, 3); // the number 100 has weight=3

- **Update the histogram:**

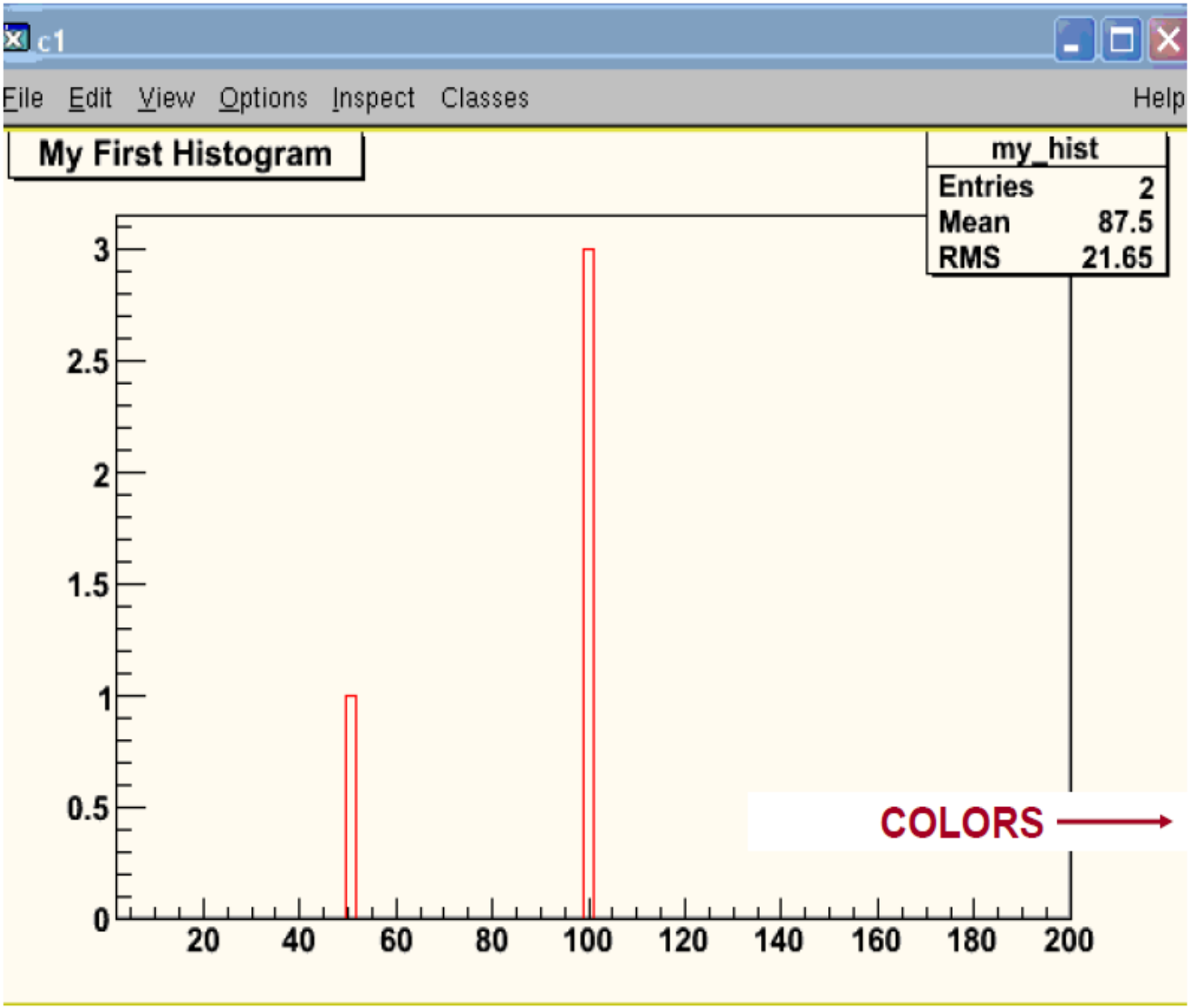
  my_hist->Draw

```
root [0]
root [0] TH1F *my_hist = new TH1F("my_hist", "My First Histogram", 100, 2, 200);
root [1] my_hist->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [2] my_hist->Fill(50);
root [3] my_hist->Fill(100,3);
root [4] my_hist->Draw();
root [5] my_hist->SetLineColor(2);
root [6] my_hist->Draw();
root [7]
```

- **Change line color:**

  my_hist->SetLineColor(2); //red

  or my_hist->SetLineColor(kRed); – my_hist->Draw();



1	Black
2	Red
3	Light green
4	Blue
5	Yellow
6	Magenta
7	Cyan
8	Green

Legends

We want to move towards being able to compare two or more histograms by plotting them on the same axes. In order to do this we need two more skills - one is to be able to plot histograms with different colours, and the other is some sort of information on which histo is which. The latter is done with a "legend". Use the following code to add a legend to your histogram:

```
leg = new TLegend(0.6,0.7,0.89,0.89); //coordinates are fractions //of pad dimensions
 leg->AddEntry(hist_1,"First histo","l"); // "l" means line // use "f" for a box
leg->Draw(); // oops
```

we forgot the header (or "title") for the legend

```
leg->SetHeader("The Legend Title");
leg->Draw();
leg->SetTextSize(0.04); // set size of text
leg->SetFillColor(0); // Have a white background
leg->AddEntry(hist_1, "text 1", "p"); // p shows points, // other options exist // (Check documentation)

leg->Draw();
```

Comparing Histograms

To illustrate how to plot two histograms on the same canvas, we will need to set up another histogram:

```
TH1F *hist_2 = new TH1F("hist_2", "Another histo", 100, 2, 300);
```

Let's fill a few bins:

```
hist_2->Fill(20,10);
hist_2->Fill(50,4);
hist_2->Fill(3);
hist_2->Draw();
hist_1->Draw("same");
hist_1->SetLineColor(8); // green
 hist_2->SetLineColor(4); // blue
hist_2.Draw(); //draw hist_2 first as it has a larger range
hist_1.Draw("same");
leg_hist = new TLegend(0.5,0.6,0.79,0.79);
leg_hist->SetHeader("Some histograms");
leg_hist->AddEntry(hist_1,"First histo","l");
 leg_hist->AddEntry(hist_2,"Second histo","l");
 leg_hist->Draw();
```

Copying Histograms

You can make an identical copy of a histogram by cloning,

```
TH1F *hist_new=(TH1F*)hist_1->Clone(); hist_new->SetName("hist_new");.
```

More Drawing Options

Here are some Draw options that you might like to experiment with for a 2-dimensional histogram:

```
h2->Draw("text");
h2->Draw("col"),
h2->Draw("colz");
h2->Draw("box");
h2->Draw("surf");
```

Including Error Bars in Histograms

You can plot errors on histograms (perhaps more appropriate to plot them on first to histograms!) by entering

```
hist_2.Draw("esame");
```

By default, errors are sqrt(entries).

Saving Histograms

Saving Histograms as Image Files

Now save your masterpiece to a file (this assumes that your histogram is printed on canvas "c1"):

```
c1->SaveAs("myimage.eps");
c1->SaveAs("myimage.ps");
c1->SaveAs("myimage.gif");
```

Saving Source Code for Histograms

```
c1->SaveAs("myimage.C");
```

you can recreate the histogram in exactly the form that you saved it in a brand new ROOT session by entering:

```
.x myimage.C
```


Thanks!