ROOT tutorial, part 1

Attilio Andreazza  Università di Milano
Caterina Doglioni  Université de Genève

HASCO school – 17/07/2012
What is ROOT?  http://root.cern.ch/

An object oriented framework for large scale data analysis
What is “Object-Oriented Programming”? (1991 revised version)

Bjarne Stroustrup
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Object-oriented programming

From Wikipedia, the free encyclopedia

Object-oriented programming (OOP) is a programming paradigm using "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. Many modern programming languages now support OOP, at least as an option.
...framework...

ROOT: a set of reusable **classes** and **libraries**

ROOT in **interactive mode**

ROOT in **compiled code**

```
cate@catelenovolinux:~$ root -l
root [0] TF1 *myFunction = new TF1("myFunction ","[0]+[1]*x",0,10);
root [1] 
```

```
#include "TF1.h"

int main() {

    TF1 *myFunction =
    new TF1("myFunction","[0]+[1]*x",0,10);

    delete myFunction;

    return(0);
}
```
...for large scale...

Enormous amount of data recorded by e.g. the LHC:

Need efficient data formats and tools to:
store the data
read data out (I/O)
extract information from data

(this plot has been made with ROOT)
...data analysis

Analysing data involves:

- Recording and storage of data/MC
- Reconstruction of physics objects
  - Discrimination of signal from background (e.g. using cuts)
- Quantitative comparison of predictions to experimental results
  - Presentation results (usually using plots)

...and much more:
ROOT is used to do all of this

this tutorial: final data analysis
Documentation and links

Class reference

ROOTTalk (forum)

ROOT manual

Tutorial disclaimer: partial / personal view of all that ROOT can do...

ROOT Support

Moderator: rootdev

NEWTOPIC 11874 topics • Page 1 of 238 • 1 2 3 4 5 ... 122

TOPICS

ROOT has moved to Subversion...
by rdm  WedOct 10, 2007 11:54

Root in Cygwin on Win7 doesn’t respond after submission
by willisfer  Tue Jul 10, 2012 20:42

qlcint.so.5.31 cannot be built in trunk@41669
by Pope Le Pew  Tue Nov 01, 2011 18:02

TTree::Draw and TVector Branches
by jtcaron  Thu Jul 05, 2012 22:22

User's Guide

The ROOT User's Guide has been translated into DocBook (XmI). The corrections and updates are now made in this new format. The new version is still under development, therefore we will continue to provide, for a limited duration, the old version (see below on this page).

Latest User's Guide (A4 format)
Latest User's Guide (HTML version)

Old version:

We will appreciate your comments on this edition. If you would like to contribute to a chapter, section, or even a paragraph, do not hesitate to contact us and send your comments to: rootdoc@root.cern.ch. You can also post your comments or questions in the section Documentation of the ROOT Forum.

Files available for download:

User's Guide v5.26
1 page per sheet ~11MB (with Hyper-links)

User's Guide v5.26 TwoInOne
2 pages per sheet ~7MB

User's Guide v5.26 MSWord Doc ~13MB

User's Guide v5.26
Preface, Table of Contents and Table of Figures
Using ROOT: interactive (CINT), ACLiC

From now on, raise your hand if you want any of the lines of code written out & demonstrated live!

ROOT Tutorial
HASCO school – 17/07/2012
Start and quit ROOT

cate@catelenovolinux:~$ root

******************************************************************************
*                                                                        *
*                     WELCOME to ROOT                                      *
*              Version  5.32/01  29 February 2012                        *
*                                                                        *
*             You are welcome to visit our Web site                      *
   http://root.cern.ch                                                   *
*                                                                        *
******************************************************************************

ROOT 5.32/01 (tags/v5-32-01@43181, Feb 29 2012, x8664gcc)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]

No splash screen

cate@catelenovolinux:~$ root -l
root [0]

To quit

cate@catelenovolinux:~$ root -l
root [0] .q

cate@catelenovolinux:~$

To quit a stubborn session

root [0] .qqqq
Info in <TRint::ProcessLine>: Bye... (try ".qqqq qqq" if still running)

cate@catelenovolinux:~$

VERSION 5

Conception: Rene Brun, Fons Rademakers

Lead Developers: Rene Brun, Philippe Canal,
Fons Rademakers

Core Engineering: Bertrand Bellonot, Olivier Couet,
Gerardo Ganes, Andrei Gheata, David Gonzalez,
Jan Iwazkiewicz, Lorenzo Moneta, Axel Naumann,
Paul Russo, Matevz Tadel

Version 5.26/00

07/17/12
CINT: necessary health warning

For most of this tutorial, we will use CINT

CINT is an **interpreter, not a compiler**

A compiler would complain about this liberal use of pointer operators on objects...

CINT has limitations, but it is easy to use on command line and works reasonably for quick plotting purposes
E.g. one advantage: CINT will look for objects in the current directory and save you some typing

However, **bad idea** to learn C++ via CINT...
Macros in CINT

Unnamed macros

```c
{  
  TF1 myFunction;
  myFunction.SetName("myFunction");
  cout << myFunction.GetName() << endl;
}
```

Named macros

```c
void MyFirstMacro(string textToSayHelloToTheFunction) {
  TF1 myFunction;
  myFunction.SetName("myFunction");
  cout << textToSayHelloToTheFunction << " "
      << myFunction.GetName() << endl;
}
```

Tab completion

```
root [0] .L MyFirstMacro.C
root [1] M
missingRuns
mozilla.pdf
myFirstMacro.C
massResolution_J5.eps
root [0] .x myFirstMacro.C
myFunction
root [1] ]
```

Function argument

```
root [0] .x m
cate@catelenovolinux:~$ root -l
root [0] .x m

MemInfo_t
MyFirstMacro
root [1] MyFirstMacro(
void MyFirstMacro(string textToSayHelloToTheFunction)
root [1] MyFirstMacro(
void MyFirstMacro(string textToSayHelloToTheFunction)
root [1] MyFirstMacro("Hello function named")
Hello function named myFunction
root [2] ]
```
Macros in ACLiC

Compiled macros

Let's try with the named macro

```c
Void MyFirstMacro(string textToSayHelloToTheFunction) {
    TF1 myFunction;
    myFunction.SetName("myFunction");
    cout << textToSayHelloToTheFunction << " "
        << myFunction.GetName() << endl;
}
```

Root [0] $L$ MyFirstMacro.C+
Info in `<TUnixSystem::ACLiC>`: creating shared library `/home cate/./MyFirstMacro_C.so`
In file included from `/home cate/MyFirstMacro_C_ACLiC_dict.h:34:0`,
   from `/home cate/MyFirstMacro_C_ACLiC_dict.cxx:17`:
   `/home cate/./MyFirstMacro.C: In function `void MyFirstMacro(std::string)`:
   `/home cate/./MyFirstMacro.C:3:3: error: `TF1` was not declared in this scope
   `/home cate/./MyFirstMacro.C:3:7: error: expected `;` before `myFunction`
   `/home cate/./MyFirstMacro.C:4:3: error: `myFunction` was not declared in this scope
   `/home cate/./MyFirstMacro.C:6:3: error: `cout` was not declared in this scope
   `/home cate/./MyFirstMacro.C:7:35: error: `endl` was not declared in this scope
   g++: error: `/home cate/MyFirstMacro_C_ACLiC_dict.o: No such file or directory`
Error in `<ACLiC>`: Compilation failed!
```

Needs a bit more work...
Macros in ACLiC

Compiled macros

Compiled macros are faster!
Worth thinking about if e.g. reading events from file

#includes
(for each class used)

namespaces
Standard library objects

myFirstMacro.C

Indicates a macro that you can try out in the tarball attached to the agenda

More info on this link

07/17/12
ROOT tutorial – A. Andreazza, C. Doglioni
Mini-introduction to OO in ROOT

ROOT Tutorial
HASCO school – 17/07/2012
An object in ROOT: TF1

A (mathematical) function **TF1** is an **object**: has **data members/methods**

**Constructor:**
makes an **instance** of the object

```
root [0] TF1 f("myFunction", "sin(x)/x", 0, 10)
(root class TF1)40879392
```

**Name**

**Formula**

**Range** (min/max)

**Methods:**
ask for/modify **properties** of the object

```
root [3] cout << f.GetName() << endl
myFunction
root [4]
root [4] f.SetName("myFunctionWithNewName")
root [5]
root [5] cout << f.GetName() << endl
myFunctionWithNewName
```

**Data members:**
properties of the object
generally inaccessible to us (encapsulation)
can be modified with **setters/getters**

From the **TF1.h** class

```
Double_t fXmin; //Lower bounds for the range
Double_t fXmax; //Upper bounds for the range
```
An object in memory: TF1*

What is the difference between an object and a pointer to an object?

- **TF1**: The actual chunk of memory needed to contain a TF1 object's functions accessed with . (e.g. `function.GetName()`)

- **TF1***: A smaller chunk of memory pointing to the object object's functions accessed with -> (e.g. `function->GetName()`)

**Nasty things** can happen if this link is broken (e.g. pointer doesn't point anywhere anymore...)

Good practice to **check the pointer**: a broken link will show up as a **null pointer**

```
root [1] invalidpf->GetName()
Error: illegal pointer to class object invalidpf 0x0 743 tmpfile):1:
*** Interpreter error recovered ***
```

```
root [2] if (invalidpf == 0) cout << "Invalid pointer!" << endl
Invalid pointer!
```
An object in memory: TF1*

What is the difference between an **object** and a **pointer to an object**?

**Main difference** (to me): **persistency**

```c
root [8] _TF1 of("myFunction","sin(x)/x",0,10)

... code...
```

Object lives in the memory **stack** → **memory gets freed** automatically when object goes **out of scope**
An object in memory: TF1*

What is the difference between an **object** and a **pointer to an object**?

**Main difference (to me): persistence**

Associated object lives in the memory **heap**
→ **memory does not get freed** automatically when it goes **out of scope**

```
root [6] TF1 * pf = new TF1("myFunction","sin(x)/x", 0, 10) | MemoryLeak.C
```

```
for (unsigned int i=0; i<100000; i++) {
    TF1 * pf = new TF1("myFunction", "sin(x)/x", 0,100);
}
```

```
<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10256</td>
<td>cate</td>
<td>20</td>
<td>0</td>
<td>3627m</td>
<td>3.5g</td>
<td>8868</td>
<td>S</td>
<td>56.6</td>
<td>45.3</td>
</tr>
</tbody>
</table>
```

2 GB of functions!

```
root [9] delete pf
```

especially in compiled code, every **new** needs a **delete to free the memory...**
otherwise **memory leak**

07/17/12      ROOT tutorial – A. Andreazza, C. Doglioni  18
Another object in ROOT: **TH1**

**Most famous object in ROOT: histogram (TH1...)**

Various types of histograms depending on type of content:
- e.g. **TH1D**: bins filled with doubles
- **TH1I**: bins filled with integers

- **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

Many properties and functionalities in common → **inheritance** from common class **TH1**
~ all functions of TH1 will be **inherited** by **derived classes**

Most ROOT objects inherit from **TNamed** class → all have a **SetName** function
Interlude: naming conventions

How does ROOT call its classes and functions?

- Class names start with capital T, e.g. TF1
- Class data members start with f, e.g. fXmin
- Names of non-class data types end with _t: e.g. Int_t
- Class methods start with _t: e.g. GetName()
- Global variable names start with _t: e.g. gPad
- Constant (or enumerator) names start with k: e.g. kTrue
- Words in names are capitalized: e.g. GetLineColor()
- Two subsequent capital letters are avoided: e.g.GetXaxis()
Objects in files
TFile: opening for reading

**TFile**: how to persistify ROOT's objects

Reading objects from a file

```c
root[0] TFile * myFile = TFile::Open("example.root", "READ")
```

- Returns a pointer to a Tfile
- Opening option: will not modify the file

```c
root[3] myFile->ls()
```

- Like unix's ls function, list the file content

```
TFile** example.root
TFile* example.root
KEY: TH1F cut_flow;1 cut_flow
KEY: TH1F averageIntPerXing;1 averageIntPerXing
KEY: TH1F delta_eta;1 delta_eta
KEY: TH1F delta_phi;1 delta_phi
KEY: TH1F mjj;1 mjj
```

A bit of pointer gymnastic: Get() returns a TObject, need to cast it to the correct object in order to access the pointer later

```c
root[6] TTree *myTree = (TTree*)myFile->Get("highestMjjEvents")
```
**TFile: writing objects**

**TFile**: how to persistify ROOT's objects

---

**Writing objects on a new file**

```c
root [0] TFile * myFile = TFile::Open("myNewFile.root", "RECREATE")
root [1] myFile.ls()
TFile** myNewFile.root
TFile* myNewFile.root

Opening option: will overwrite any existing file with the same name (alternative: UPDATE)

---

```c
root [2] TF1 * myFunction = new TF1("myFunction", "sin(x)/x", 0, 10)
root [3] myFunction->Write()
root [4] myFile->ls()
TFile** myNewFile.root
TFile* myNewFile.root
KEY: TF1 myFunction;1 sin(x)/x

---

```
root [5] myFunction->Write("theCopyOfMyFunction")
(Int_t)212
root [6]
root [6] myFile->ls()
TFile** myNewFile.root
TFile* myNewFile.root
KEY: TF1 myFunction;1 sin(x)/x
KEY: TF1 theCopyOfMyFunction;1 sin(x)/x

Simply write the function(object) to the file

---

Write the function to the file with a different name
TBrowser: ROOT's GUI

**TBrowser**: convenient way of accessing objects quickly

```
cate@catelenovalinux:~/Work/HASCO$ root -l example.root
root [0]
Attaching file example.root as _file0...
root [1] TBrowser b
```

List of filenames to be opened by ROOT and put in current directory

Time for a demo
Functions: TF1s
TF1 with parameters

A function can have **parameters** (e.g. floating parameters for fits...)

```c
{
TF1 * f1 = new TF1("f1",
"[0] / sqrt(2.0 * 3.1416) / [2] * exp(-(x-[1])*(x-[1])/2./[2]/[2]) + [3]",
0., 100.);
f1->SetParameter(0, 200.0);
f1->SetParameter(1, 50.0);
f1->SetParameter(2, 10);
f1->SetParameter(3, 10);
f1->Draw();
}
```

`GaussianWithOffset.C`
Let's draw a TF1 on a TCanvas

Like most objects in ROOT, functions can be drawn on a canvas

```
root [3] TF1 of("myFunction","sin(x)/x",0,10)
Info in <TCanvas::MakeDefCanvas>: created default TCanvas
with name c1
```
Let's draw a TF1 on a TCanvas

Like most objects in ROOT, functions can be **drawn** on a **canvas**

```c
root [3] TF1 of("myFunction","sin(x)/x",0,10)
Info in <TCanvas::MakeDefCanvas>: created default TCanvas
with name_c1
```

gPad: global variable pointing to current canvas

```c
root [7] gPad->GetName()
(const char* 0x24905d9)"c1"
```

gPad controls properties of current canvas, e.g. log scale

```c
root [8] gPad->SetLogy()
```
Let's draw a TF1 on a TCanvas

Like most objects in ROOT, functions can be drawn on a canvas. A TCanvas is an object too...

\begin{verbatim}
root [3] TF1 of("myFunction","sin(x)/x",0,10)
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name_c1
\end{verbatim}

...it can be divided in TPads

\begin{verbatim}
root [1] c.Divide(2,2)
root [3] c.cd(2)
(class TVirtualPad*)0x242a890
root [5] c.cd(1)
(class TVirtualPad*)0x242a510
root [7] c.cd(3)
(class TVirtualPad*)0x242ac30
root [8] of.Draw()
root [9] c.cd(4)
(class TVirtualPad*)0x242af0
root [10] of.Draw()
\end{verbatim}

...and saved as an image

\begin{verbatim}
root [6] c.SaveAs("myFunction.png")
Info in <TCanvas::Print>: png file myFunction.png has been created
\end{verbatim}
Graphical properties of TF1 can be changed

\begin{verbatim}
root [2] of.SetLineColor(kBlue+1)
\end{verbatim}

This will work for histograms too!
Formatting TF1s

Graphical properties of TF1 can be changed

\begin{verbatim}
root [4] of.SetLineStyle(2)
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{sinx_div_x.png}
\caption{Graph showing \( \frac{\sin(x)}{x} \)\).
\end{figure}

Some available line styles

This will work for histograms too!
Histograms: TH1/TH2s

ROOT Tutorial
HASCO school – 17/07/2012
1-dimensional histograms (1)

1-D histograms can be instantiated in various ways

With fixed bin size

```cpp
TH1D (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)
TH1F *myHistogram = new TH1F("myHistogram", "My histogram title", 100, 0, 4.4);
```

With variable bin size

```cpp
TH1D (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)
root [7] Double_t Bins[4] = {0, 2, 5, 8}
root [8] TH1F *myHistogram_varBinSize = new TH1F("myHistogram_varBinSize", "My histogram title", 3, Bins);
```

The number of bins is equal to the number of elements in the vector of bins minus one
Filling a histogram, getting information from a histogram

```c
{ TH1Basic.C

Double_t Bins[4] = {0,2,5,8};
TH1F *myHistogram_varBinSize = new TH1F("myHistogram_varBinSize", "My histogram title", 3, Bins);
myHistogram_varBinSize->Fill(1);

cout << "Bin 1 now has 
    " << myHistogram_varBinSize->GetBinContent(1) << " entries"
    << endl;

myHistogram_varBinSize->Print("all");
}
```

**Can also:**

- fill with weights:
  call `Fill(xEntry, weight)` and `TH1::SetSumw2` for calculating errors correctly
- Set entire bin content: call `setBinContent(iBin, binContent)`

Useful when no graphic session

```
root [0] .x TH1Basic.C
Bin 1 now has 1 entries
TH1.Print Name = myHistogram_varBinSize, Entries= 1, Total sum= 1
fSumw[0]=0, x=-1.33333
fSumw[1]=1, x=1
fSumw[2]=0, x=3.5
fSumw[3]=0, x=6.5
fSumw[4]=0, x=9.33333
```
1-dimensional histograms (3)

Useful information on **bin conventions**

**Overflows and underflows**
Every ROOT histogram has:
- **overflow bin** → where entries beyond the upper edge of the last bin go
- **Underflow bin** → where entries beyond the low edge of the first bin go

**Bin numbering conventions**
- \( \text{bin} = 0; \) underflow bin
- \( \text{bin} = 1; \) first bin with low-edge included
- \( \text{bin} = \text{nbins}; \) last bin with upper-edge excluded
- \( \text{bin} = \text{nbins}+1; \) overflow bin
Many options to **draw** a histogram (see THistPainter)

- `h1->Draw()`
- `h1->Draw("E")`
- `h1->Draw("TEXT")`
- `h1->Draw("L")`
The TBrowser editor

Let's click our way through **editing a histogram**...

Time for a demo
Don't forget the axis labels (1)

**TAxis**: class controlling x and y axes

Incidentally, this **always happens**
Don't forget the axis labels (2)

**TAxis**: class controlling x and y axes

```c
root[13] TAxis \* xAxis = myHistogram->GetXaxis()
root[14] xAxis->SetTitle("My units")
```

My histogram title

Axis title
Many 1-dimensional histograms (1)

How to plot **many histograms** at once?

\[ \int L \, dt \sim 4.7 \, \text{fb}^{-1} \]

**ATLAS** Preliminary

**TLatex**

**THStack**

**TPaveText**

**TPad**

- Data 2011 (\(\sqrt{s} = 7 \, \text{TeV}\))
- Background prediction
- Multi-jets (inc. \(t\bar{t} \rightarrow qq\))
- AlpGen \(t\bar{t} \rightarrow q\bar{q},ll\)
- AlpGen \(W \rightarrow (e,\mu,\tau)\nu\)
- AlpGen \(Z \rightarrow \nu\bar{\nu}\)
- AlpGen \(Z \rightarrow \tau\tau\)
- SUSY \(m_0=2960, m_{1/2}=240\)

Multi-jet control region

6 jets \(p_T > 55 \, \text{GeV}\)
Many 1-dimensional histograms (2)

How to plot many histograms at once, the **easy** way

```c
{
  TH1F *h1= new TH1F("h1", "h1", 100, -5,5);
  h1->FillRandom("gaus",10000);
  h1->Draw(""yor;)

  TH1F *h2= new TH1F("h2", "h2", 100, -5,5);
  h2->FillRandom("expo",10000);
  h2->SetLineColor(kRed);
  h2->Draw("same");
}
```

Draw histogram (or anything else) on the same canvas

Disadvantage: any formatting of axes, title etc will be tied to first histogram
THStack (1)

How to plot many histograms at once and **stack** them as well

More on random number generators later...

```cpp
TH1F *h1 = new TH1F("h1", "h1", 100, -5, 5);
h1->FillRandom("gaus", 10000);

TH1F *h2 = new TH1F("h2", "h2", 100, -5, 5);
h2->FillRandom("expo", 10000);
h2->SetLineColor(kRed);

THStack *hStack = new THStack();
hStack.Add(h1);
hStack.Add(h2);

hStack.Draw();
```

**Stacked histograms:**
Total bin content displayed = sum of bin contents of individual histograms
THStack (2)

How to plot many histograms at once and **stack** them as well

```c
TH1F *h1 = new TH1F("h1", "h1", 100, -5, 5);
h1->FillRandom("gaus",10000);

TH1F *h2 = new TH1F("h2", "h2", 100, -5, 5);
h2->FillRandom("expo",10000);
h2->SetLineColor(kRed);

THStack *hStack = new THStack();
hStack.Add(h1);
hStack.Add(h2); // Needed to 'create' the axis

hStack->Draw("A");
hStack->GetXaxis()->SetTitle("my units");
hStack->Draw("nostack");
```

```
07/17/12

ROOT tutorial – A. Andreazza, C. Doglioni
```

**nostack** option:

Equivalent to drawing with “*same*”

Advantage: control global drawing properties (axes etc) using THStack only
How to have e.g. a **data/MC inset** on the bottom of your plot

TPad: contained in a TCanvas, can contain other TPads

```cpp
//**Making the pads

//Set the coordinates of the current pad
//xLow, yLow, xHigh, yHigh
pad1 = new TPad("pad1","pad1",0.05,0.30,1,1);
pad2 = new TPad("pad2","pad2",0.05,0.05,1,0.30);
pad1->SetTopMargin(0.02);
pad1->SetLogy();
pad2->SetTopMargin(0.0);
pad1->SetBottomMargin(0.0);
pad2->SetBottomMargin(0.20);
pad1->Draw();
pad2->Draw();
pad1->cd();
```

Parameters: xLow, yLow, xHigh, yHigh
Coordinates are relative to the canvas: (x,y)=(0,0) is bottom left

If plots share the same x axis, cover axis for first plot

```cpp
//now draw the histograms
stack.Draw("nostack");
//Update() is used to make the
//canvas realise something has happened
cv->Update();
pad2->cd();
ratip->Draw();
cv->Update();
```

From now on, everything will be Draw()n on pad1

```
TPadExample.C
```

From now on, everything will be Draw()n on pad2
How to have e.g. a **data/MC inset** on the bottom of your plot

Final result (with some more formatting + a TLegend needed...)

![Plot diagram]
How to draw a **legend** for multiple histograms

```cpp
// constructor takes normalized coordinates within the pad
// with x=0, y=0 being the bottom left corner
TLegend *l = new TLegend(0.2, 0.6, 0.6, 0.8);
// let's make the legend background white
l.SetFillColor(kWhite);
// arguments: pointer to histogram, text, options: draw line (L)
l.AddEntry(h1, "The first histogram", "L");
l.AddEntry(h2, "The second histogram", "L");
l.Draw("same");
```

---

**TLegend**

07/17/12
2-dimensional histograms

2-D histogram can be instantiated in a similar way as 1-D ones, with one dimension more (there are also 3D histograms...)

With fixed bin size

```c
TH2 (const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup, Int_t nbinsy, Double_t ylow, Double_t yup)

root [0] TH2D * h2 = new TH2D("h2", "h2", 100, 0, 100, 200, 0, 200)
```

With variable bin size

```c
TH2 (const char* name, const char* title, Int_t nbinsx, const Double_t* xbins, Int_t nbinsy, const Double_t* ybins)
```

C arrays with low edges for each bin + high edge of last bin

```c
```

The number of bins is equal to the number of elements in the vector of bins minus one
2-dimensional histograms

Filling a 2-D histogram

```cpp
TH2D * h2 = new TH2D("h2", "h2", 1000, -5, 5, 1000, -5, 5);
// avoid underflow and overflow
for (unsigned int iBinX = 1; iBinX < h2.GetNbinsX()+1; iBinX++) {
    for (unsigned int iBinY = 1; iBinY < h2.GetNbinsY()+1; iBinY++) {
        // same syntax as TH1s, with one dimension more
        h2->SetBinContent(iBinX, iBinY, iBinX+iBinY);
    }
}
```

TH2Basic.C
2-dimensional histograms

Getting information from a 2-D histogram

```c
// finding the identifier of a bin
cout << "In the TH2 bin numbering scheme"
    << "x=4.5, y=4.5 is located in bin: "
    << h2->FindBin(4.5,4.5)
    << endl;

// this is particularly useful for 2D histograms
// as the function to find the bin content uses this
cout << "The bin content for the "
    << "x=4.5, y=4.5 bin is: "
    << h2->GetBinContent(h2->FindBin(4.5,4.5))
    << endl;
```

```
root [0] .x TH2Basic.C
In the TH2 bin numbering schemex=4.5, y=4.5 is located in bin: 953853
The bin content for the x=4.5, y=4.5 bin is: 1902
```
Pretty 2-dimensional histograms

How to set a new **palette** (credits to this website)

```c
void set_plot_style() {
    const Int_t NRGBs = 5;
    const Int_t NCont = 255;
    Double_t stops[NRGBs] = { 0.00, 0.34, 0.61, 0.84, 1.00 };
    Double_t red[NRGBs] = { 0.00, 0.00, 0.87, 1.00, 0.51 };
    Double_t green[NRGBs] = { 0.00, 0.81, 1.00, 0.20, 0.00 };
    Double_t blue[NRGBs] = { 0.51, 1.00, 0.12, 0.00, 0.00 };
    TColor::CreateGradientColorTable(NRGBs, stops, red, green, blue, NCont);
    gStyle->SetNumberContours(NCont);
}
```

```
root [0] .L SetPlotStyle.C
root [1] set_plot_style()
```
Graphs with errors
**TGraph**

**TGraph**: two arrays of points representing x and y coordinates

**TGraphErrors**: TGraph with symmetric errors on x and y points

**TGraphAsymmErrors**: TGraphErrors, with asymmetric errors

```c
{
  Double_t x[100], y[100];
  Int_t n = 20;
  for (Int_t i=0; i<n; i++) {
    x[i] = i*0.1;
    y[i] = exp(x[i]);
  }
  TGraph * g = new TGraph(n, x, y);
  //set marker style and size
  g->SetMarkerStyle(kFullCircle);
  g->SetMarkerSize(1.0);
  g->SetMarkerColor(kBlue);
  g->SetLineColor(kBlue);
  //in TGraph, need to draw Axis (A)
  //want to draw markers (P) and line (L)
  g->Draw("APL");
}
```

**TGraph.C**
TGraphAsymmErrors

**TGraph**: two arrays of points representing x and y coordinates

**TGraphErrors**: TGraph with symmetric errors on x and y points

**TGraphAsymmErrors**: TGraphErrors, with asymmetric errors

```c
TGraphAsymmErrors * g = new TGraphAsymmErrors();
//set a couple of points - index starts from 0
//parameters: point index, x coordinate, y coordinate
 g->SetPoint(0, 1.0, 2.0);
g->SetPoint(1, 2.0, 5.0);
//set the errors
//parameters: point index,
//x err down, x err up, y err down, y err up
 g->SetPointError(0, 0.25, 0.35, 1.0, 1.1);
g->SetPointError(1, 0.65, 0.5, 2.5, 2.0);

 g->SetMarkerStyle(kFullSquare);
g->SetMarkerSize(1.0);
g->SetMarkerColor(kRed);
g->SetLineColor(kRed);
//in TGraph, need to draw Axis (A)
//want to draw markers (P) and line (L)
g->Draw("AP");
```

TGraphAsymmErrors.C
Many TGraphs

How to plot many graphs at once?

ATLAS Preliminary

TLegend

TLatex

TMultiGraph
TMultigraph

How to plot many graphs at once?

// (snip) instantiate two graphs

// add pointers to graphs
mg->Add(gr1);
mg->Add(gr2);

// set title and range for both graphs at once
mg->Draw("A");
mg->GetXaxis()->SetLimits(0,2);
mg->GetXaxis()->SetTitle("My axis title");
mg->Draw("AP");

TMultigraph.C
Data storage and more: TTrees

ROOT Tutorial
HASCO school – 17/07/2012
What is a TTree?

TTree: made for **saving (and processing) data**

Simple idea:
it's like a table with
rows = events
columns = data fields

...more complex
(more functionalities)
than this: e.g.
- TTree can contain entire
  objects (branches → leaves)
- TTree can perform
  operations on itself
  (scanning, dumping to
  histogram, cuts)

<table>
<thead>
<tr>
<th>Run number</th>
<th>Event number</th>
<th>( \text{m}_{jj} ) (GeV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>203353</td>
<td>87535595</td>
<td>2669.6731</td>
</tr>
<tr>
<td>203353</td>
<td>74292059</td>
<td>2617.4563</td>
</tr>
<tr>
<td>203353</td>
<td>84096111</td>
<td>2452.7685</td>
</tr>
<tr>
<td>203353</td>
<td>74541499</td>
<td>2450.3027</td>
</tr>
<tr>
<td>203353</td>
<td>87499206</td>
<td>2399.0742</td>
</tr>
</tbody>
</table>
Preparing a TTree

Branching a TTree → creating **data fields** to save entries

```c
//construct the TTree
TTree * t = new TTree("myFirstTree", "myFirstTree");

//have some variables that will be read from the TTree
int runNumber = 0, eventNumber = 0;
double mjj = 0;

//let's branch the TTree
//arguments: branch name, address of variable, variable name and type
//see http://root.cern.ch/root/html/TTree.html#TTree:Branch
//t->Branch("runNumber", &runNumber, "runNumber/I");
t->Branch("eventNumber", &eventNumber, "eventNumber/I");
t->Branch("mjj", &mjj, "mjj/D");
```

This will associate the variables to the tree so it will read from the right locations in memory.
Filling a TTree

Filling a TTree → **inserting entries** in data fields

```c
// now let's loop on some toy events
for (unsigned int iEvent = 0; iEvent<10; iEvent++) {
    runNumber = 150000;
    eventNumber = iEvent;
    // fictitious dijet mass...
    mjj = double(runNumber*iEvent)/1000.;
    // let the TTree pick up the variables for each event
    t->Fill();
}
```

// see what the TTree contains
```
t->Print();
```

See the **TTree class doc** for more ways to fill a TTree...

---

**Unterhansli der Genève**
Reading a TTree: Scan

Simple by-eye **inspection** of TTree entries

Ttree::Scan()

Without any arguments, Scan() will display all entries and all branches sequentially

```
root [3] myFirstTree->Scan()

<table>
<thead>
<tr>
<th>Row</th>
<th>runNumber</th>
<th>eventNum</th>
<th>mjj</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>150000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>150000</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>150000</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>150000</td>
<td>3</td>
<td>450</td>
</tr>
<tr>
<td>4</td>
<td>150000</td>
<td>4</td>
<td>600</td>
</tr>
<tr>
<td>5</td>
<td>150000</td>
<td>5</td>
<td>750</td>
</tr>
<tr>
<td>6</td>
<td>150000</td>
<td>6</td>
<td>900</td>
</tr>
<tr>
<td>7</td>
<td>150000</td>
<td>7</td>
<td>1050</td>
</tr>
<tr>
<td>8</td>
<td>150000</td>
<td>8</td>
<td>1200</td>
</tr>
<tr>
<td>9</td>
<td>150000</td>
<td>9</td>
<td>1350</td>
</tr>
</tbody>
</table>
```
Reading a TTree: Scan

Simple by-eye **inspection** of TTree entries

Ttree::Scan("branchName")

You can Scan() single / multiple branches (first argument of the function needs to be the branch name)

```
root [4] myFirstTree->Scan("mjj")
*************************************************************************
* Row  *  mjj *
*************************************************************************
* 0    *  0   *
* 1    *  150 *
* 2    *  300 *
* 3    *  450 *
* 4    *  600 *
* 5    *  750 *
* 6    *  900 *
* 7    * 1050 *
* 8    * 1200 *
* 9    * 1350 *
*************************************************************************
```
Cuts on a TTree with TTree::Scan

Simple by-eye inspection of TTree entries + apply cuts

Ttree::Scan("","branchName>cut")

You can apply cuts using Scan() and the syntax of TFormulas

e.g.

\[ [0]*\sin(x) + [1]*\exp(-[2]*x) \]

\[ 2*pi*\sqrt{x/y} \]

root [5] myFirstTree->Scan("","mjj>1000")

<table>
<thead>
<tr>
<th>Row</th>
<th>runNumber</th>
<th>eventNumber</th>
<th>mjj</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>150000</td>
<td>7</td>
<td>1050</td>
</tr>
<tr>
<td>8</td>
<td>150000</td>
<td>8</td>
<td>1200</td>
</tr>
<tr>
<td>9</td>
<td>150000</td>
<td>9</td>
<td>1350</td>
</tr>
</tbody>
</table>
TTree branches can easily be **drawn** on 1D histograms

`TTree::Draw("branchName", "cuts","", "histogram painting options")`

```
root [16] myFirstTree->Draw("mjj")
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```

...not very physical...
TTree branches can easily be drawn on 2D (or 3D) histograms

```cpp
TTree::Draw("branchName1:branchName2", "...")
```

```
root [18] myFirstTree->Draw("mjj:eventNumber","", "COLZ")
```

...still not very physical...
The result of Draw() can be **saved** on a custom histogram

\[ \text{TTree::Draw("branchName", "branchName">h1(TH1 nBinsX, xLow, xHigh)")} \]

```
root [29] myFirstTree->Draw("mjj">h1(5,0,1500", ",", "COLZ")
(Long64_t)10
root [30] h1->Draw("E")
```
Inspecting TTree with TTreeViewer

Scan(), Draw() and more by clicking on branches

Time for a demo
A TChain is a TTree (inheritance...) - advantage: **split** over **many files**

...after having generated two separate large TTrees...

```
root [0] TChain * c = new TChain("myTree")
root [1] c->Print()
root [2] c->Add("ChainExample_*.root")
root [3] c->Print()
```

Chained TTrees must have the same branches and the same name, given to the TChain.

Wildcards work to give files containing TTrees to TChain.
Tomorrow...

- Reading TTrees efficiently: TSelector
- Random number generation
- Fitting in ROOT and more
- pyROOT

(things will get more interesting for the experienced ones among you!)

HASCO school – 18/07/2012