Introduction to ROOT
Practical Part

Jan Fiete Grosse-Oetringhaus, CERN PH/ALICE
Fons Rademakers, CERN PH/SFT

Summer Student Lectures 2009
6. – 7. July

Content

- Practical introduction to the ROOT framework
  - Starting ROOT
  - Macros
  - Histograms
  - Trees
  - Creating ROOT classes
  - Basics of debugging
  - ROOT prompt
  - Functions
  - Files
  - TBrowser
- Nomenclature
  - **Blue: you type it**
  - **Red: you get it**

Example macros and histograms are in http://www.cern.ch/jgrosseo/permanent/summerschool2009.tgz

ROOT prompt

- Starting ROOT

  $ root **$ root -l** (without splash screen)

- The ROOT prompt

  root [ ] 2+3 **root [ ] int i = 42**

  root [ ] log(5) **root [ ] printf("%d\n", i)**

- Command history
  - Scan through with **arrow keys** ↑↓
  - Search with **CTRL-R** (like in bash)
- Online help

  root [ ] new TF1(<TAB>

  TF1 TF1()

  TF1 TF1(const char* name, const char* formula, Double_t xmin = 0, Double_t xmax = 1)

  …

ROOT Prompt (2)

- Typing multi-line commands

  root [ ] for (i=0; i<3; i++) printf("%d\n", i)

  or

  root [ ] for (i=0; i<3; i++) {

  end with '}', '@':abort > printf("%d\n", i);

  end with '}', '@':abort > }

- Aborting wrong input

  root [ ] printf("%d\n, i)

  end with ';', '@':abort > @

Don't panic!
Don't press CTRL-C!
Just type @

Macros

- Combine lines of codes in macros
- Unnamed macro
  - No parameters
  - For example: macro1.C

```
{
   for (Int_t i=0; i<3; i++)
     printf("%d\n", i);
}
```

Data types in ROOT
Int_t (4 Bytes)
Long64_t (8 Bytes)
…
to achieve platform-independency

- Executing macros

```
root [ ] .x macro1.C
$ root –l macro1.C
$ root –l –b macro1.C (batch mode → no graphics)
$ root –l –q macro1.C (quit after execution)
```

Macros (2)

- Named macro
  - May have parameters
  - For example macro2.C:

```
void macro2(Int_t max = 10)
{
   for (Int_t i=0; i<max; i++)
      printf("%d\n", i);
}
```

- Running named macro

```
root [ ] .x macro2.C(12)
```

- Loading macros

```
root [ ] .L macro2.C
root [ ] macro2(12)
```

- Prompt vs. Macros
  - Use the prompt to test single lines while developing your code
  - Put code that is to be reused in macros

Don't forget to change the function name after renaming a macro

Plots for Papers

It is very useful to have all the code that creates a plot in one macro. Do not create "final" plots using the prompt or the mouse (you'll be doing it again and again).

Functions

- The class TF1 allows to draw functions

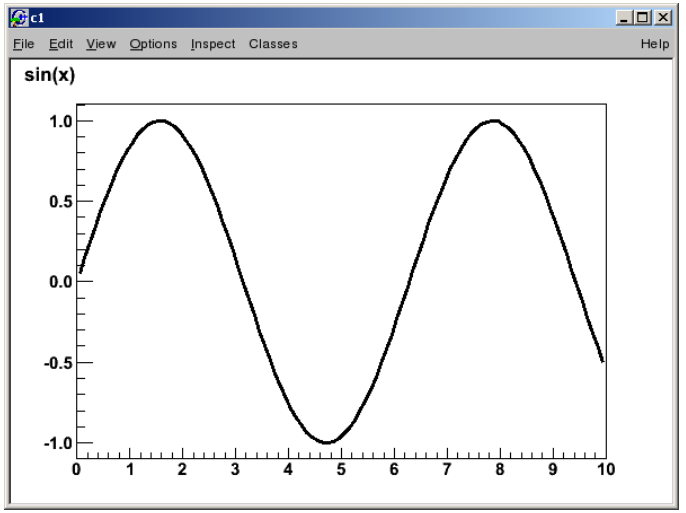
  root [ ] f = new TF1("func", "sin(x)", 0, 10)

  – **"func"** is a (unique) name

  – **"sin(x)"** is the formula

  – **0, 10** is the x-range for the function

  root [ ] f->Draw()



- The style of the function can be changed on the command line or with the context menu (→ right click)

  root [ ] f->SetLineColor(kRed)

- The class TF2(3) is for 2(3)-dimensional functions

Canvas

Histograms

- Contain binned data – probably the most important class in ROOT for the physicist
- Create a TH1F (= one dimensional, float precision)

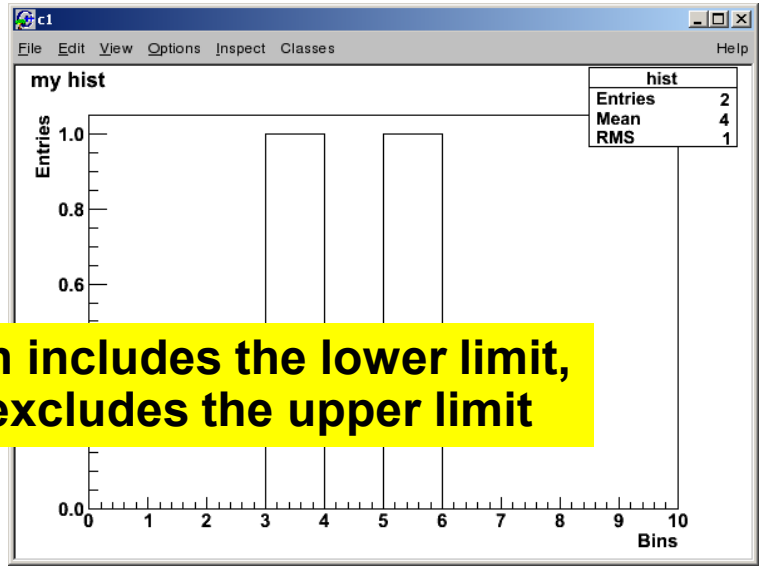
```
root [ ] h = new TH1F("hist", "my hist;Bins;Entries", 10, 0, 10)
```

  - **"hist"** is a (unique) name
  - **"my hist;Bins;Entries"** are the title and the x and y labels
  - **10** is the number of bins
  - **0, 10** are the limits on the x axis. Thus the first bin is from 0 to 1, the second from 1 to 2, etc.
- **Fill the histogram**

```
root [ ] h->Fill(3.5)
root [ ] h->Fill(5.5)
```

- **Draw the histogram**

```
root [ ] h->Draw()
```



A bin includes the lower limit, but excludes the upper limit

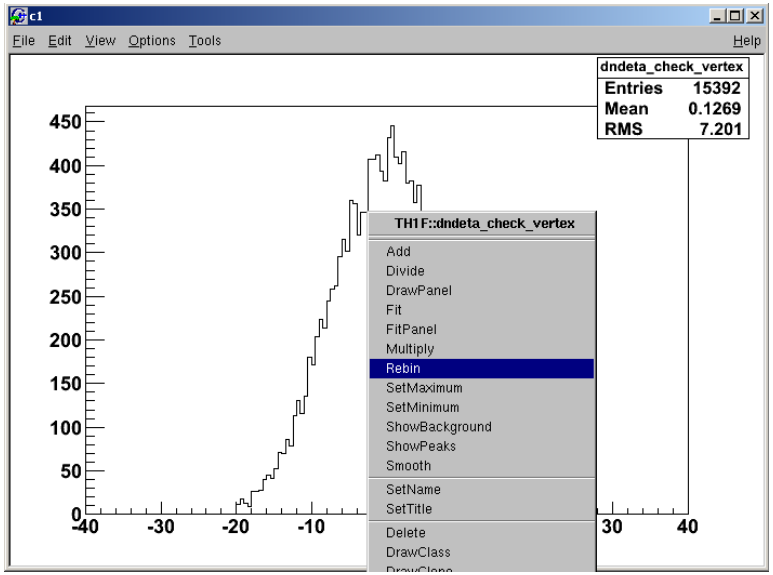
Histograms (2)

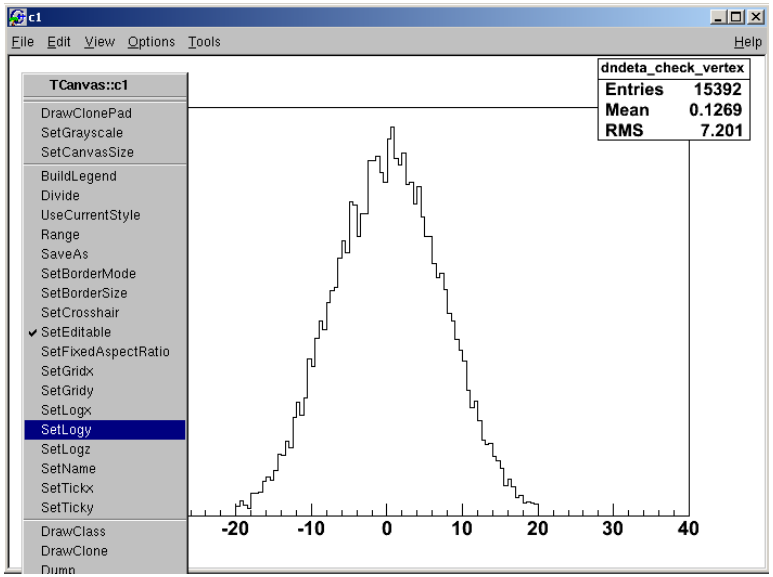
- Rebinning
  root [ ] h->Rebin(2)
- Change ranges
  - with the mouse
  - with the context menu
  - command line

  root [ ] h->GetXaxis()->SetRangeUser(2, 5)
- Log-view
  - right-click in the white area at the side of the canvas and select SetLogx (SetLogy)
  - command line

  root [ ] gPad->SetLogy()

NB: example histogram in file hist.root

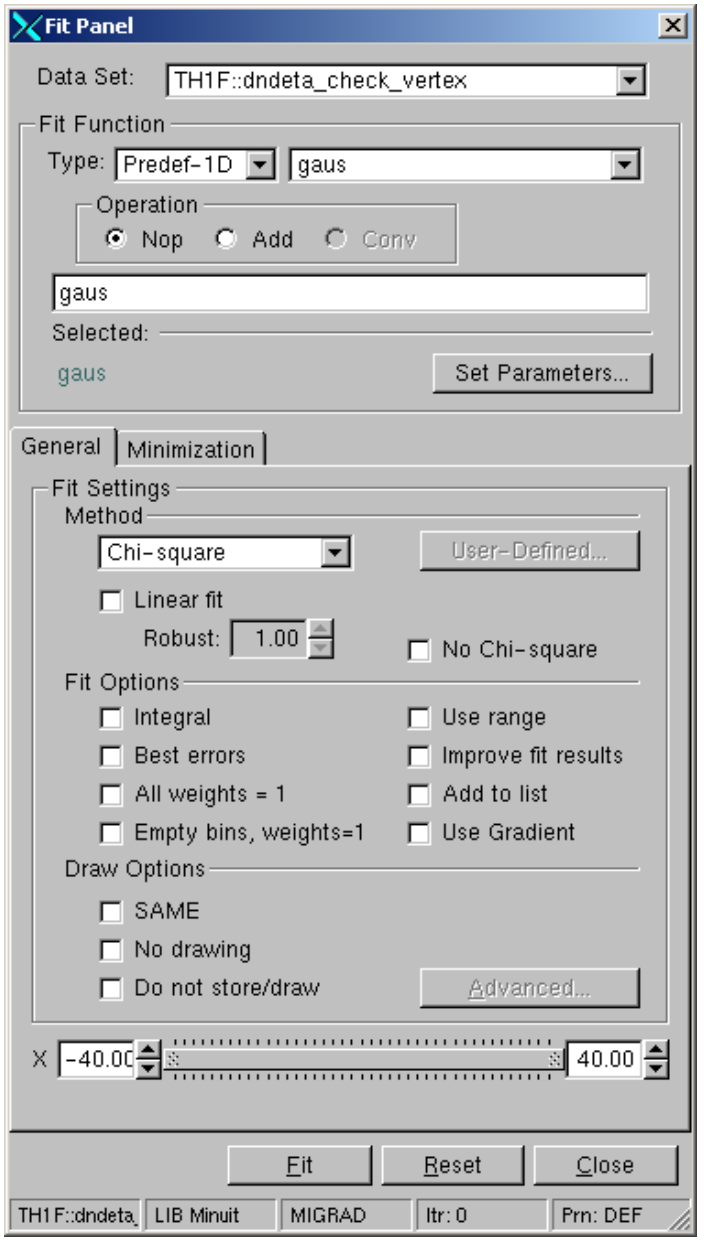




Fitting Histograms

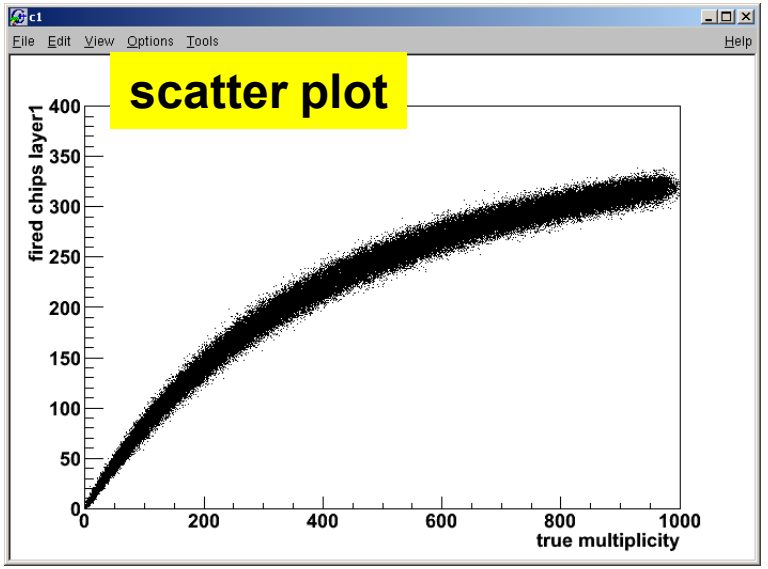
- Interactive
  - Right click on the histogram and choose "fit panel"
  - Select function and click fit
  - Fit parameters
    - are printed in command line
    - in the canvas: options - fit parameters
- Command line

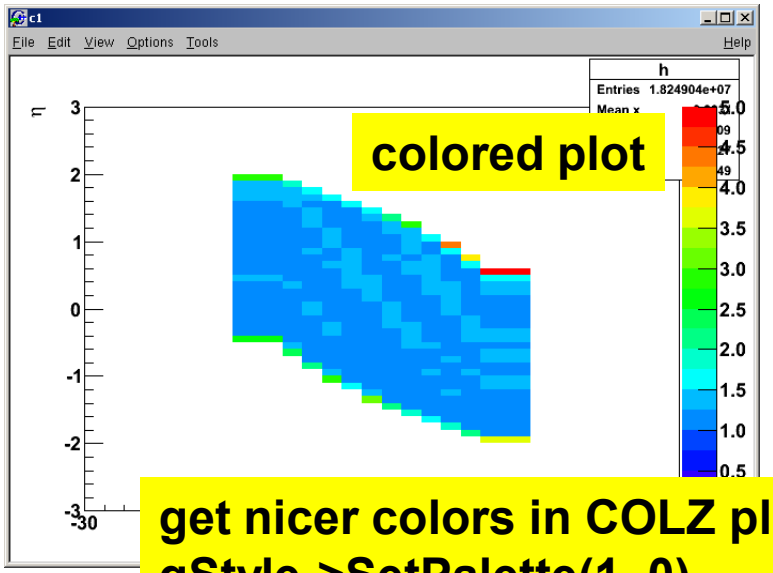
  root [ ] h->Fit("gaus")
  - Other predefined functions polN (N = 0..9), expo, landau



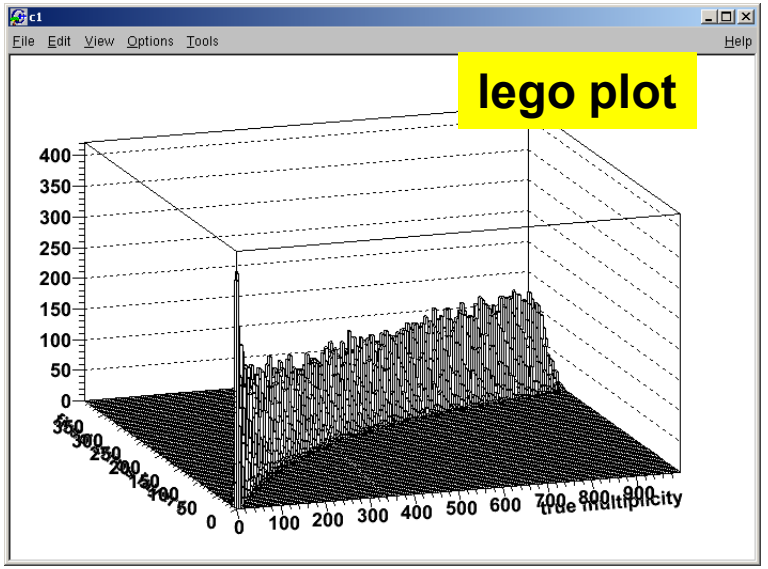
2D Histograms

```
root [ ] h->Draw()
root [ ] h->Draw("LEGO")
root [ ] h2->Draw("COLZ")
```





get nicer colors in COLZ plots by gStyle->SetPalette(1, 0)



NB: h and h2 are in file hist2.root

Files

- The class TFile allows to store any ROOT object on the disk
- Create a histogram like before with
  h = new TH1F("hist", "my hist;…", 10, 0, 10)
  etc.

"hist" will be the name in the file

- Open a file for writing

  root [ ] file = TFile::Open("file.root", "RECREATE")
- Write an object into the file

  root [ ] h->Write()
- Close the file

  root [ ] file->Close()

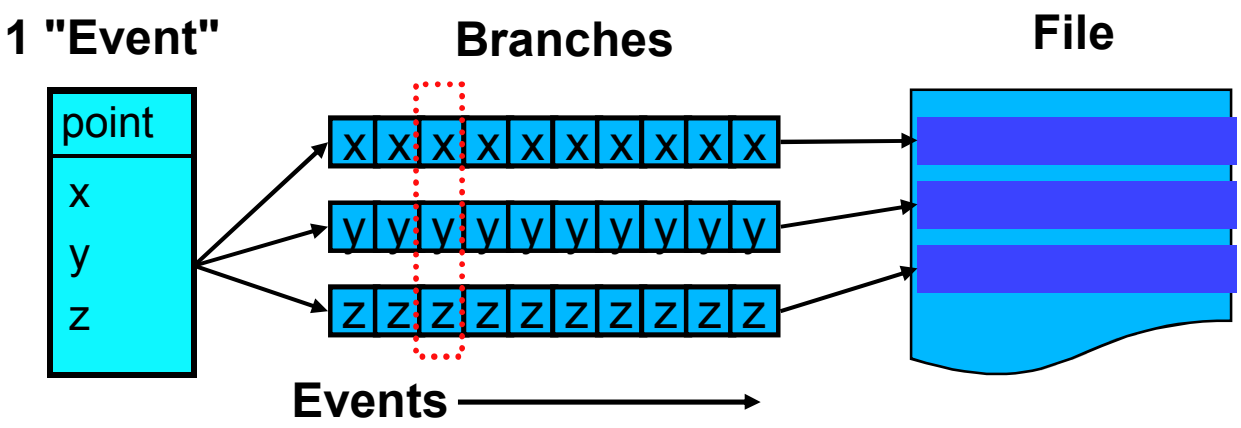
Files (2)

- Open the file for reading
  root [ ] file = TFile::Open("file.root")
- Read the object from the file
  root [ ] hist->Draw()
  (only works on the command line!)
- In a macro read the object with
  TH1F* h = 0;
  file->GetObject("hist", h);
- What else is in the file?
  root [ ] .ls
- Open a file when starting root
  $ root file.root
  – Access it with the **_file0** or **gFile** pointer

Object ownership
After reading an object from a file don't close it! Otherwise your object is not in memory anymore

Trees

- The class TTree is the main container for data storage
  - It can store any class and basic types (e.g. Float_t)
  - When reading a tree, certain branches can be switched off → speed up of analysis when not all data is needed
- First example: the class TNtuple which is derived from TTree and contains only Float_t



TNtuple

- Create a TNtuple

  root [ ] ntuple = new TNtuple("ntuple", "title", "x:y:z")

  – **"ntuple"** and **"title"** are the name and the title of the object

  – **"x:y:z"** reserves three variables named x, y, and z
- Fill it

  root [ ] ntuple->Fill(1, 1, 1)
- Get the contents

  root [ ] ntuple->GetEntries() number of entries

  root [ ] ntuple->GetEntry(0) for the first entry

  root [ ] ntuple->GetArgs()[1] for y (0 for x, and 2 for z)

  – These could be used in a loop to process all entries
- List the content

  root [ ] ntuple->Scan()

NB: The file ntuple.C produces this TNtuple with some random entries

TNtuple (2)

- Draw a histogram of the content
  - to draw only x

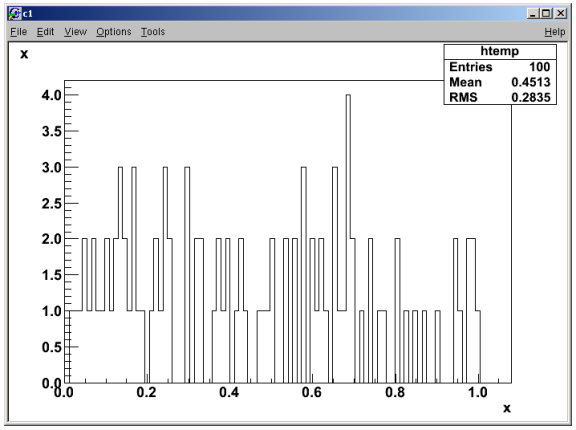
```
root [ ] ntuple->Draw("x")
```

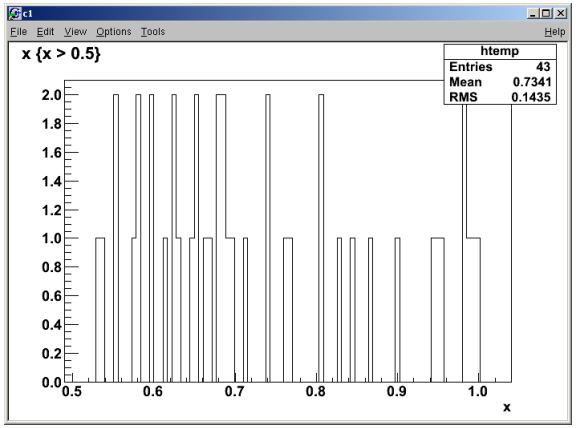
  - draw all x that fulfill x > 0.5

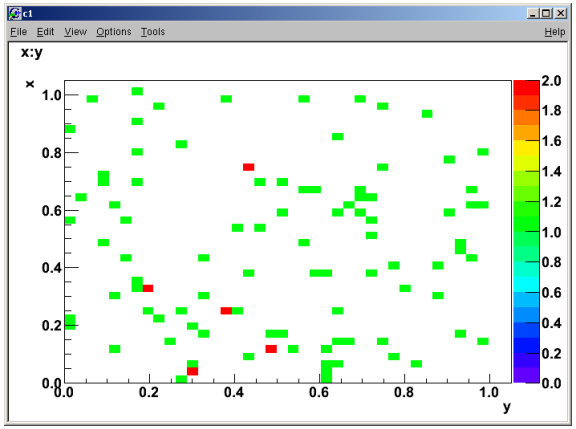
```
root [ ] ntuple->Draw("x", "x > 0.5")
```

  - to draw x vs. y in a 2d histogram

```
root [ ] ntuple->Draw("x:y", "", "COLZ")
```







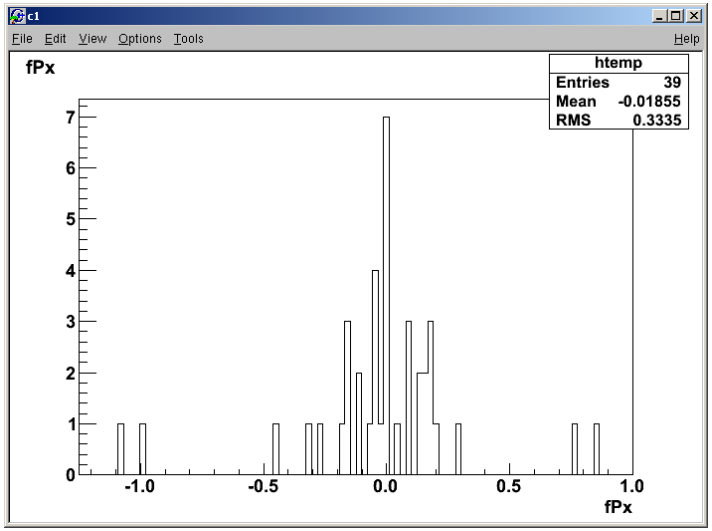
TNtuple (or TTree) with many entries may not fit in memory
→ open a file before creating it

Trees (2)

- Accessing a more complex tree that contains classes
  - Members are accessible even without the proper class library
  - Might not work in all LHC experiments' frameworks
- Example: tree.root (containing kinematics from ALICE)

```
$ root tree.root
root [ ] tree->Draw("fPx")
root [ ] tree->Draw("fPx", "fPx < 0")
root [ ] tree->Draw("fPx",
    "abs(fPdgCode) == 211")
```

PDG code of pions



- From where do you know fPx, fPdgCode?
  - The tree contains TParticles
  - Check ROOT documentation: http://root.cern.ch/root/html/TParticle

Trees (3)

- Connecting a class with the tree

  root [ ] TParticle* particle = 0

  root [ ] tree->SetBranchAddress("Particles", &particle)

- Read an entry

  root [ ] tree->GetEntry(0)

  root [ ] particle->Print()

  root [ ] tree->GetEntry(1)

  root [ ] particle->Print()

  – These commands could be used in a loop to process all particles

The content of the TParticle instance is replaced with the current entry of the tree

```
root [5] particle->Print()
TParticle: pi0                  p: -0.036864 -0.0
```

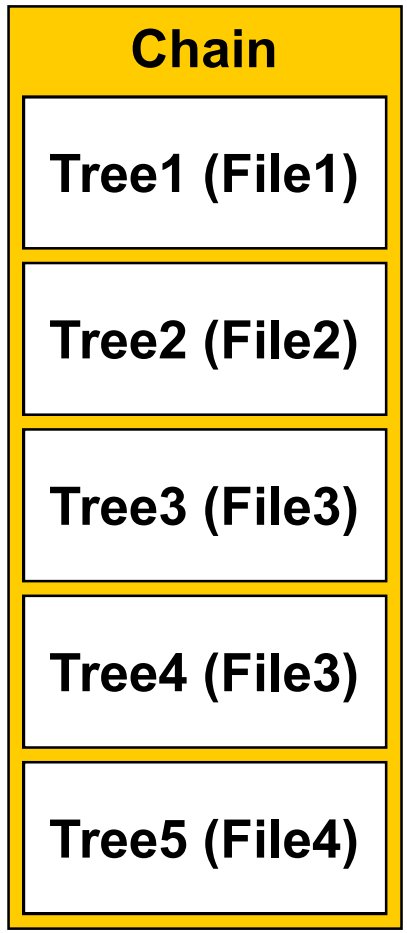
TChain

- A chain is a list of trees (in several files)
- Normal TTree functions can be used

```
root [ ] chain = new TChain("tree")
root [ ] chain->Add("tree.root")
root [ ] chain->Add("tree2.root")
root [ ] chain->Draw("fPx")
```

  - The Draw function iterates over both trees

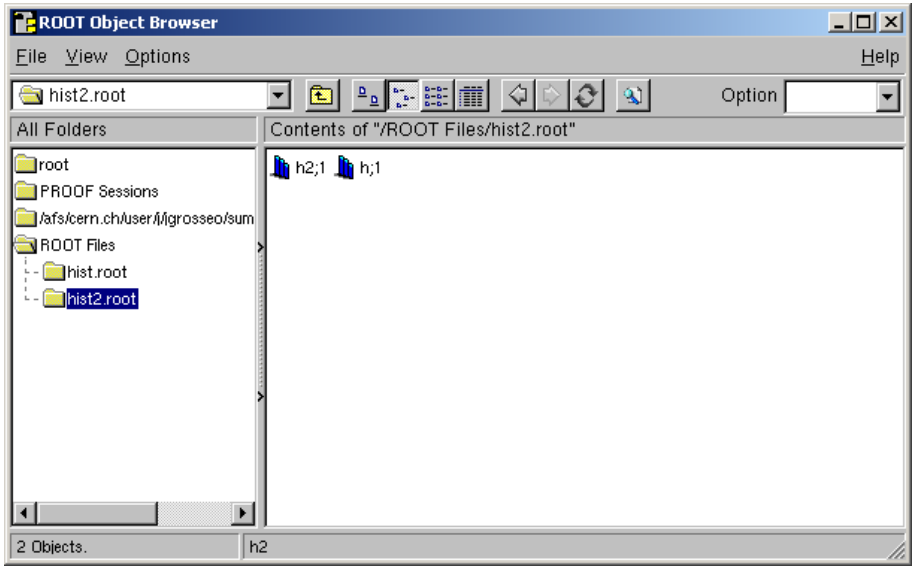
Name of the tree in the files tree.root and tree2.root

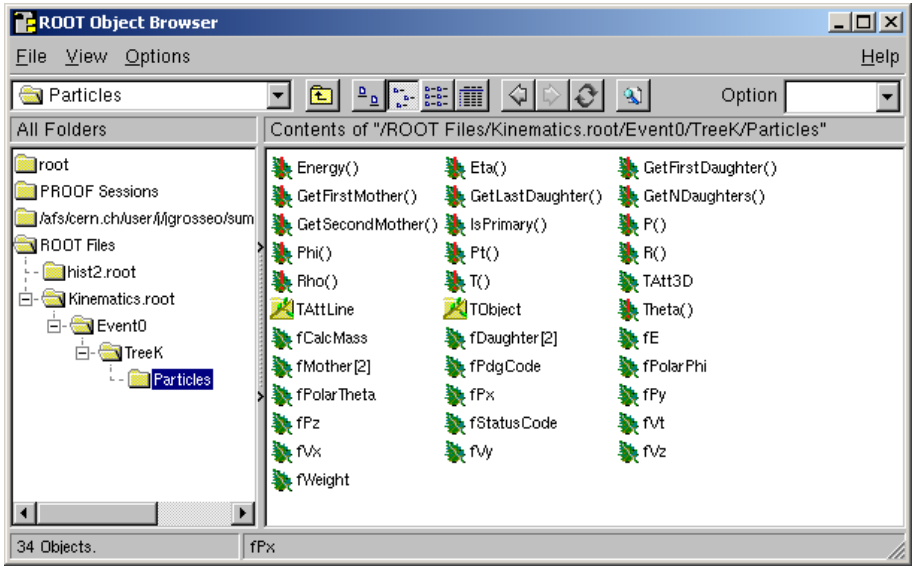


TBrowser

- The TBrowser can be used
  - to open files
  - navigate in them
  - to look at TTrees
- Starting a TBrowser

  root [ ] new TBrowser
- Open a file
- Navigate through the file
- Draw a histogram
- Change the standard style
  - Drop down menu in the top right corner
- Access a tree
- Plot a member





Creating Classes

NB: This code is in TSummerStudent.C

- Any C++ class can be used with ROOT
- Classes derived from TObject can be used directly with many other ROOT classes (e.g. TList, TObjArray)

```
#include <TObject.h>
#include <TString.h>
class TSummerStudent : public TObject {
  private:
    TString fFirstName;
    Int_t fAge;
  public:
    const char* GetFirstName() const { return fFirstName; }
    Int_t GetAge() const { return fAge; }
    TSummerStudent(const char* firstname, Int_t age)
              : fFirstName(firstname), fAge (age) { }
    virtual ~TSummerStudent () {}
    ClassDef(TSummerStudent, 1)
};
```

TString to store strings

version number of class layout

when you add or change a member, increase the version number!

0 = not streamable

This macro adds some ROOT magic by including a dictionary created by CINT

Creating Classes (2)

- Include the class in ROOT

  root [ ] .L TSummerStudent.C+g

  "g" adds debug symbols

- Use it

  root [ ] s = new TSummerStudent("Matthew", 24)

  root [ ] s->GetFirstName()

- The object can be written in a file, send over the network etc.
- You can show the content of any ROOT class

  root [ ] s->Dump()

Understanding Errors

Distinguish

- Compiling error
  - Syntax errors
  - Missing declarations
- Error while loading the library "dlopen error"
  - Missing implementation of a declared function (much more subtle)
  - Might even be in parent class

- Read error messages from top. Many other (weird) messages follow. Examples:
  - missing }
  - Missing include file
- Problems with macros? → Compile them to find errors

  root [ ] .L macro2.C+

Basics of Debugging

- When there is a segmentation violation, you get the stack trace
  - It tells you where the crash happens
  - Find the relevant piece in the stack trace
    - Start from top
    - Few lines after "signal handler called"
    - Most of the times it makes only sense to look at lines that reference to your own code
  - Compile with debug ("g") to see line numbers

Stack Trace

```
 *** Break *** segmentation violation
Using host libthread_db library "/lib/tls/libthread_db.so.1".
Attaching to program: /proc/23893/exe, process 23893
[Thread debugging using libthread_db enabled]
[New Thread -1208858944 (LWP 23893)]
0x0077c7a2 in _dl_sysinfo_int80 () from /lib/ld-linux.so.2
#1  0x002b34b3 in __waitpid_nocancel () from /lib/tls/libc.so.6
#2  0x0025c779 in do_system () from /lib/tls/libc.so.6
#3  0x0022198d in system () from /lib/tls/libpthread.so.0
#5  0x009db83e in TUnixSystem::StackTrace (this=0x9daa440) at core/unix/src/TUnixSystem.cxx:2132
#6  0x009d962d in TUnixSystem::DispatchSignals (this=0x9daa440, sig=kSigSegmentationViolation) at core/unix/src/TUnixSyst
#7  0x009d745d in SigHandler (sig=kSigSegmentationViolation) at core/unix/src/TUnixSystem.cxx:350
#8  0x009de7aa in sighandler (sig=11) at core/unix/src/TUnixSystem.cxx:3368
#9  <signal handler called>
#10 0x003effd8 in TSummerStudent::SomeFunction (this=0xa0154b0) at /home/shuttle/Fiete/./TSummerStudent_debug.C:14
#11 0x003ee355 in G__TSummerStudent_debug_C_ACLiC_dict_2564_0_3 (result7=0xbffe0420, funcname=0xa0153f8 "\001", libp=0xb
    at /home/shuttle/Fiete/./TSummerStudent_debug_C_ACLiC_dict.cxx:186
#12 0x00ed8dbf in Cint::G__ExceptionWrapper (funcp=0x3ee32e <G__TSummerStudent_debug_C_ACLiC_dict_2564_0_3>, result7=0xb
    hash=0) at cint/cint/src/Api.cxx:384
#13 0x00f81786 in G__execute_call (result7=0xbffe0420, libp=0xbffda5a0, ifunc=0xa0153f8, ifn=0) at cint/cint/src/newlink
#14 0x00f81ea6 in G__call_cppfunc (result7=0xbffe0420, libp=0xbffda5a0, ifunc=0xa0153f8, ifn=0) at cint/cint/src/newlink
#15 0x00f6295a in G__interpret_func (result7=0xbffe0420, funcname=0xbffe0020 "SomeFunction", libp=0xbffda5a0, hash=1242,
    at cint/cint/src/ifunc.cxx:5277
#16 0x00f4907c in G__getfunction (item=0xbffe3263 "SomeFunction()", known3=0xbffe267c, memfunc_flag=1) at cint/cint/src/
#17 0x0103b145 in G__getstructmem (store_var_type=112, varname=0xbffe0670 "\357/5", membername=0xbffe3263 "SomeFunction()",
    varglobal=0x10d9ea0, objptr=2) at cint/cint/src/var.cxx:6691
#18 0x0102f234 in G__getvariable (item=0xbffe3260 "s->SomeFunction()", known=0xbffe267c, varglobal=0x10d9ea0, varlocal=0
#19 0x00f3ccc9 in G__getitem (item=0xbffe3260 "s->SomeFunction()") at cint/cint/src/expr.cxx:1884
#20 0x00f3b338 in G__getexpr (expression=0xbffe4b50 "s->SomeFunction()") at cint/cint/src/expr.cxx:1470
```

Basics of Debugging (2)

- Reproduce the problem in the debugger
- Most linux systems include gdb (GNU debugger)
- **$ gdb root.exe** (gdb root does not work)
  - Parameter to root have to be passed with
    $ gdb --args root.exe macro.C
  - On the gdb prompt, start the program: **(gdb) run**
- You will see the line where the crash happened
- Basic commands
  - **bt** = backtrace, gives the stack
  - **up**, **down** to navigate in the stack → go to the first frame with your code
  - **p <var>** → prints the variable <var> (of your code, e.g. particle)
  - **quit** to exit

Resources

- Main ROOT page
  - http://root.cern.ch
- Class Reference Guide
  - http://root.cern.ch/root/html
- C++ tutorial
  - http://www.cplusplus.com/doc/tutorial/
- Hands-on tutorials (especially the last one)
  - http://root.cern.ch/drupal/content/tutorials-and-courses