

An Introduction to ROOT



Gerhard Brandt

(Humboldt Universität zu Berlin)



Summer Student Tutorial
28 Juli 2011

ROOT

An Object-Oriented
Data Analysis Framework



What we do

- Go through these slides
- Run some of the tutorials that come with ROOT
`$ROOTSYS/tutorials`
- More slides and exercises on cosmetics (style) by Mira Krämer

What you do

- Try to run the same steps we demonstrate
- ***Ask questions at any time***

Introduction



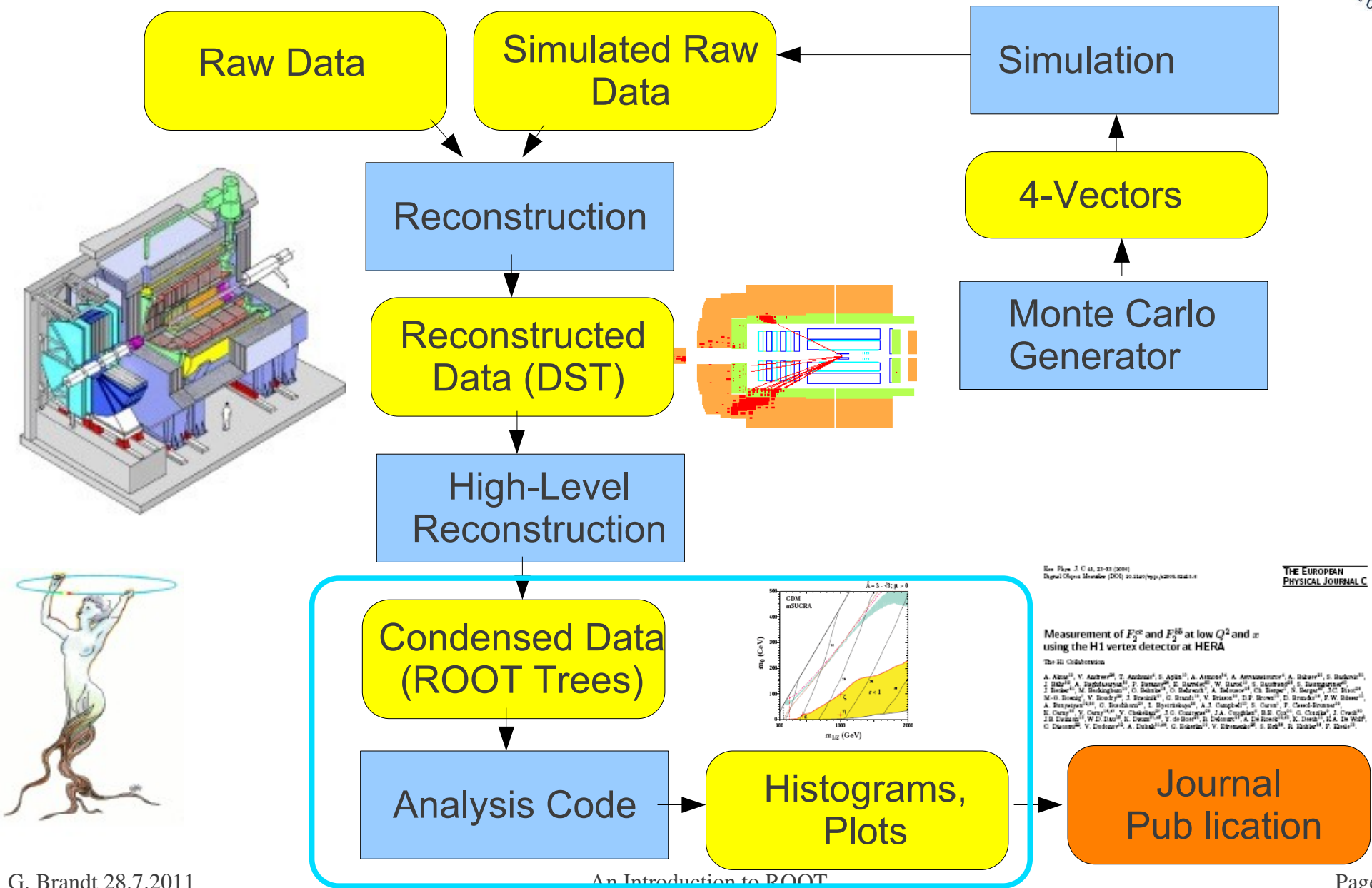
ROOT is a Package for Data Analysis

ROOT Provides:

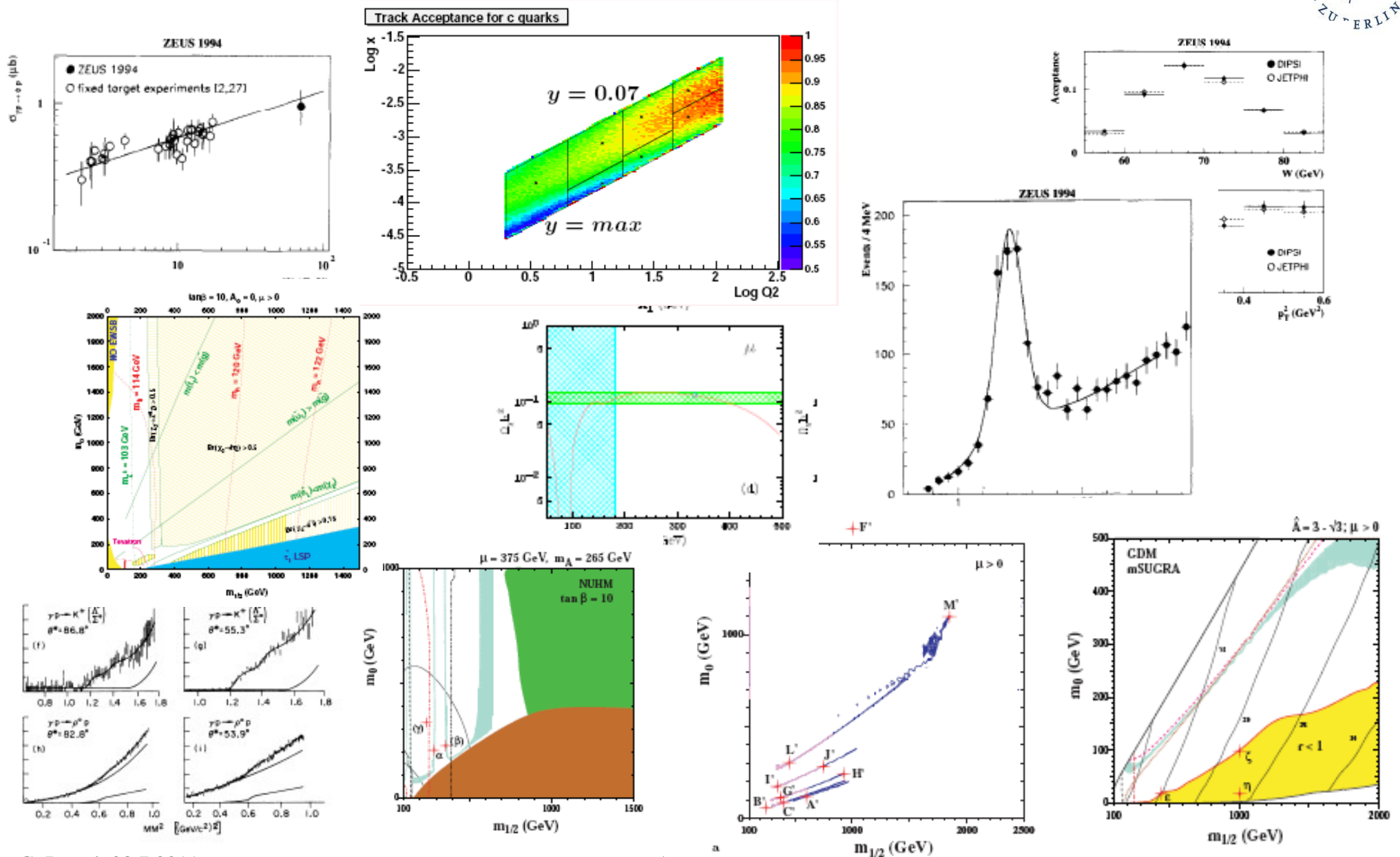
- Several C++ Libraries
 - To store data in histograms
 - To store data in n-tuples, called “ROOT Trees”
 - To visualize histograms and n-tuples
 - To perform fits
- An Interactive Environment
 - To run C++ programs interactively (C++ interpreter CINT)
 - To visualize data
 - To perform fits



The Analysis Chain in High Energy Physics



Histograms are Important in HEP



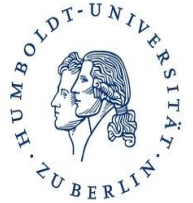
ROOT Information



- Web page: <http://root.cern.ch/>
- We use pro-version ROOT 5.30.00
- Experiment's Software may use previous versions + specific fixes
- You can download ROOT yourself and compile it, for Linux, MacOS and Windows
- There is a User's guide at (still for 5.26)
<http://root.cern.ch/drupal/content/users-guide>
- Reference for all versions is available at
<http://root.cern.ch/drupal/content/reference-guide>
- The Class Index for the current pro version is available at
<http://root.cern.ch/root/html/ClassIndex.html>

The most important link in HEP – make it your homepage

Class Index and Class Reference





ROOT

Quick Links: ROOT Homepage Class Index

Class Index

Modules: ROOSTATS TMVA BINDINGS CINT MATH MISC MONTECARLO NET P

Jump to: C ROOT: ROOT::Math: ROOT::Math:: ROOT::Math::SV ROOT::T ROOT::TI TEveG TEveP TEveT TF TG TGH TGU TI TM TMV TMVA: TMa T

ColorStruct_t	
CpuInfo_t	
Event	
EventHeader	
Event_t	
FileStat_t	
FontAttributes_t	
FontMetrics_t	
Foption_t	
GCValues_t	
HistogramManager	Manages all histograms
MemInfo_t	Memory utilization information.
PictureAttribute_t	
Point_t	
ProcInfo_t	System resource usage of given process.
PyROOT	
PyROOT::TPyException	C++ exception for throwing python exceptions
PyROOT::TPyROOTApplication	Setup interactive application
ROOT	

class TH1F: public TH1, public TArrayF

TH1F methods

TH1F : histograms with one float per channel. Maximum precision 7 digits

Function Members (Methods)

public:

```
TH1F ()
TH1F (const TVectorF& v)
TH1F (const TH1F& h1f)
TH1F (const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)
TH1F (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)
TH1F (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)

virtual ~TH1F ()
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 bin, Int_t) const
virtual Double_t GetBinContent (Int_t bin, Int_t, Int_t) const
virtual TClass* IsA () const
TH1F& operator= (const TH1F& h1)
virtual void Reset (Option_t* option = "")
virtual void SetBinContent (Int_t bin, Double_t content)
virtual void SetBinContent (Int_t bin, Int_t, Double_t content)
virtual void SetBinContent (Int_t bin, Int_t, Int_t, Double_t content)
```

```
void TH1F::SetBinContent(Int_t bin, Double_t content)
{
    // Set bin content
    // see convention for numbering bins in TH1::GetBin
    // In case the bin number is greater than the number of bins and
    // the timedisplay option is set or the kCanRebin bit is set,
    // the number of bins is automatically doubled to accomodate the new bin
    if (bin < 0) return;
    if (bin >= fNcells-1) {
        if (fXaxis.GetTimeDisplay()) {
            while (bin >= fNcells-1) LabelsInflate();
        } else {
            if (!TestBit(kCanRebin)) {
                if (bin == fNcells-1) fArray[bin] = Float_t (content);
                return;
            }
            while (bin >= fNcells-1) LabelsInflate();
        }
    }
    fArray[bin] = Float_t (content);
    fEntries++;
    fTsumw = 0;
}
```

Source Code
The best
Documentation

- The full story: All ROOT Classes are documented via THtml

Interactive ROOT

- Setup ROOT
 - Depends on environment
 - Possibility if you just know ROOT's location:

```
$ROOTSYS> sh bin/thisroot.sh
```
- At DESY/NAF: You can setup ROOT version with `ini` command

```
$> ini root530
```
- Be prepared to have to use different setups and versions of ROOT!
- Start ROOT interactively with

```
$> root
```
- At the ROOT prompt, enter

```
root [1] TBrowse t;
```
- This opens a browser GUI to show current ROOT objects, files etc in memory

CINT



- ROOT uses a C++ interpreter CINT for interactive use
- You can enter any C++ command; trailing “;” is not required
- Resetting the interpreter (erasing variables etc):

```
root [ ] gROOT->Reset ( )
```

But more often a restart of ROOT is needed...

(my strategy: run ROOT in batch mode as much as possible)

- Special CINT commands start with a dot:
 - .q Quit
 - .x script.C Execute script “script.C”
 - .L script.C Load script “script.C” (if script.C contains class definitions)
 - .? Show all special commands
- More in Chapter 7: “CINT the C++ Interpreter” of ROOT manual

CINT Extensions to C++



- If you create a pointer and assign to it with “new”, you don't need to declare the pointer type:

```
h = new TH1F ("h", "histogram", 100, 0, 1)
```

- h is automatically of type TH1F*

- “.” can be used instead of “->”
=> Don't do that habitually!

- If you use a variable that has not been declared earlier, ROOT tries to create one for you from all named objects it knows
=> If you have opened a file that contains a histogram “hgaus”, you can directly use

```
hgaus->Draw( )
```

- But be careful: Sometimes you get a different object than you thought :-)

- Sometimes (often...) objects get created automatically
 - Eg. TTree::Draw() creates a histogram and a canvas

ROOT Histograms



- The most important class in ROOT for data analysis
- 1-Dimensional Histograms: class TH1F
 - Gives the number of entries versus one variable
 - By far the most common type
- 2-Dimensional Histograms: class TH2F
 - Gives the number of entries versus two variables
 - Used to show dependencies/correlations between variables
- Profile Histograms: class TProfile
 - Gives the average of one variable versus another variable
 - Used to quantify correlations between variables
 - Often used to quantify reconstruction resolutions/biases:
Plot reconstructed quantity versus true (“generated”) quantity in Monte Carlo events

What Histograms Can Do



- **Booking**

```
TH1F(const char* name, const char* title, int nbinsx, double xlow, double xup);  
TH1F(const char* name, const char* title, int nbinsx, const double* xbins);
```

- **Filling**

```
virtual int Fill(double x);  
virtual int Fill(double x, double w);
```

- **Getting information**

```
virtual double GetBinContent(int bin) const;  
virtual double GetMaximum(double maxval = FLT_MAX) const;  
virtual double GetMaximum(double maxval = FLT_MAX) const;
```

- **Adding etc.**

```
virtual void Add(TF1* h1, Double_t c1 = 1, Option_t* option);  
likewise: Multiply, Divide
```

- **Drawing**

```
virtual void Draw(Option_t* option);
```

- **Writing to a file (inherited from TObject)**

```
virtual int Write(const char* name = "0", int option = 0, int bufsize = 0);
```

A Histogram Code Example

file gausexample.C:

```
#include <TH1.h>
#include <TFile.h>
#include <TRandom.h>

int main() {
    TH1F *histo = new TH1F ("hgaus", "A Gauss Function", 100, -5.0, 5.0);
    TRandom rnd;

    for (int i = 0; i < 10000; ++i) {
        double x = rnd.Gaus (1.5, 1.0);
        histo->Fill (x);
    }

    TFile outfile ("gaus.root", "RECREATE");
    histo->Write();
    outfile.Close();
    return 0;
}
```

Here we “book” the histogram

- ID is “hgaus” (must be unique, short, no spaces)
- Title is “A Gauss Function”
- 100 bins between -5 and 5

rnd is an object of type TRandom,
a random number generator.

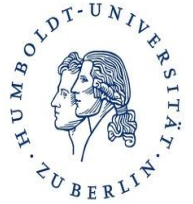
rnd.Gaus returns a new Gaussian distributed
random number each time it is called.

Open the ROOT output file
Write the histogram to it
Close the output file

Compile and run:

```
$> g++ -I `root-config --incdir` -o gausexample gausexample.C `root-config --libs`
$> ./gausexample
```

TF1 Functions and Fitting



file tflexample.C:

```
#include <TH1F.h>
#include <TF1.h>
#include <TFile.h>

Double_t mygauss (Double_t *x, Double_t *par) {
    // A gauss function, par[0] is integral, par[1] mean, par[2] sigma
    return 0.39894228*par[0]/par[2]*exp(-0.5*pow(*x -par[1])/par[2], 2));
}

int main() {
    TF1 *gaussfun = new TF1 ("gaussfun", mygauss, -10, 10, 3);
    gaussfun->SetParameters (100, 0., 1.);
    gaussfun->SetParNames ("Area", "Mean", "Sigma");
    TFile *file = new TFile ("gaus.root");
    TH1F *hgaus = dynamic_cast<TH1F *>(file->Get("hgaus"));
    if (hgaus) {
        hgaus->Fit(gaussfun);
    }
}
```

Defines a Gauss function

Note that the argument must be handed over by a pointer!!!

Defines a TF1 function object

- ID is "gaussfun"
- It executes function mygauss
- It is valid for x between -10 and 10
- It has 3 parameters

Here we load the histogram "hgaus" from the file "gaus.root", and if it was found, we fit it.

`file->Get()` returns only a pointer to a `TObject`, which is a base class of `TH1F`.

With `dynamic_cast` we convert the pointer to the correct type.

If the object pointed to is not a `TH1F` (it could something completely different!), the `dynamic_cast` returns a null pointer.

Remark: ROOT Coding Conventions

ROOT uses some unusual coding conventions
just get used to them...

- Class names start with capital T: TH1F, TVector
- Names of non-class data types end with `_t`: Int_t
- Class method names start with a capital letter: TH1F::Fill()
- Class data member names start with an f: TH1::fXaxis
- Global variable names start with a g: gPad
- Constant names start with a k: TH1::kNoStats
- Seperate words with in names are capitalized: TH1::GetTitleOffset()
- Two capital characters are normally avoided: TH1::GetXaxis(),
not TH1::GetXAxis()

Clicking

Click here to display a histogram

Click here to open a file

Enter this to get the browser window

The screenshot displays the ROOT software interface with three main windows:

- ROOT Object Browser:** Shows a file tree with 'gaus.root' selected. A red circle highlights the 'gaus,1' object in the 'Contents of "/ROOT Files' pane. Another red circle highlights the 'gaus.root' folder in the 'All Folders' pane.
- c1:** A plot window titled 'A Gauss Function' showing a histogram of a Gaussian distribution. The x-axis ranges from -10 to 10, and the y-axis ranges from 0 to 400. A statistics table in the top right corner shows:

gaus	
Entries	10000
Mean	1.493
RMS	0.9803
- Shell - Konsole:** A terminal window showing the command prompt and the execution of the ROOT browser. A red circle highlights the command `root [1] TBrowser t;`. The terminal output includes:

```
CINT/ROOT C/C++ Interpreter version 5.15.169, Mar 14 2005
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
Attaching file gaus.root as _file0...
root [1] TBrowser t;
root [2] <TCanvas::makeDefCanvas>: created default TCanvas with name c1
```


ROOT Command Line



```
$> root

root [0] TFile *file0 = TFile::Open("gaus.root")
root [1] hgaus.Draw()
root [2] hgaus.Draw("E")
root [3] hgaus.Draw("C")
root [4] gStyle->SetOptStat(1111111)
root [5] hgaus.GetAxis()->SetTitle("Abscissa")
root [6] hgaus.GetAxis()->SetTitle("Ordinate")
root [7] gPad->SetLogx(1)
root [8] hgaus.Draw("E2")
root [9] hgaus.SetLineColor(3)
root [10] hgaus.SetLineStyle(2)
root [11] hgaus.SetLineWidth(2)
root [12] hgaus.SetMarkerStyle(20)
root [13] hgaus.SetMarkerSize(1.5)
root [14] hgaus.SetMarkerColor(4)
root [15] hgaus.Draw("E1")
root [16] hgaus.SetFillColor(4)
root [17] hgaus.Draw("C")
root [18] gPad->Print("gaus1.ps")
root [19] .q
```

Interpreted scripts

- Un-named scripts:

```
{  
  #include <iostream.h>  
  cout << "Hello, World!\n";  
}
```

includes not needed

- Code must be enclosed in curly braces!

- Execute with

```
root[] .x script.C
```

- Named scripts:

```
#include <iostream.h>  
int main() {  
  cout << "Hello, World!\n";  
}
```

- More like normal C++ programs, recommended form!

- Execute with:

```
root[] .L script.C  
root[] main()
```

Compiled programs linked with ROOT



- Will normally be done by a Makefile
- Command “root-config” tells you necessary compiler flags:

```
$> root-config --incdir  
/opt/products/root/5.18.00/include  
$> root-config --libs  
-L/opt/products/root/5.18.00/lib -lCore -lCint -lHist -lGraf  
-lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics  
-pthread -lm -ldl -rdynamic
```
- To compile a file Example.C that uses root, use:

```
$> g++ -c -I `root-config --incdir` Example.C
```
- To compile and link a file examplemain.C that uses root, use:

```
$> g++ -I `root-config --incdir` -o examplemain  
examplemain.C `root-config --libs`
```
- The inverted quotes tell the shell to run a command and paste the output into the corresponding place
- There is also a hybrid between interpreting and compiling: ACLIC

Let's try now what we've seen so far...

Interpreted Example

- Set up ROOT
- Go to \$ROOTSYS/tutorials/hist
- \$> root
- root [0] .x fillrandom.C

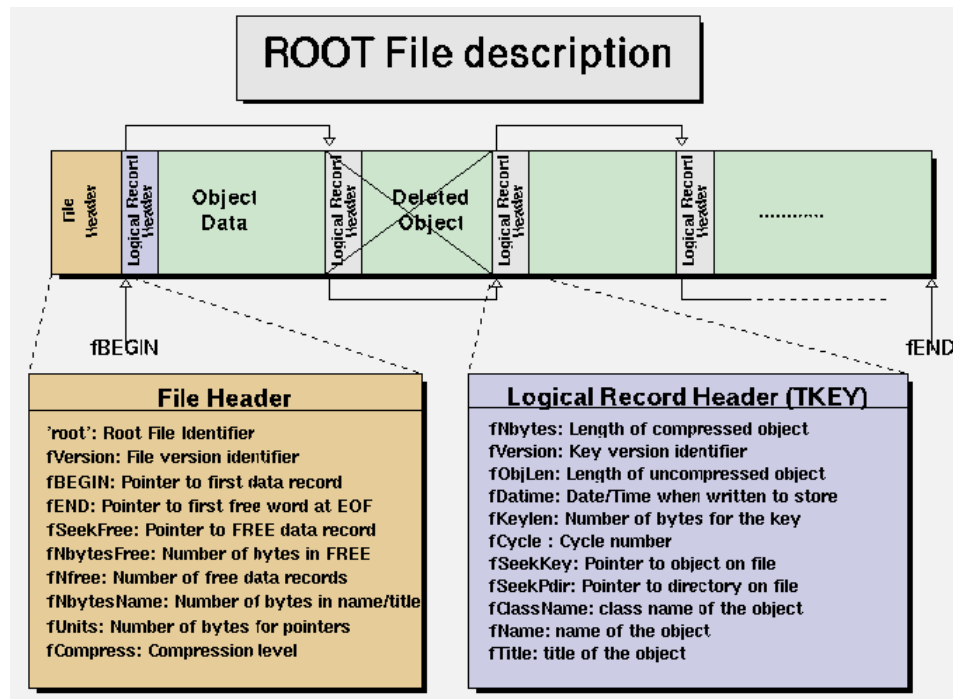
Compiled Example

- Go to \$ROOTSYS/test
- \$> make hsimple
- ./hsimple

For each example inspect the output root File with the TBrowser

TFile and TDirectory

- Root TFiles contain a sequence of ROOT objects stored in TKey's



- TFile's derive from TDirectory
 - These form a directory hierarchy within ROOT
 - Like a meta-level file system
 - Can be confusing at first (and later...)
- Loading and looking at contents of a ROOT File without the TBrowser
 - `$ root -l ntuple.root`
 - `$.ls`
 - `$_file0->ls()`

Five Minutes on ROOT Trees

- A ROOT Tree holds many data records of the same type, similar to an n-tuple. One record can be described by a C++ Class:

```
class EventData {  
    public:  
        Int_t run;  
        Int_t event;  
        Float_t x;  
        Float_t Q2;  
};
```

- The ROOT Tree knows how many entries (here: events) it contains.

It can fill one instance (one object) of class EventData at a time with data, which we then can use to plot the data.

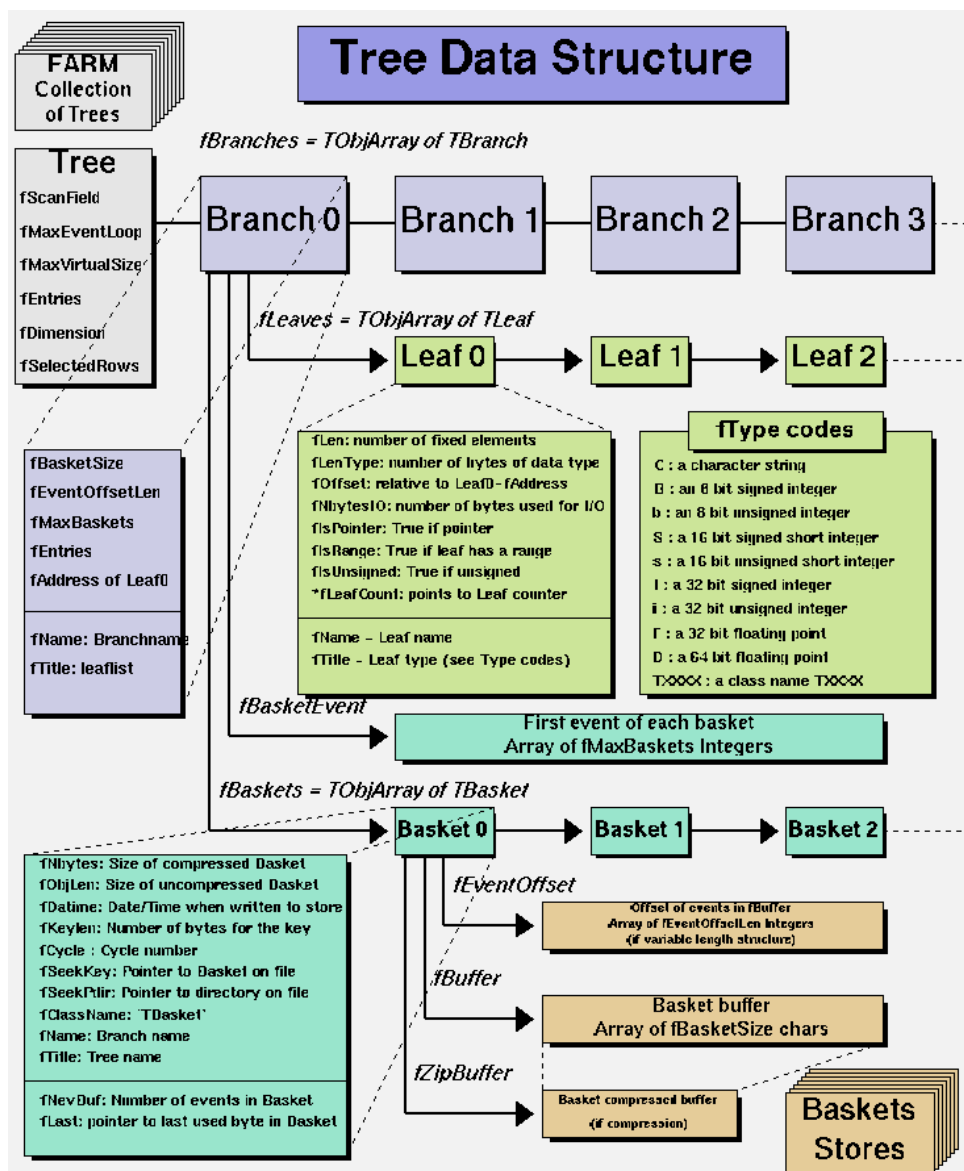
```
TH1F *histox = new TH1F ("histox", "Bjorken x", 1000, 0., 1.);  
TFile *file ("eventdata.root");  
TTree *tree = dynamic_cast<TTree *>(file->Get("eventdata"));  
EventData *thedata = new EventData;  
TBranch *branchx = tree->GetBranch("x");  
branchx->SetAddress (&(event->x));  
for (int i = 0; i < tree->GetEntries(); ++i) {  
    branchx->GetEntry(i);  
    histox->Fill (x);  
}
```

Trees, Branches, and Leaves



- A TTree can contain the whole data set
- But in “real” root files, often more than one tree are used
 - Can be associated via “friendship”
- A TTree spread over several TFiles is a TChain
- A TBranch contains the data of one or several variables, e.g. the x and Q2 values of all events.
 - A TTree consists of several TBranches.
 - How the TBranches are set up is determined by the program that writes the Tree
- A TLeaf is the data of a single variable (like x)
 - A TBranch consists of several TLeaves

Structure of a ROOT Tree



Logical Organisation

- A TTree has many entries
- A TTree contains many TBranches
 - They can hold single variables (“n-tuple”) or complex objects

Physical Organisation

- Each branch is saved in several TBaskets containing a certain number of entries
- TBasket: minimal amount of data that has to be read from disk
- TBaskets are zipped

Using Trees



Creating

- The `hsimple.cxx` example was creating a `TNtuple`, a simplified derivative of a `TTree`
- For some examples to create `TTrees` see [\\$ROOTSYS/tutorials/tree/tree*.C](#)

Reading

- You will have an “event loop” which loops over all entries of the tree.
- Use this data to select “good” entries and plot their properties in histograms.
- The most simple way to use a `TTree` to do these steps is

`TTree::Draw()`

The Sketch of an Analysis Program



```
int main() {
    // some initializations here:
    //   reading steering parameters
    //   open event files

    // Book histograms

    for (int i = 0; i < events; ++i) {
        // Load event number i into memory
        // Get/calculate event properties
        if (selection_is_filfilled) {
            // fill histograms
        }
    }

    // draw the histograms
    // write out histogram file
    // write out info like number of events etc...
    return 0;
}
```

The skeleton of such an analysis program will typically be provided to you by your supervisor

That's all for today...



- ROOT is best learnt by doing (like everything...)
- For support directly from the ROOT team there is the RootTalk mailing list and forum
 - ROOT support is know to be excellent, but please first ask locally or RTFM
- If you want, you can look at the sister-tutorial in the CERN summer students lecture

<http://indico.cern.ch/conferenceDisplay.py?confId=134329>

BACKUP



Drawing Options for 1D-Histograms



"AXIS "	Draw only axis
"AH "	Draw histogram, but not the axis labels and tick marks
"] ["	When this option is selected the first and last vertical lines of the histogram are not drawn.
"B "	Bar chart option
"C "	Draw a smooth Curve through the histogram bins
"E "	Draw error bars
"E0 "	Draw error bars including bins with 0 contents
"E1 "	Draw error bars with perpendicular lines at the edges
"E2 "	Draw error bars with rectangles
"E3 "	Draw a fill area through the end points of the vertical error bars
"E4 "	Draw a smoothed filled area through the end points of the error bars
"L "	Draw a line through the bin contents
"P "	Draw current marker at each bin except empty bins
"P0 "	Draw current marker at each bin including empty bins
"*H "	Draw histogram with a * at each bin
"LF2 "	Draw histogram like with option "L" but with a fill area. Note that "L" draws also a fill area if the hist fillcolor is set but the fill area corresponds to the histogram contour.

 Tutorials by Mira

Drawing Options for 2D-Histograms

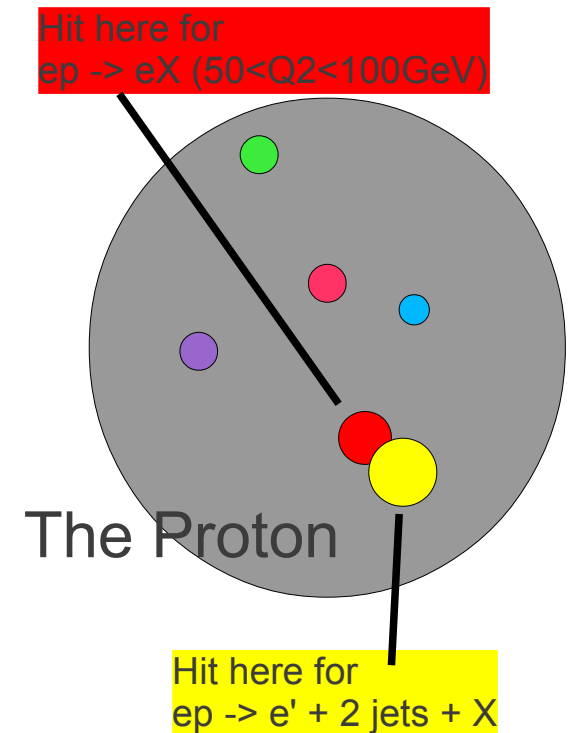


AXIS	Draw only axis
ARR	arrow mode. Shows gradient between adjacent cells
BOX	a box is drawn for each cell with surface proportional to contents
COL	a box is drawn for each cell with a color scale varying with contents
COLZ	same as "COL". In addition the color palette is also drawn
CONT	Draw a contour plot (same as CONT0)
CONT0	Draw a contour plot using surface colors to distinguish contours
CONT1	Draw a contour plot using line styles to distinguish contours
CONT2	Draw a contour plot using the same line style for all contours
CONT3	Draw a contour plot using fill area colors
CONT4	Draw a contour plot using surface colors (SURF option at theta = 0)
CONT5	Draw a contour plot using Delaunay triangles
LIST	Generate a list of TGraph objects for each contour
FB	Draw current marker at each bin including empty bins
BB	Draw histogram with a * at each bin
SCAT	Draw a scatter-plot (default)
TEXT	Draw bin contents as text
TEXTnn	Draw bin contents as text at angle nn ($0 < nn < 90$)
[cutg]	Draw only the sub-range selected by the TCutG named "cutg"

 Tutorials by Mira

What is a Cross Section?

- Imagine small area on proton's surface
 If area σ is hit by electron, an event of a certain type happens
 Unit of σ cm², or barn: 1 barn = 10⁻²⁴ cm² = (10fm)²
 Area of proton: approx 0.02 barn (radius 0.8fm)
 Typical cross sections at HERA: pb (10⁻³⁶ cm²)
- Instantaneous luminosity \mathcal{L} :
 Number of events per second per cross section
 Unit of \mathcal{L} : cm⁻² s⁻¹, or nb⁻¹ s⁻¹
 HERA-II Design Lumi:
 5·10³¹ cm⁻² s⁻¹, or 50 μ b⁻¹ s⁻¹
- Integrated luminosity: $\int \mathcal{L} dt$
 Number of events per cross section
 Unit of $\int \mathcal{L} dt$: cm⁻², or pb⁻¹
 HERA-II values: order 100pb⁻¹



How Do we Measure a Cross Section?



- The Master Formula:

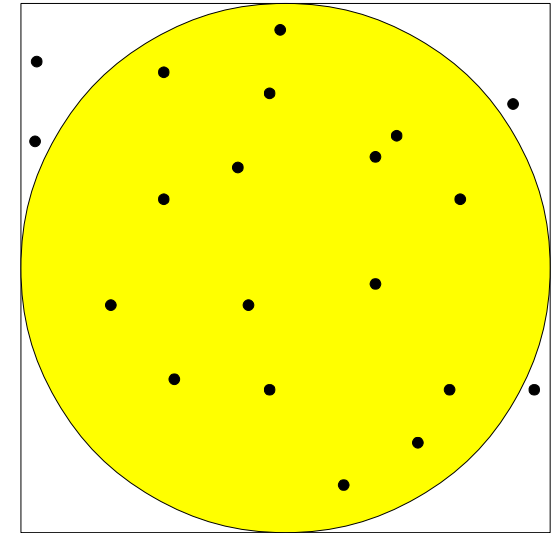
$$\text{Number of events: } N = \sigma \cdot \int \mathcal{L} dt$$

- We count events for a given data sample
=> observed number of events N_{obs}
- For this data sample, we know the integrated luminosity $\int \mathcal{L} dt$
- We are generally interested for cross sections for theoretically well defined processes, e.g. for $ep \rightarrow e' X$, $0.001 < x < 0.002$, $5 < Q^2 < 6 \text{ GeV}^2$
- But we can only count events which we have observed, and where we have reconstructed certain x , Q^2 values, which are not exact
- => We have to correct the observed number of events for background, trigger and reconstruction inefficiencies, and resolution effects

How Do we Correct for Detector Effects?



- Analytical calculations generally not possible
- The Monte Carlo Method:
“Generate events” randomly, which have the expected distributions of relevant properties (x , Q^2 , number of tracks, vertex position...)
- Simulate detector response to each such event (hits in chambers, energy in calo)
- Pass events through same reconstruction chain as data
- Now we have events where we can count events that truly fulfill our cross section criteria, and those which pass the selection criteria. The ratio is called “efficiency” and is used to correct the data



Measuring π with the Monte Carlo method:
The fraction f of random points within the circle is $\pi/4$.
We measure: $f = 16/20 = 0.8$
Uncertainty on f : $\text{sqrt}(f*(1-f)/N) = 0.09$
So: $\pi/4 \sim f = 0.80 \pm 0.09$
and $\pi \sim 4f = 3.2 \pm 0.3$

How Do we Count Events?

Typically: Write (and run) a program that

- Selects events with certain properties, e.g.:
 - Scattered electron with energy $E'_e > 10\text{GeV}$
 - Tracks visible that come from a reconstructed vertex with $-35 < z < 35\text{cm}$
 - Reconstructed Bjorken- $x > 0.001$
- Counts events in “bins” of some quantity, e.g. Q^2 :
 $Q^2 = 10\dots20, 20\dots30, 30\dots40, \dots$
- Shows the number of events as a histogram