ROOT5

L. Peter Alonzi III

University of Virginia

November 10, 2010
Philosophy

ROOT was written by people who haven’t done physics for 20 years for people who won’t do physics for 20 years.

Back in the day it was Fortran and Geant3 and Paw. Now it’s C++ and Geant4 and ROOT, oh my!

Segfaults are the norm when working with ROOT.

ROOT is object oriented. If you want to use a histogram you better know the name of the class (TH1) and if you want to make a histogram you better know what the member functions of the class are. Replace histogram with anything you will ever do. If you know the name of the class you can google it (eg follow me), if you don’t know the name of the class you’re screwed!
Outline

Getting Your Bearings

The Basics

Fitting

Display

Getting to Your Data
Let’s Get Acquainted

Where Am I

- shell prompt: `[user@pc ~]$`
- ROOT prompt: `root [0]`

Starting ROOT

- `[user@pc ~]$ root -l`

Quitting ROOT

- `root [0] .q`

Shell Commands in root start with “.!”

- `root [0] .! ls . | grep pdf`

ROOT can do simple calculations. But be careful, ROOT is using a C++ interpreter. It knows some things but not everything. Do not forget decimal points when using floating numbers!
Make a Plot, Save a Plot

Let’s draw a sine function. ROOT will automaticall take care of most things for us. All we need to do it tell root what our function is (and then of course tell ROOT to draw the function).

To save the plot type $\text{Alt}+\text{F}$ then s then d to save as a pdf file. When you save the plot with an extension it will write the data to file in the appropriate format such as pdf or eps, take note of all the saving options (ps,eps,pdf,gif,jpg,png,C,root).
Some Object Oriented Fun

Now let’s make the sine function a little more robust. Try creating the sine function the following way:

```cpp
TF1 *pf = new TF1("f","[0]+[1]*TMath::Sin([2]*x+[3])","-10,10");
pf->SetParName(0, "offset");
pf->SetParName(1, "amplitude");
pf->SetParName(2, "frequency");
pf->SetParName(3, "phase");
pf->SetParameter(0, 6);
pf->SetParameter(1, 3);
pf->SetParameter(2, 5);
pf->SetParameter(3, 1);
pf->Draw();
```
1D Histograms TH1

To Draw a histogram you must do the following steps. Feel free to follow along.

- **Construct**: `TH1F * p_histogram = new TH1F("histogram","Title:Horizontal Axis Label",100,0,10)`
- **Fill**: `p_histogram->Fill(3.)`
- **Draw**: `p_histogram->Draw()`

Did you notice that the title for this slide is a hyperlink to the online documentation of TH1F?
Functions TF1

TMath::Pi()
ROOT files
.C files
Fitting Functions Theory

Presumably one would like to describe their results quantitatively with a theory. Often this theory will take the form of a function. ROOT offers the user two ways to fit data to functions: **Standard Packages** and **RooFit** (see [http://roofit.sourceforge.net/](http://roofit.sourceforge.net/)). For simple tasks the standard packages are sufficient. More complex tasks such as maximum likelihood analysis should be tackled with roofit. Roofit is beyond the scope of this work and will not be mentioned again (maybe).

Using the Standard ROOT packages to execute a fit the user must:

- Create a function
- Create the object to fit
- Initialize the parameters of the function (**good guesses are critical**)
- Execute a fit command
### Fitting Functions Example

- Assume `pHist` is a pointer to a histogram you want to fit

```cpp
TF1 *pf2 = new TF1("f2","[0]+[1]*TMath::Sin([2]*x+[3])",-10,10);
pf2->SetParName(0,"offs");
pf2->SetParName(1,"ampl");
pf2->SetParName(2,"freq");
pf2->SetParName(3,"phas");
pf2->SetParameter(0,6);
pf2->SetParameter(1,3);
pf2->SetParameter(2,5);
pf2->SetParameter(3,1);
pf2->Draw();
pHist->Fit("f2","R+");
pHist->Draw();
```
ROOT Display

Again C++ objects

- **TCanvas**
- **TPad**
- **TF1**
~/.rootlogon.C

Default ROOT Appearance

Modified ROOT Appearance

- The default root appearance is atrocious
- Edit ~/.rootlogon.C to fix
- Use TStyle::Set methods to choose defaults (sample on next line)
  - pTStyle->SetCanvasColor(0); // Sets Canvas color to white
• ~/rootlogon.C cannot fix everything
• The canvas should be transparent
• call canvas->SetFillStyle() to change transparency of canvas
• But style->SetCanvasFillStyle() does not exist
• You must set the transparency by hand after the canvas is constructed
Double Down or Split

double pi=3.14159;
TF1 * psin = new TF1("sin","sin(x)",0.,2.*pi);
TF1 * pcos = new TF1("cos","cos(x)",0.,2.*pi);

```c
pcos->SetLineColor(4);
psin->SetLineColor(2);
pcos->Draw();
psin->Draw("same");
```

```c
pTCanvas->Divide(2,1);// r,c
pTCanvas->cd(1);
psin->Draw();
```

L.P. Alonzi (UVa) 17 / 26
selector
TBrowser

So now you’ve got this root file and you want to look at it.

- root [0] new TBrowser
- Navigate around
- Dopple Click on a root file
- Find a folder with a tree on it
- Right Click it and select “StartViewer”
Sometimes you know you want to analyze your root files in the same way every time. There is no need to use the TBrowser/TreeViewer, just make some macaroni and cheese.

L.P. Alonzi (UVa)
Macaroni and Cheese
(Macros and Chains)

- Create a file called blerg.C:

```c
void blerg() {
    TChain * pchain = new TChain("ntuple");
    TH1F * phist1 = new TH1F("hist1", "", 160, 0., 80.);
    TH1F * phist2 = new TH1F("hist2", "", 160, 0., 80.);
    pchain->Add("file_1.root"); // nb: ntuple must be in file_1.root
    pchain->Add("file_2.root"); // and so on
    Float_t g = 68./68.4;
    pchain->Draw("tree_leaf>>phist1","cuts");
    // the >> fills phist1 with the variable tree_leaf
    pchain->Draw(Form("tree_leaf*%f>>phist2","cuts");
    // Form replaces %f in the string with g
    phist1->Draw();
    phist2->Draw("same");}
```

- root [0] .x blerg.C
You can’t just use `g++` to compile C++ code with root classes. You should use a makefile. E.G. where `foo` is the name of the file (`foo.c`) that contains the main function.

```bash
ROOTCFLAGS = $(shell $(ROOTSYS)/bin/root-config –cflags)
ROOTLIBS = $(shell $(ROOTSYS)/bin/root-config –libs)
ROOTGLIBS = $(shell $(ROOTSYS)/bin/root-config –glibs)
CXX = g++
CXXFLAGS = -g -Wall -O2 -fPIC
LD = g++
LDFLAGS = -g
SOFLAGS = -shared
CXXFLAGS += $(ROOTCFLAGS)
LIBS = $(ROOTLIBS)
NGLIBS = $(ROOTGLIBS)
NGLIBS += -lMinuit
GLIBS = $(filter-out -lNew, $(NGLIBS))
.SUFFIXES: .cc,.C
all: foo
foo: foo.o
$(CXX) $(CXXFLAGS) -o $@ foo.o $(GLIBS)
clean::
rm -f *.o * #*
```
but windows don’t open

If you want to open a canvas when running compiled code you must jump through an extra hoop. The TApplication class must be invoked. Simply create an application at the beginning of your code then after your work issue the Run() subroutine. E.G.

```c
TApplication myapp("App",0,0);
TFile * alpha = TFile::Open("filename.root","READ");
TFolder * pf = (TFolder*) alpha->Get("histos");
TH2F* ph = (TH2F*) pf->FindObject("Check/h2TIMING");
ph->Draw();
myapp.Run();
```
Assume psim4 and pmeas are pointers to histograms

- `psim4` → `Draw();`
- `pmeas` → `Draw("p,same");`
- `TLegend * pleg = new TLegend(0.6,0.7,0.89,0.89);`
- `pleg` → `AddEntry(pmeas,"Measurement","P");`
- `pleg` → `AddEntry(psim4,"Simulation","F");`
- `pleg` → `Draw();`
Loose Strings

Form(“st%fring”, float)
fin