Introduction to ROOT

An object oriented HEP analysis framework.

Akiya Miyamoto, KEK

Bases on ROOT lecture by Suzanne Panacek (FNAL)
http://www-root.fnal.gov/root

The ROOT Team

Plan of this lecture

1. Overview of the ROOT Framework
2. GUI basics
3. Command line basics
4. Root Commands and CINT
5. Examples of Histograms, Functions and Fitting
6. How to add your Own Class

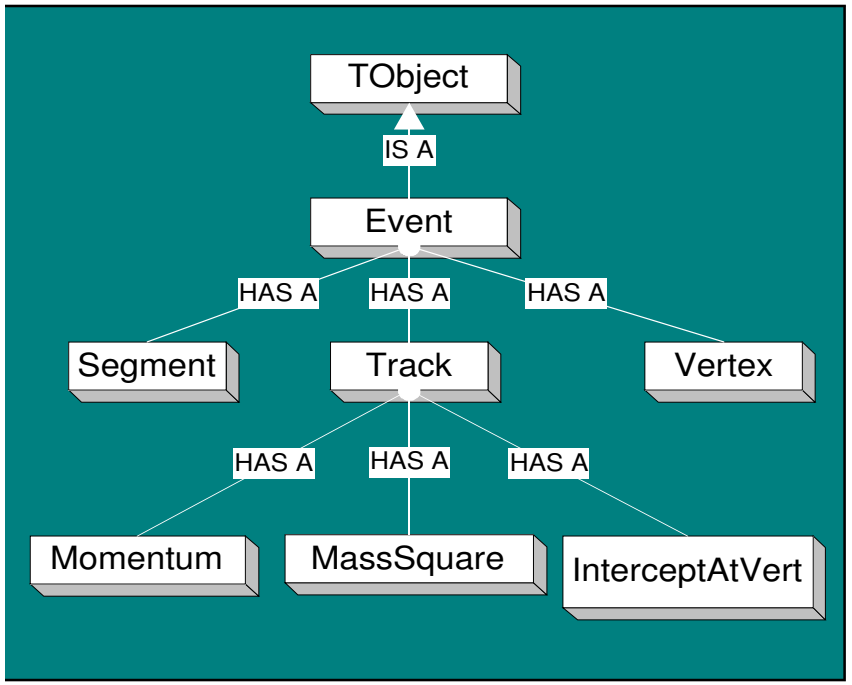
ROOT Overview

- What about PAW
- Concepts: Object Oriented Design, Frameworks
- Services and Utilities
- Libraries
- Physical Organization

Object Oriented Concepts

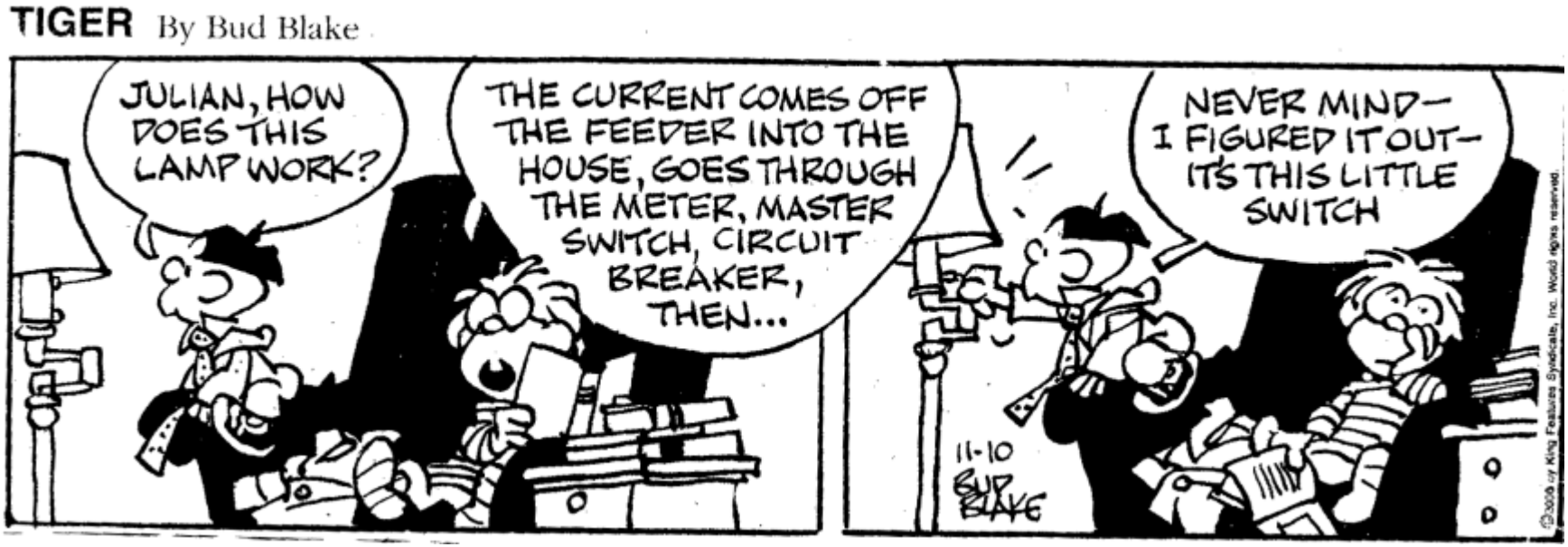
- Class: the description of a “thing” in the system
- Object: instance of a class
- Methods: functions for a class

- Members: a “has a” relationship to the class.
- Inheritance: an “is a” relationship to the class.



A Framework

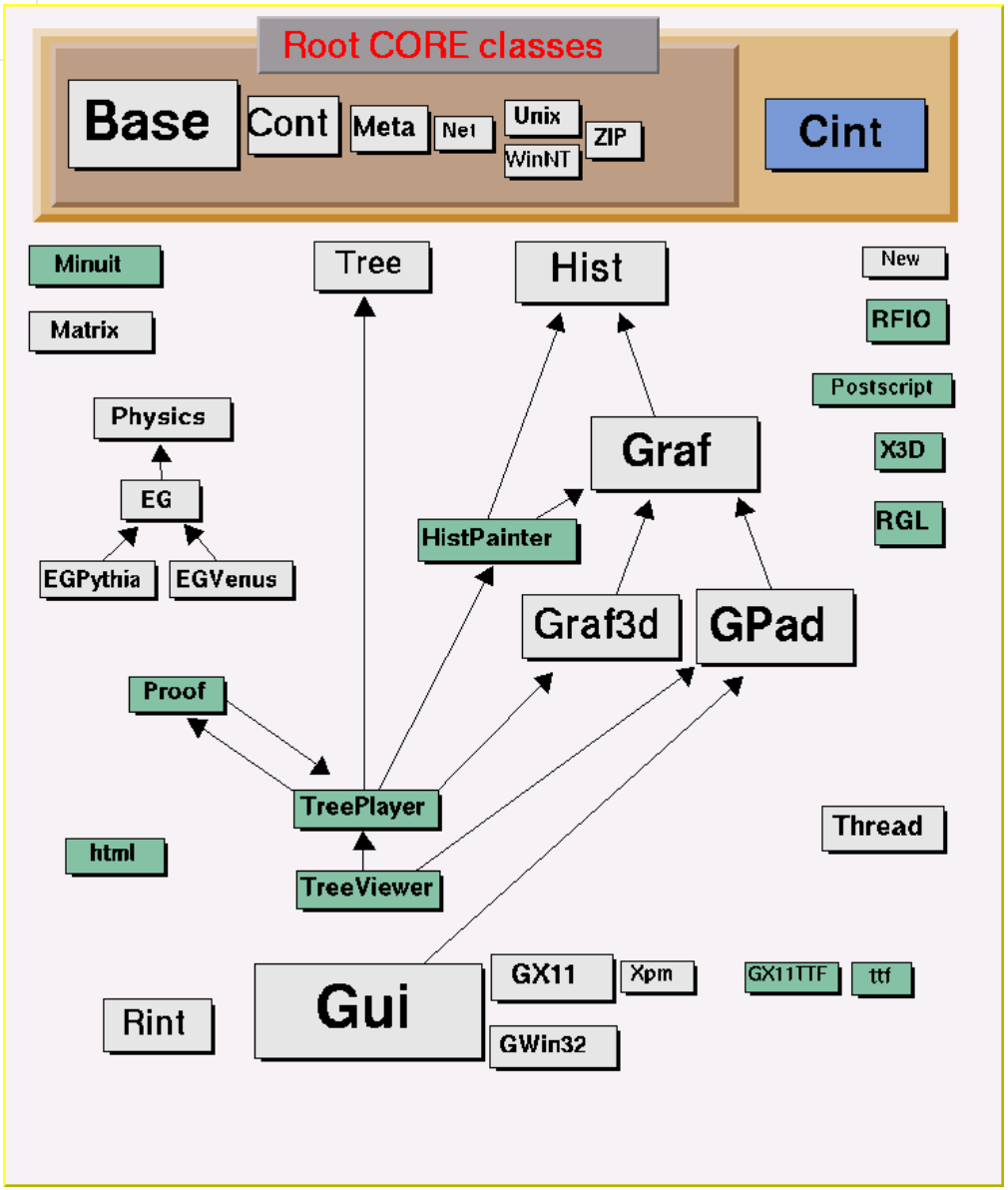
provides utilities and services.



ROOT's Services/Utilities

- Histogramming and Fitting
- Graphics (2D, 3D)
- I/O to file or socket: specialized for histograms, Ntuples (Trees)
- Collection Classes and Run Time Type Identification
- User Interface
  - GUI: Browsers, Panels, Tree Viewer
  - Command Line interface: C++ interpreter CINT
  - Script Processor (C++ compiled ⇔ C++ interpreted)

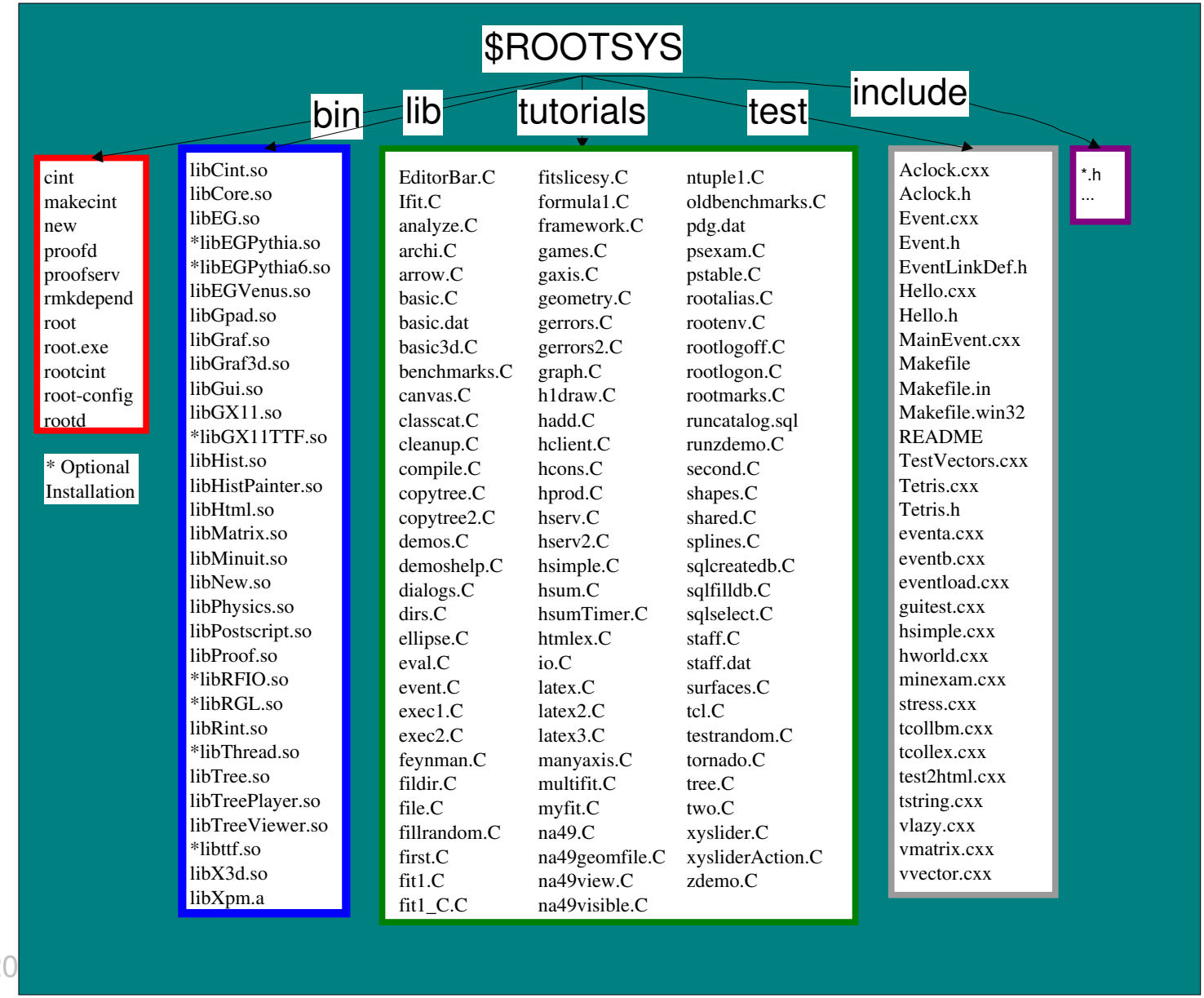
RO
- Hist
- Gra
- I/O
Ntup
- Coll
Iden
- Use
  - G
  - C
  - S



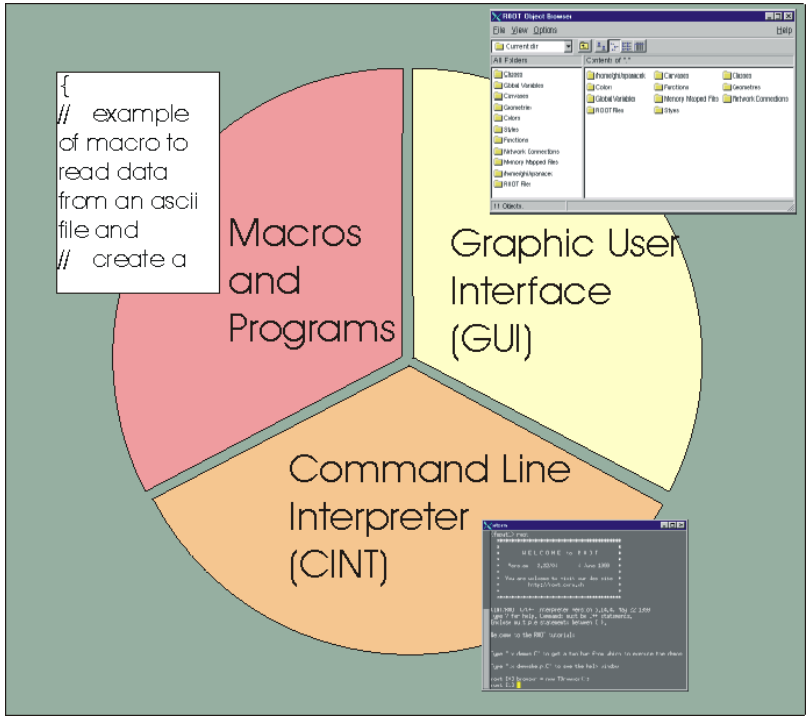
The Libraries

- Over 350 classes
- Core
- CINT
- Libraries loaded at startup: Hist, Tree …
- Libraries loaded when needed: HistPainter, TreePlayer,…
- Special purpose libraries: EG, Physics…

The Framework Organization



Three User Interfaces



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

Differences from PAW

- Regular grammar (C++) on command line
- Single language (compiled and interpreted)
- Object Oriented (use your class in the interpreter)
- Advanced Interactive User Interface
- Well Documented code. HTML class descriptions for every class.
- Object I/O including Schema Evolution
- 3-d interfaces with OpenGL and X3D.

PAW to ROOT File Conversion

- Get the example PAW file from http://www-root.fnal.gov/root/class/examples/toyz.rz
- At the system prompt type

```
> h2root toyz.rz  toyz.root
```

- Once you've done the conversion, you can then start a ROOT session and open toyz.root
- Also see: http://root.cern.ch/root/HowtoConvert.html

open terminal

ROOT Overview Summary

- PAW
- Concepts: Object Oriented Design, Frameworks
- Services and Utilities
- Libraries
- Physical Organization

GUI Basics

- Browsing and opening files
- Drawing histograms
- Right click, left click, middle click
- Draw Panel
- Fit Panel
- Adding Color and Zooming
- Adding text and other objects
- Dividing the canvas
- Setting the log scale

GUI Basics

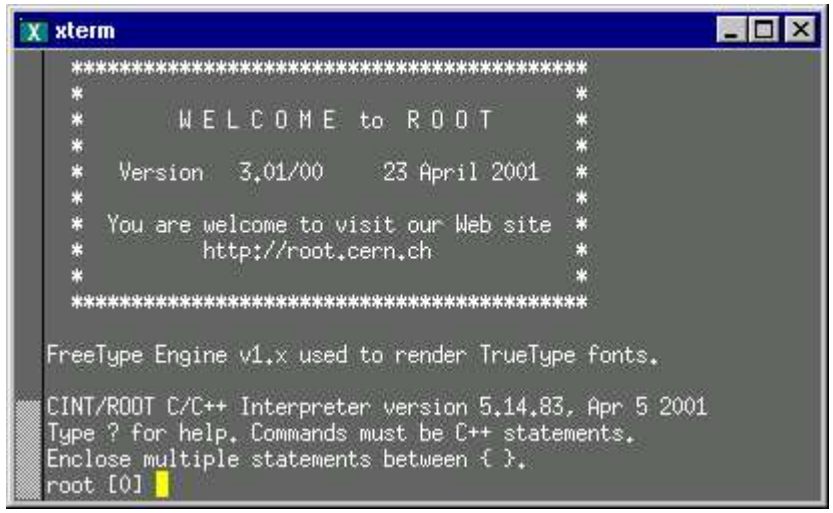
run root

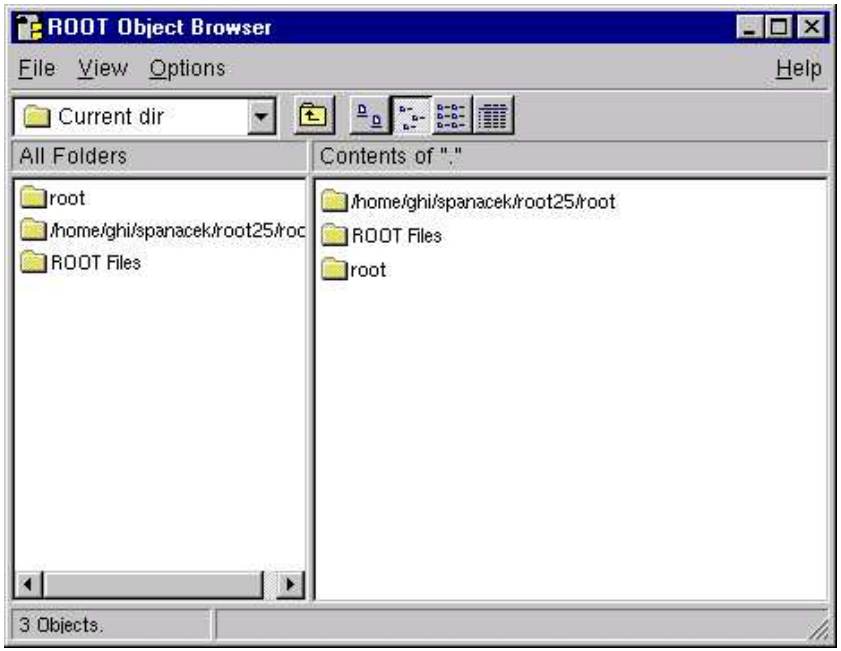
Start root

```
$ root
```

Quit root (just in case)

```
root[0] .q
```



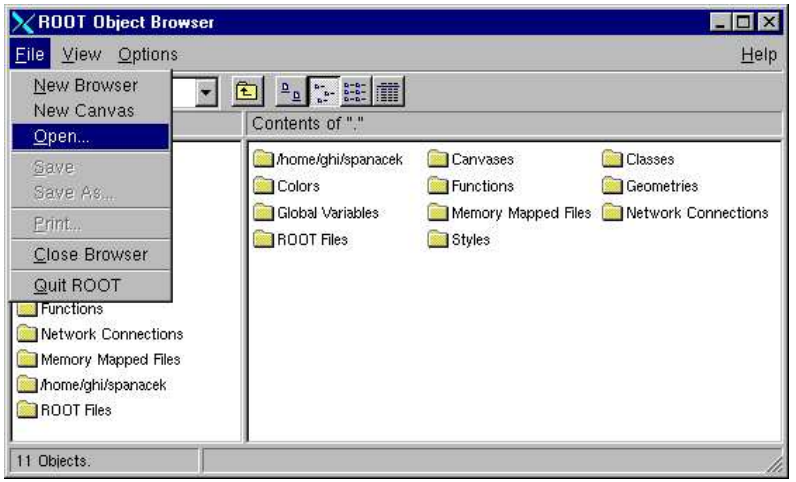


Display the browser

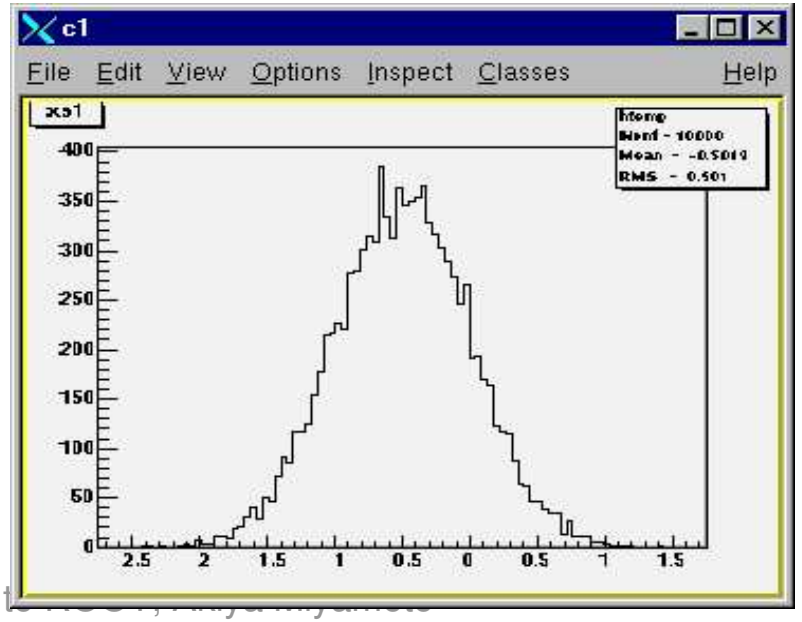
- `TBrowser b;`

Displaying a Histogram

Open the root file
Browse the file

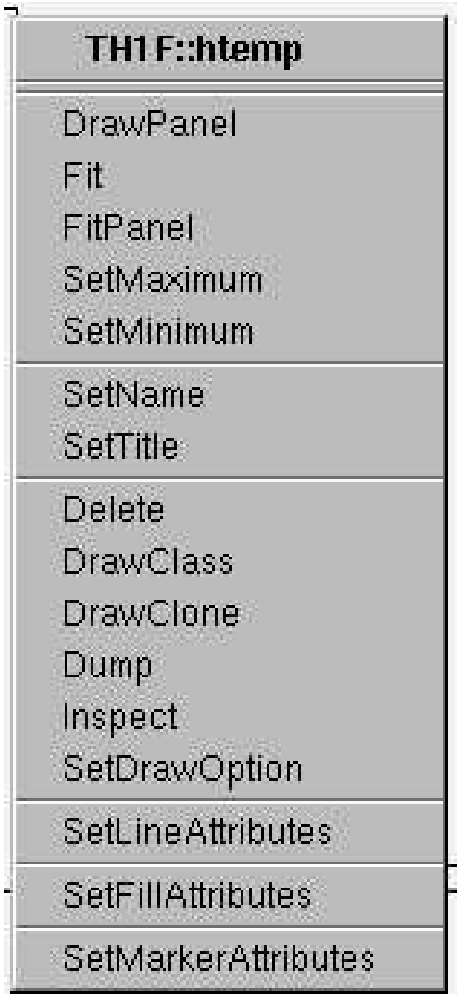


Display a histogram
The Canvas



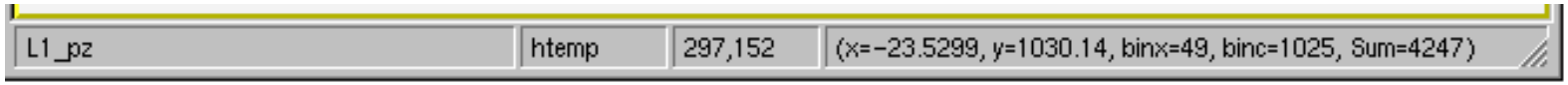
Basic Navigation by Clicking

- Left Click
  - select the object
  - drag the object
  - resize the object
- Right Click
  - context menu
  - class::name
  - methods
- Middle Click
  - activate canvas
  - freezes event status bar

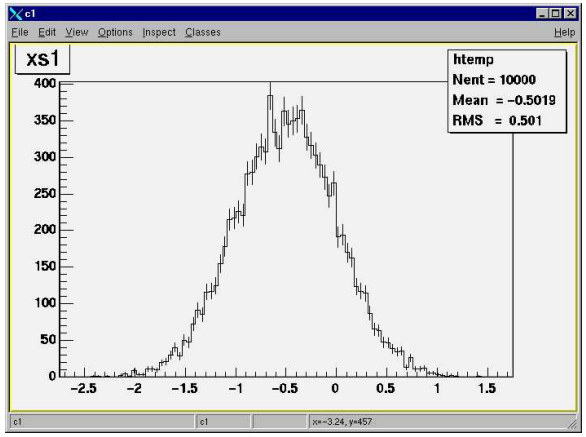


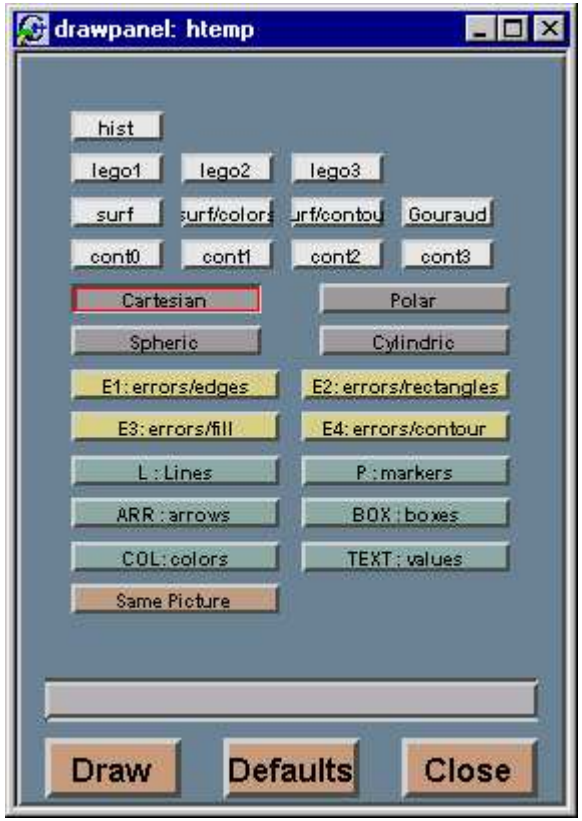
The Draw Panel

The Event Status

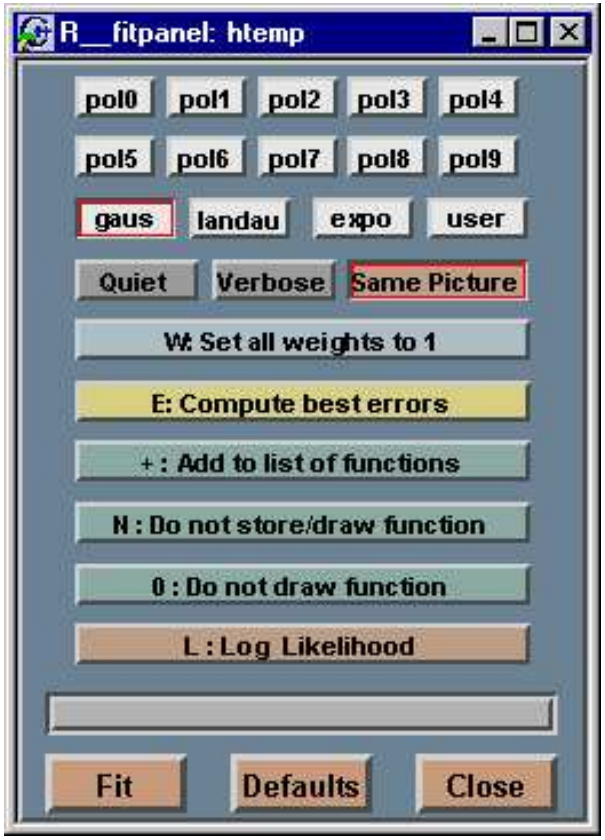


The Draw Panel
Adding Error bars
Slider
Defaults

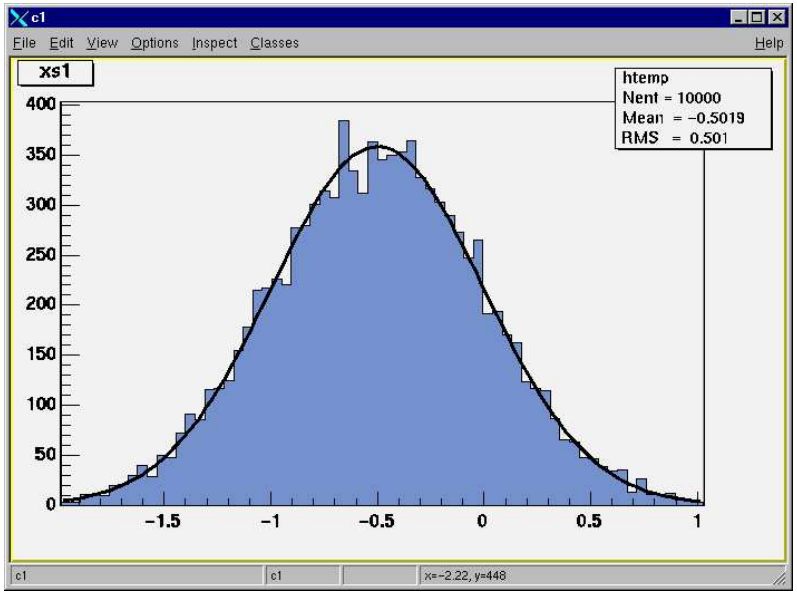




Fitting, Coloring, and Zooming

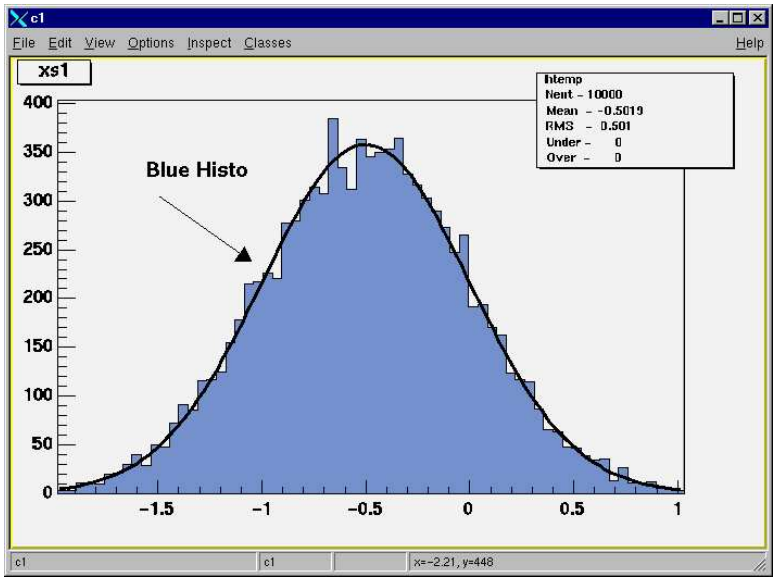


- Adding a gaussian fit
- Coloring the histogram
- Zooming/unzooming

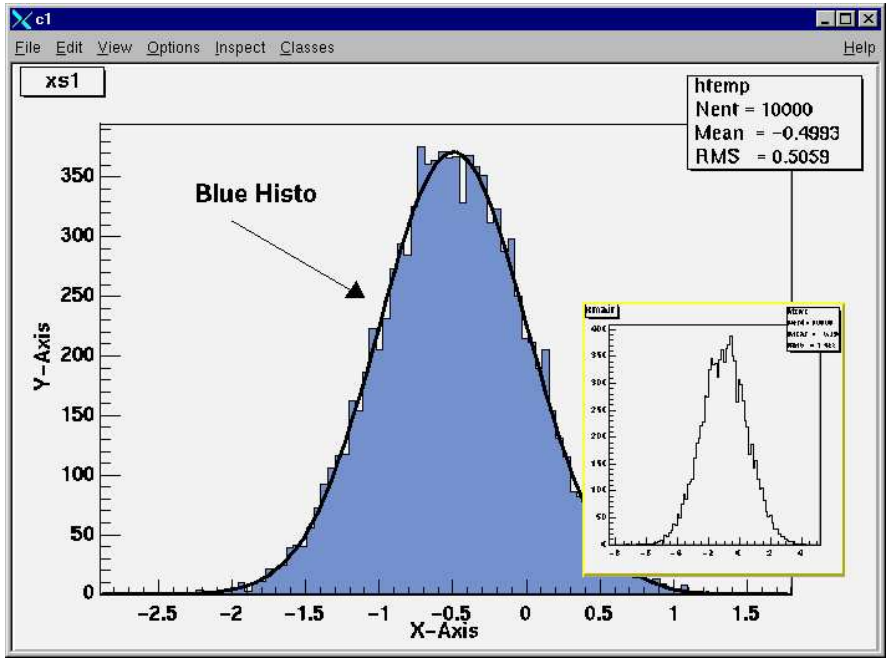


Adding Objects to the Canvas

- The Editor
- Adding an Arrow
- Adding Text

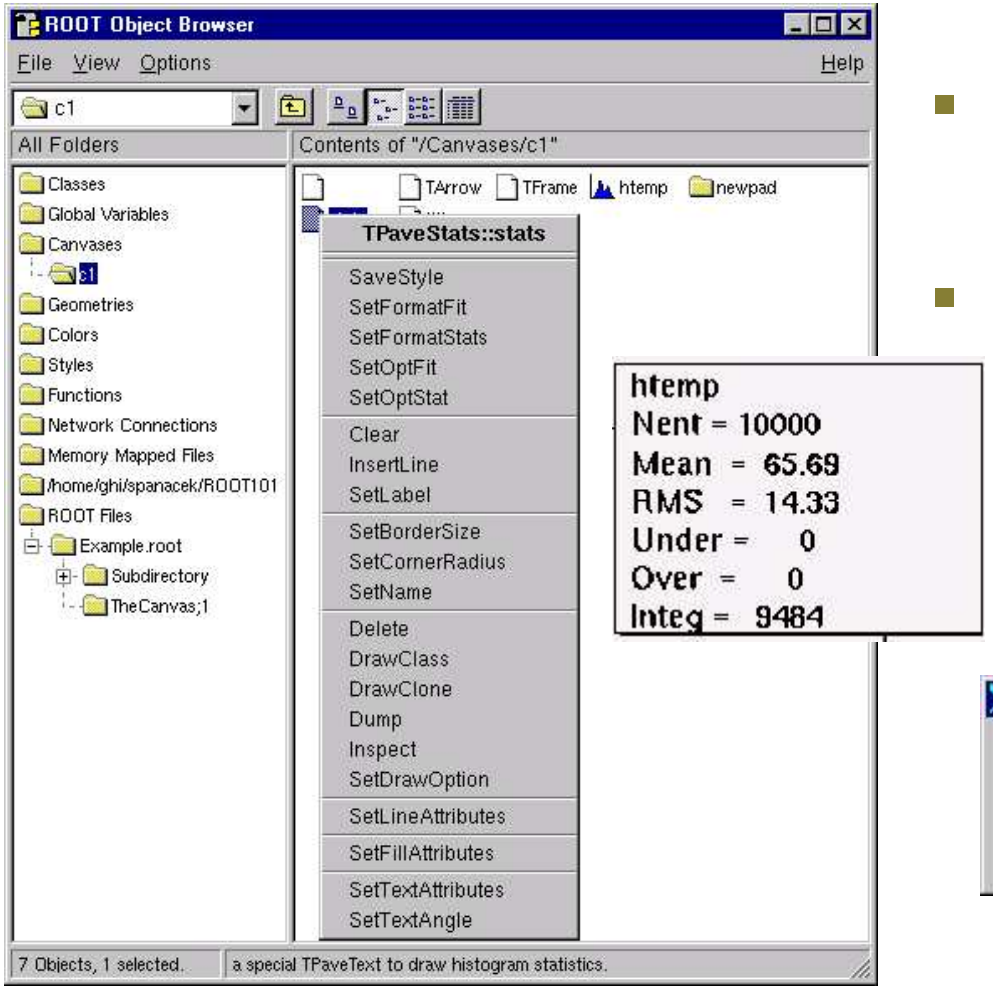


Adding another Pad

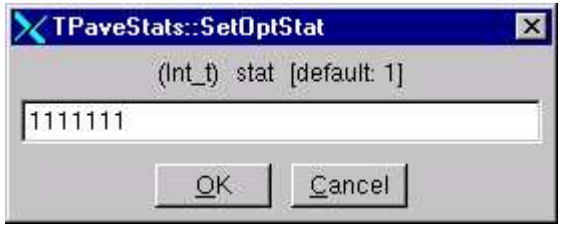


- Add a Pad
- Select the new Pad
- Draw a histogram
- Add a title for the axis

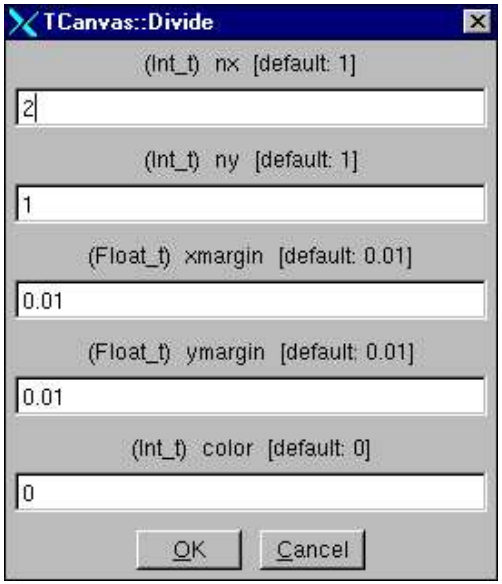
Modifying the Statistics



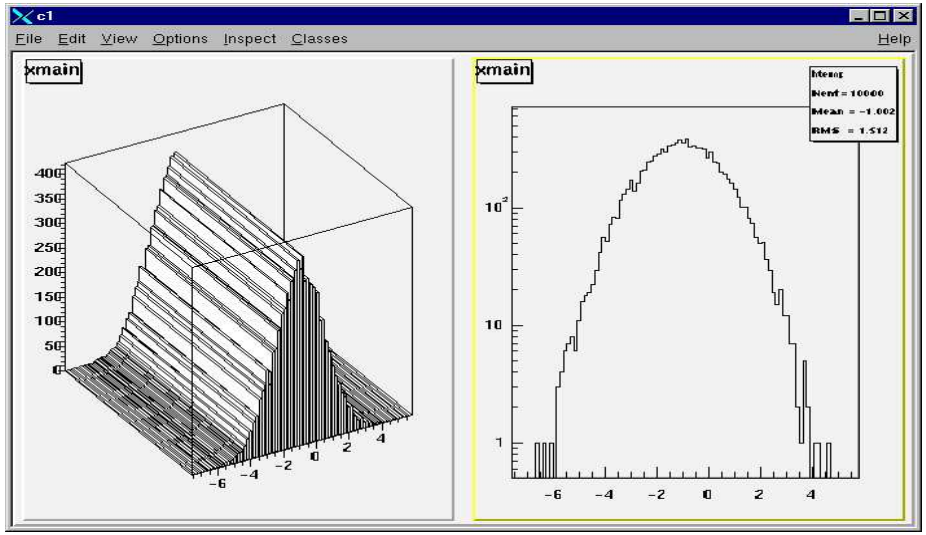
- The Canvas in the Browser
- Setting the (7) statistics options
  - default = 0001111



Dividing the Canvas



- Create a new Canvas
- Divide it in 2
- Draw two histograms.
  1. Lego plot
  2. LogY



Command Line Basics

- Use up and down arrows to recall commands
  - $HOME/.root_hist
- Use emacs commands to navigate
  - <tab>, <ctrl>-a, <ctrl>-e, <ctrl>-f

Open a File

- Open a file for reading

  root [] TFile f("Example.root")
- Look at the contents of the file

  root [] f.ls()

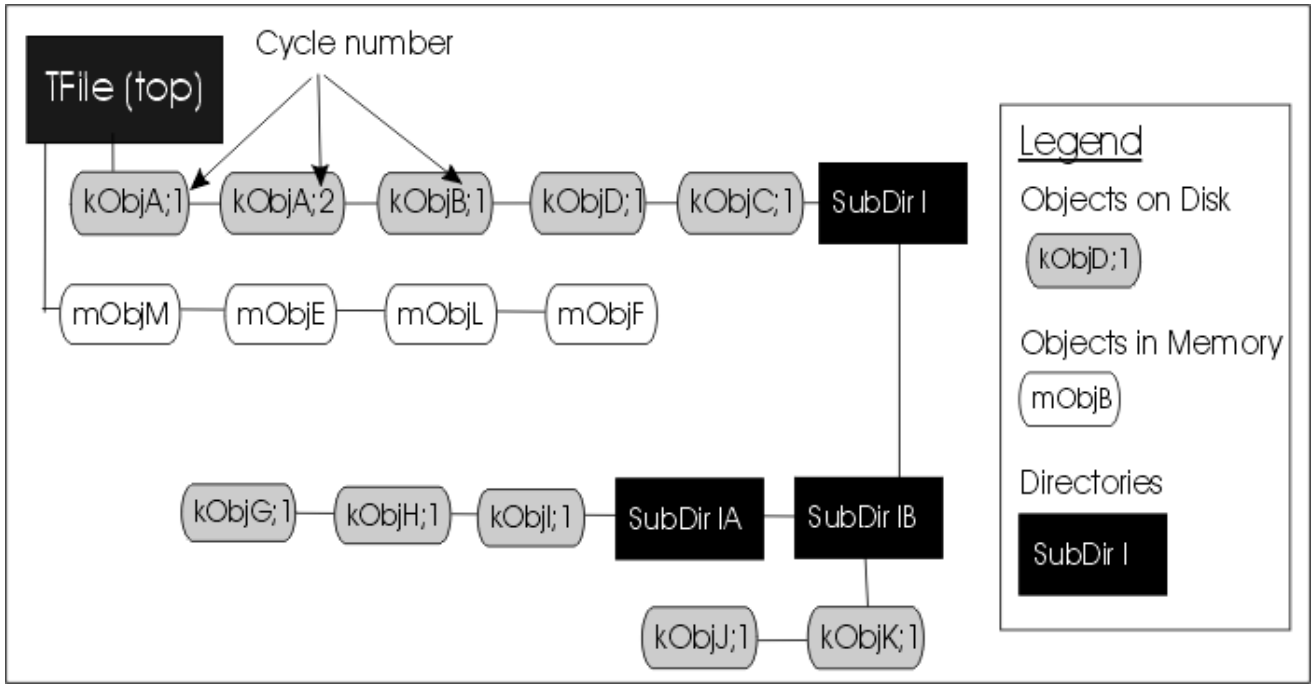
```
TFile**         Example.root    ROOT file
 TFile*         Example.root    ROOT file
  KEY: TTree    myTree;1        Example ROOT tree
  KEY: TH1F     totalHistogram;1        Total Distribution
  KEY: TH1F     mainHistogram;1 Main Contributor
  KEY: TH1F     s1Histogram;1   First Signal
  KEY: TH1F     s2Histogram;1   Second Signal
```

Plotting a Variable

- Plot a variable
  - `root [] myTree->Draw("xs1")`
  -
- Where did myTree come from?
  - ROOT executed an implicit `gROOT->FindObject("myTree")`
- And now..
  - `root[] f.ls()`
  - …

```
OBJ: TTree myTree  Example ROOT tree: 0 KEY:
TTree myTree;1 Example ROOT tree
```

The ROOT File



- A TFile is a directory structure like UNIX
- Object in Memory (OBJ) Object on Disk (KEY)

The ROOT Tree

- Stores large quantities of same class objects:
  - Optimize disk space
  - Optimize access speed
  - Had analysis methods (TTree::Draw)
- TNtuple is a TTree limited to floating point numbers.
- More details later …

Command Line Basics

- Open and browse a file
- Drawing histograms
- Adding a Fit
- Adding Error bars
- Adding Color

Summary

- Overview of ROOT
- GUI Basics
- Command line basics
- Using the Class reference guide
- PAW to ROOT conversion

S

Day 2

- Last time
  - GUI, Command line
- Today
  - Command Line (CINT)
  - Scripts (CINT & ACLiC)
  - Exercises
  - Functions and Fitting
  - TreeViewer

Command line

- Environment Settings
- Command types
- CINT Commands
- Global Variables
- TObject

Environment Settings

- Environment setting file:
- `$ROOTSYS/etc/system.rootrc`
  - looks first in current directory for .rootrc
  - second in $HOME/.rootrc
  - third in $ROOTSYS/etc/system.rootrc
- Find the current settings
  - root[ ] gEnv->Print()

open terminal

Environment Settings (cont.)

- The .rootrc file:
- The Macro Path
  - `Unix.*.Root.MacroPath:.:$(HOME)/myRootMacros`
- Options in .rootrc
  - `Root.ShowPath: false`
- History File
  - `$HOME/.root_hist`
- Automatically Executing Macros
  - `rootlogon.C`
  - `rootlogoff.C`
  - `rootalias.C`

Command Line Options

> root -/?

- Usage: root [-l] [-b] [-n] [-q] [file1.C ... fileN.C]
- Options:
- -b : run in batch mode without graphics
- -n : do not execute logon and logoff macros as specified in .rootrc
- -q : exit after processing command line macro files
- -l : do not show splash screen

> roo
- Usag
- Optio
- -b : r
- -n : q
- -q : e
- -l : d

Three Types of Commands

- CINT commands start with "."
  - `root[0].?`
    - this command will list all the CINT commands
  - `root[1].X [filename]`
    - load [filename] and execute function [filename]
  - `root[2].L [filename]`
    - load [filename]
- SHELL commands start with ".!" for example:

```
root[3] .! ls
```

Three Types of Commands

3. C++ syntax (almost)

```
root [0] TBrowser *b = new TBrowser()
or
root [0] TBrowser *b = new TBrowser();
```

The optional Semicolon:

Leave off the semicolon to see the return value of the command.

```
   root [0] 23+5  // show return value
   (int)28
   root [1] 23+5; // no return value
root [2]
```

Command Line Help

- Use the Tab feature to get help …

```
root [0] b = new TB <TAB>
root [1] b = new TBrow<TAB>
root [2] b = new TBrowser(<TAB>
```

- Find List of Methods
- Find Parameter list

Coding Conventions

Based on Taligent

- Classes begin with T TTree, TBrowser
- Non-class types end with _t Int_t
- Data members begin with f fTree
- Member functions begin with a capital Loop()
- Constants begin with k kInitialSize, kRed
- Static variables begin with g gEnv
- Static data members
  begin with fg fgTokenClient

CINT Types

C++ type	Size (bytes)	ROOT types	Size (bytes)	FORTRAN analog
(unsigned)char	1	(U)Char_t	1	CHARACTER*1
(unsigned)short (int)	2	(U)Short_t	2	INTEGER*2
(unsigned)int	2 or 4	(U)Int_t	4	INTEGER*4
(unsigned)long (int)	4 or 8	(U)Long_t	8	INTEGER*8
float	4	Float_t	4	REAL*4
double	8 (=4)	Double_t	8	REAL*8
long double	16 (= double)			REAL*16

CINT Extensions to C++

1. Declaration can be omitted

```
f = new TFile("Example.root")
```

2. "." notation rather than "->"

```
f.ls()
```

3. Search for an object by its name

```
TH1F *smallHisto = new TH1F
          ("small","fPx 100",100,-5,5);
small->Draw();
```

Warning: These will not work in compiled code!

CINT Commands

- [expression] evaluates the expression

```
root[3] 3*4
(int)12
```

- .files show loaded source files
- .class [name] show class definition
- .g prints all objects in the root session
- .ls ls on current directory
- .pwd list the current directory, canvas, and style.
- .![shell-command] execute shell command

Demo on CINT Commands

.class

```
root [0] .L $ROOTSYS/test/libEvent.so
root [1] .class Event
```

.g

```
root [2] .g
  ...
  0x104c7560 Event e , size=56
    0x0         private: Int_t fNtrack
    0x0         private: Int_t fNseg
    0x0         private: Int_t fNvertex
    ...
```

Global Variables

- gRandom
  - `gRandom->Gaus(1,2)`
  - You can replace the random generator with your own:
    ```
    delete gRandom;
    gRandom = new TRandom2(0); //seed=0
    ```
- gFile
  - `gFile->GetName()`
- gDirectory
  - `gDirectory->GetName()`
- gSystem

    `gSystem->HostName()`

gROOT

- global ROOT session object
- gROOT->GetListOf< list type >();
- gROOT->LoadMacro();
- gROOT->Time();
- gROOT->ProcessLine()

gR
- gl
- gR
- gR
- gR
- gR

gROOT->FindObject()

```
TH1F *smallHisto = new TH1F
   ("small","fPx 100",100,-5,5);
```

- "small" is the Object Name

```
gROOT->FindObject("small")
   (class TObject*)0x104c7528
```

- "smallHisto" is the Variable Name

```
gROOT->FindObject("smallHisto")
   (class TObject*)0x0 // null pointer
```

- FindObject needs the Object Name.

gROOT->FindObject() ++

FindObject returns a pointer to TObject.

This generates an error:

```
gROOT->FindObject("small")->GetBinContent(2)
```

This is OK:

```
gROOT->FindObject("small")->ClassName()
TH1F* histo=(TH1F*) gROOT->FindObject("small")
histo->GetBinContent(2)
```

gROOT->FindObject() cont.

Due to CINT magic this is also OK:

```
TH1F *smallHisto = new TH1F
        ("small","fPx 100",100,-5,5);
small->GetBinContent(2);
```

- CINT implicitly executes a FindObject("small")
- Casts it to the correct class
- Creates a variable called "small" of the correct class

Warning: This will not work in compiled code!

Summary (Command Line)

- Environment Settings
- Command types
- CINT Commands
- Global Variables
- TObject

Writing Scripts

- Named and Un-named Scripts
- Debugging
- ACLiC

Scripts

Un-named Script

- Start with "{" and end with "}"
- All variables are in the global scope
- No class definitions
- No function declarations
- No parameters

Named Script

- C++ functions
- Scope rules follow standard C++
- Function with the same name as the file is executed with a .x
- Parameters
- Class definitions (derived from a compiled class at your own risk)

Scripts Examples

- Un-named Script: hello.C

```
{
  cout << "Hello" << endl;
}
```

- Named Script:say.C

```
void say(char * what = "Hello")
{
  cout << what << endl;
}
```

- Executing the Named Script

```
root [3] .x say.C
Hello
root [4] .x say.C("Hi there")
Hi there
```

Resetting the Environment

- gROOT->Reset()
  - Calls destructors of all objects created on the stack
  - Objects on Heap are not deleted, but pointer variable is disassociated

Tracking Memory Leaks

- Counting Objects and Memory use

  In the .rootrc or system.rootrc file:

```
Root.MemStat:    1
Root.ObjectStat:1
```

- Print the Object count and Memory use

```
gObjectTable->Print();
```

Tracking Memory Leaks+

- Example output:

  Before .x FirstContour.C:

	count	on heap	size	total size	heap size
Total:	1,079	1,046	3,160	**49,992**	**45,824**

  After:

Total:	1,783	1,749	17,920	**118,912**	**114,568**

- Put `gObjectTable->Print()` before and after code segment in your script to find memory leaks.

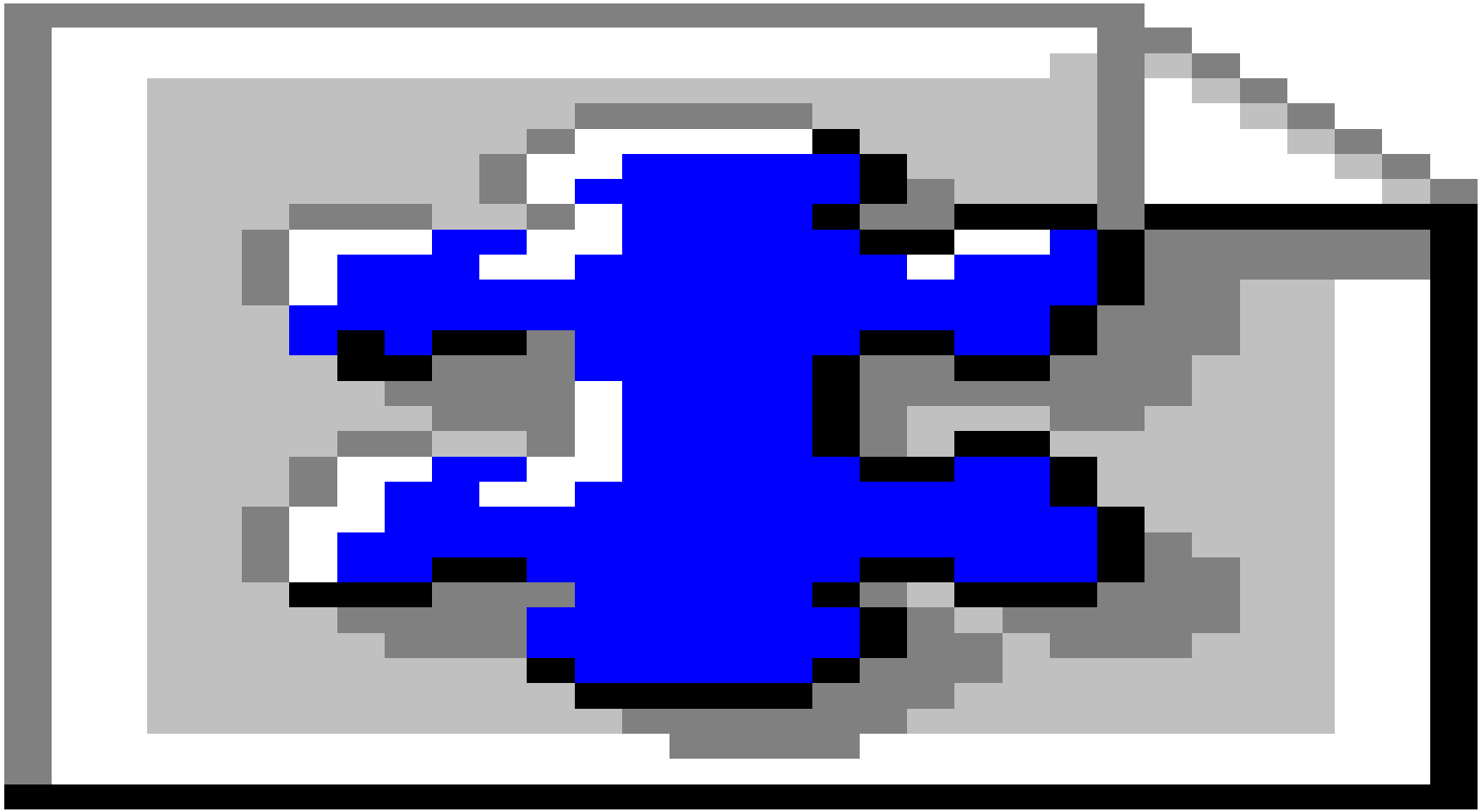
Summary (Scripts)

- Named and Un-named Scripts
- ~~Debugging~~
- ~~ACLiC~~

Functions and Fitting

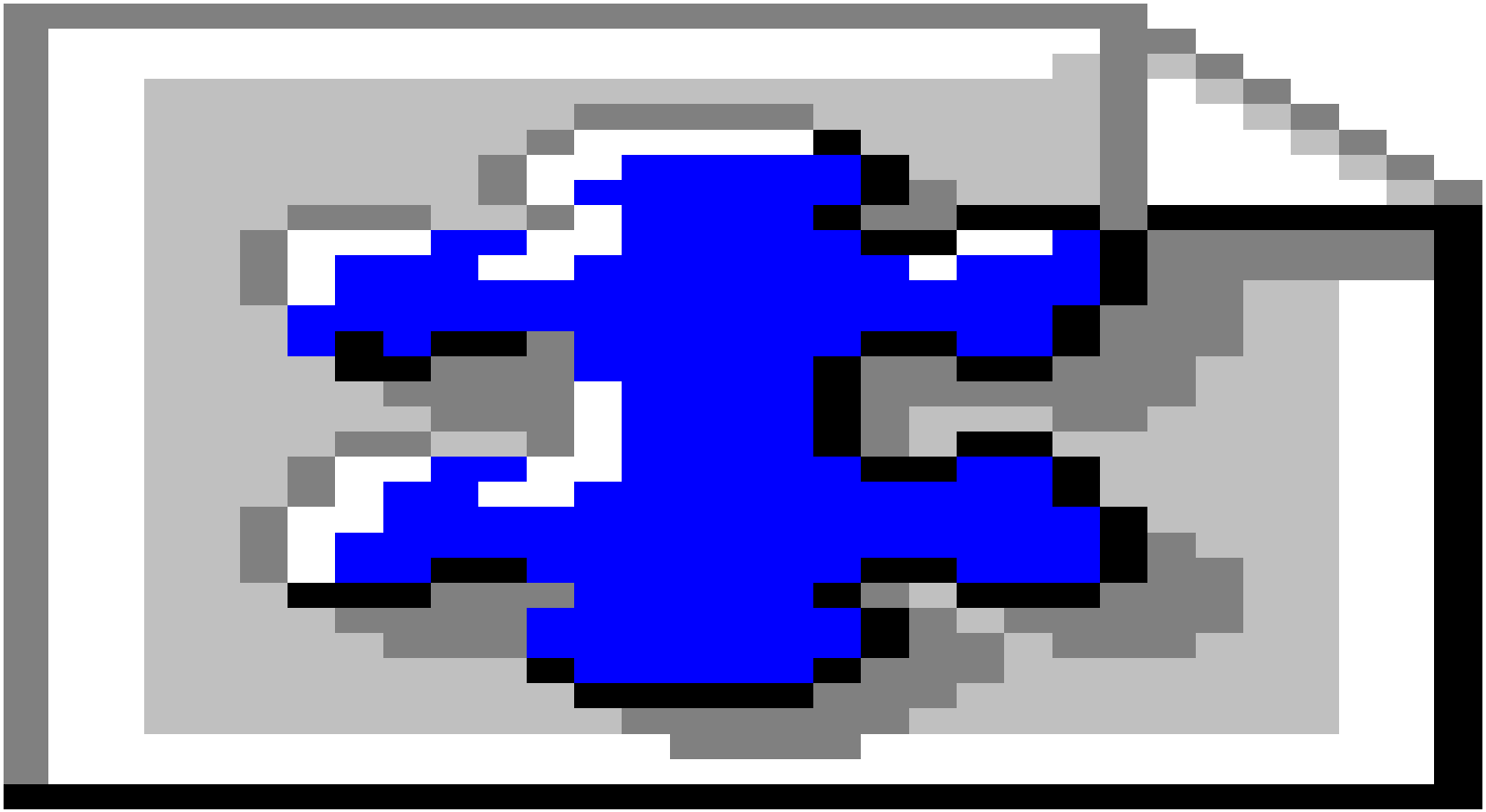
- Function Objects (TF1)
  - Three constructors for TF1
  - User Defined Functions
- Fitting
  - Fit()
  - Fitting with a user defined function
  - Fitting subranges and combining functions
  - Demonstration of background and signal function

Histogram example-1



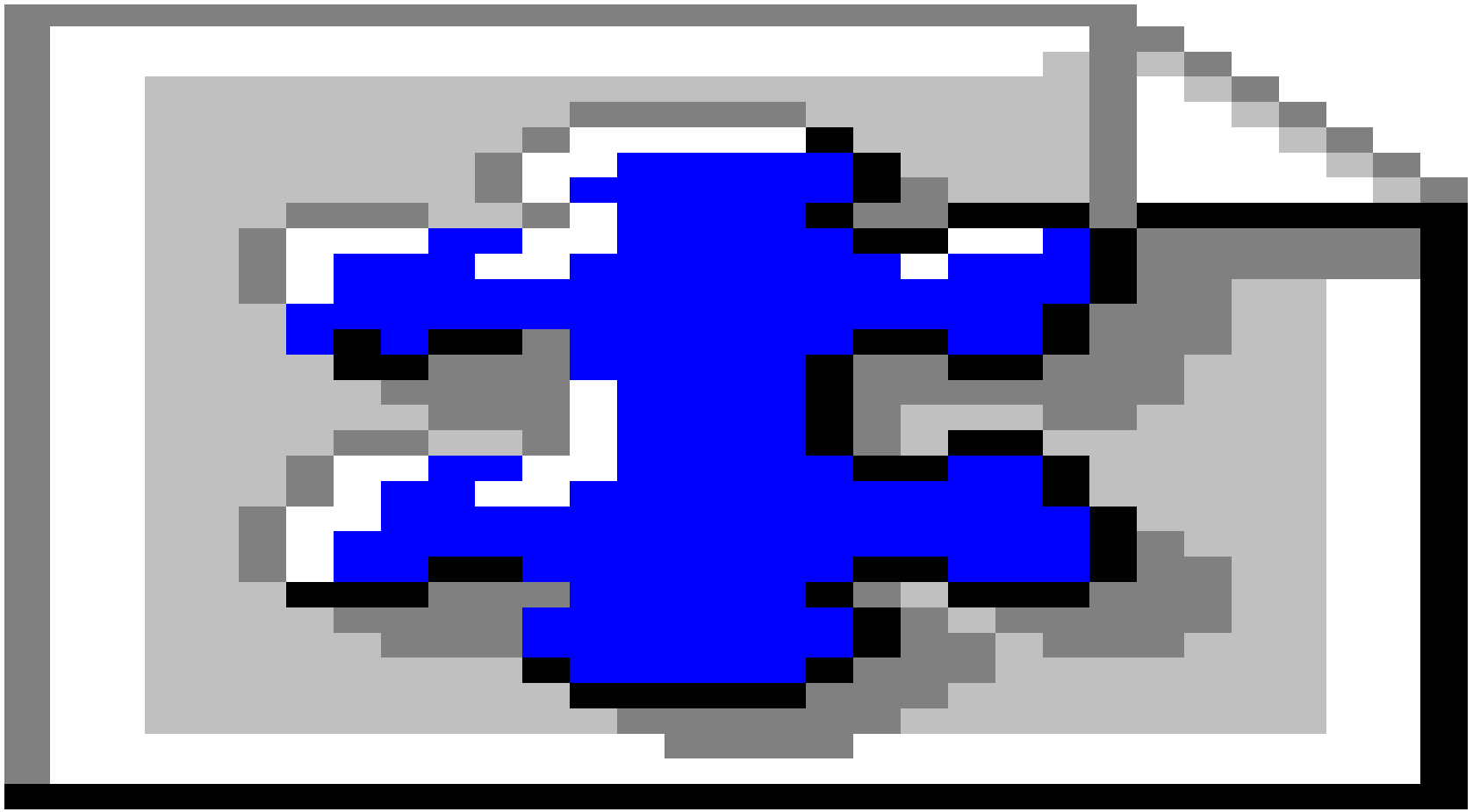
root/demo/ex0/hist1.C

Histogram Example – 2



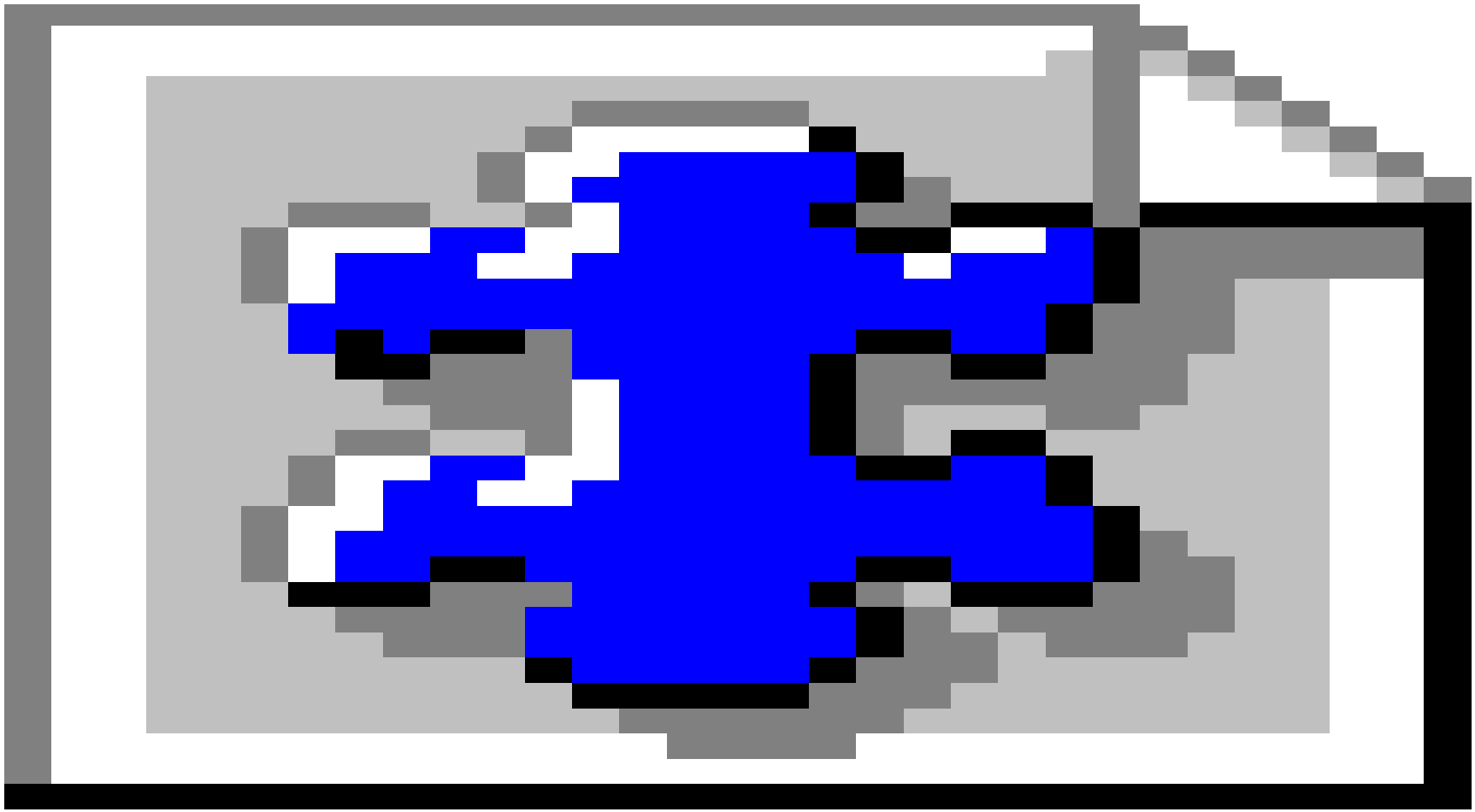
root/demo/ex0/hist2.C

Histogram ex.3 - Save to file



root/demo/ex0/hist3.C

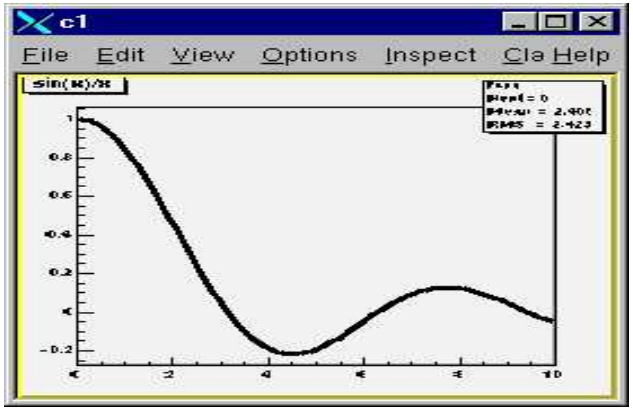
Plot data points: TGraph, TGraphError, ...



root/demo/ex0/plot2.C

TF1 Constructors

1. A C++ like expression using x with a fixed set of operators and functions defined in TFormula




```
TF1 *f1 = new TF1("f1","sin(x)/x",0,10);
f1->Draw();
TF1 *f2 = new TF1("f2","f1 * 2",0,10);
```

TF1 Constructors (cont.)

2. Same as the previous TF1 with Parameters

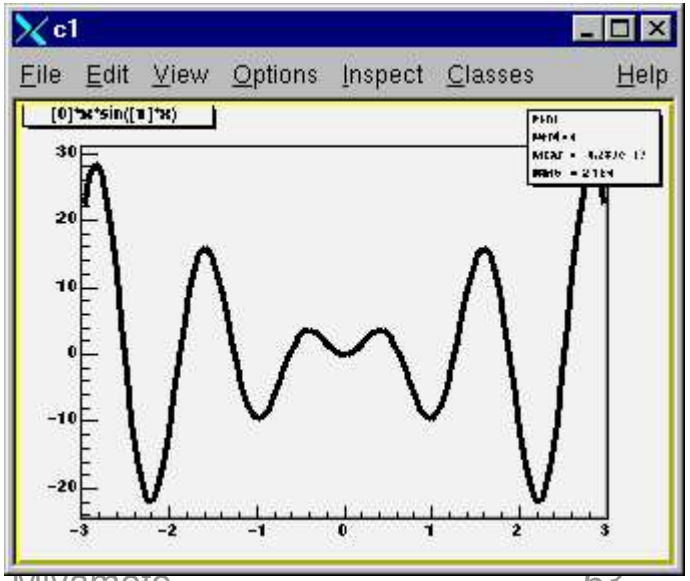
   Call the constructor with parameter indices

```
TF1 *f1 =
  new TF1 ("f1","[0] *x*sin( [1] *x)",-3,3);
```

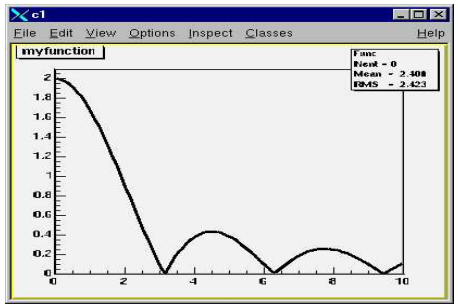
   See TFormula for valid expressions

Set the parameters explicitly

```
f1->SetParameter(0,10);
f1->SetParameter(1,5);
f1->Draw();
```



TF1 Constructors (cont.)



3. Use a defined function

- Define a function

```
Double_t MyFunction(Double_t *x, Double_t *par){
 Float_t  xx =     x[0];
 Double_t val=
       TMath::Abs(par[0]*sin(par[1]*xx)/xx);
 return val;
}
```

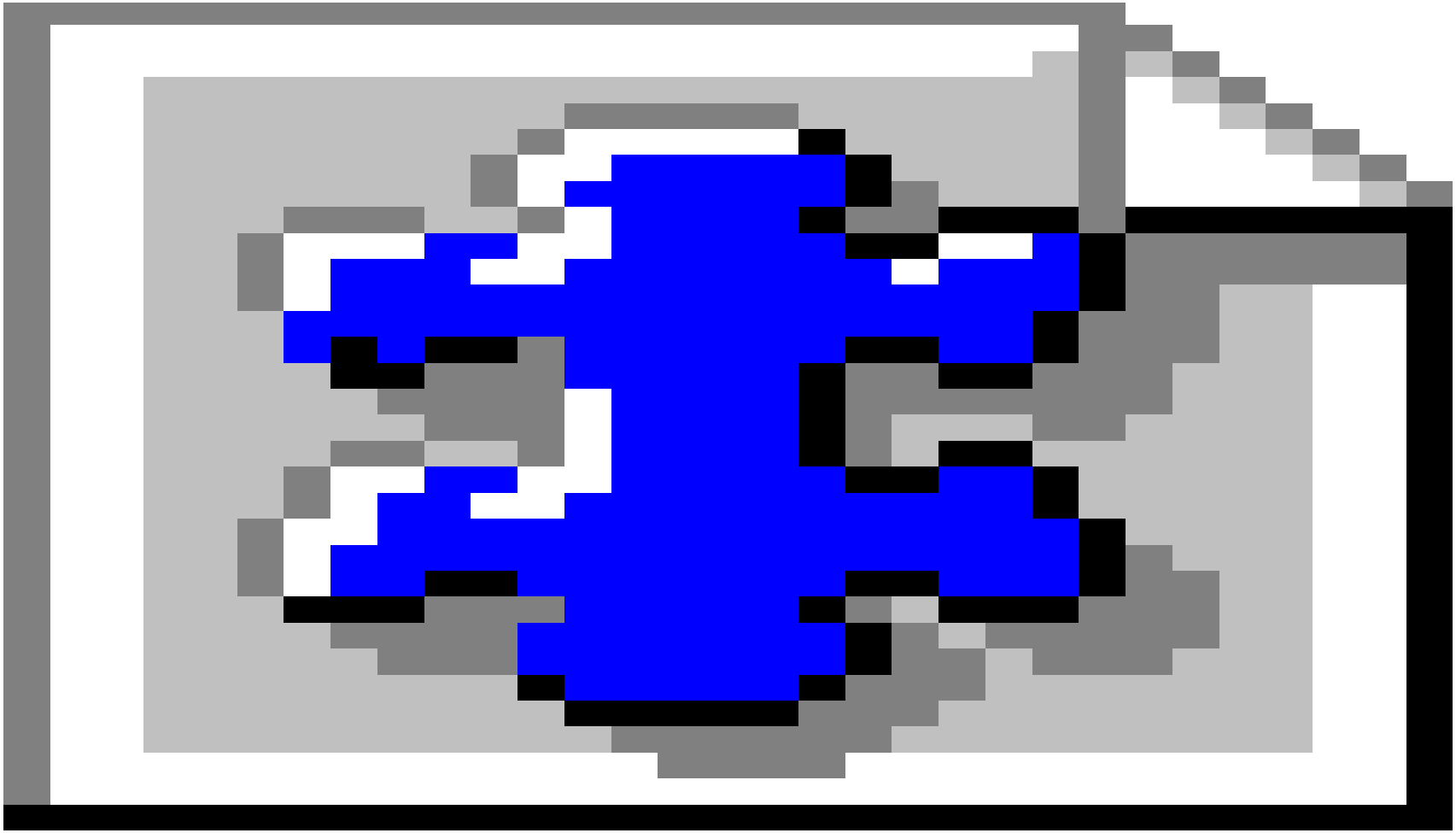
- TF1 constructor

```
TF1 *f1 = new TF1("f1",MyFunction,0,10,2);
```

- NOTE: The 2 is the number of parameters in MyFunction.
- Set the parameters

```
f1->SetParameters(2,1);
```

Fitting - example



root/demo/ex0/hist4.C

Fitting - other options

- Fitting sub-range
  h->Fit(“func”,”R”);
  h->Fit(“func2”,”R+”);
- Fit using Minuit
- Get fitted results

Summary: Fitting and Functions

- Functions Objects (TF1)
  - Three constructors for TF1
  - User Defined Functions
- Fitting
  - Fit()
  - Fitting with a user defined function
  - Fitting sub ranges and combining functions
  - Demonstration of background and signal function

Tree Create and Analysis

- Create tree and branch: root/demo/tree/make_tree.C
- Read Tree:
  - By Draw method: root/demo/tree/read_tree.C
    t->Draw(variable, critera, option, nentries, 1st_entry)
    - Draw variables: 1D, 2D, OverWrite
    - Put cuts
    - Limit number of events
    - Extract histogram
  - Event loop by yourself

Create a html file from source

- Use THtml class
- Anything between begin_html and end_html is raw html command
- Link to class name is created automatically.
-

```
THtml  html;
html.MakeClass( class_name );
html.Convert( script_file, title, out_dir );
```

Other Useful ROOT Classes

- TMath::xxxx
- TRandom/TRandom2/TRandom3
- TMatrix
- TLorentzVector/TLorentzRotation
- TLatex
- TMinuit
- TGxxxxx : graphic
- TPythia6

TObjArray,TClonesArray,TIter

- TObjArray
  - Container class for variable-size object, any class
    - Add
      Add(obj)
    - Read
      At(i), UncheckedAt(i), TIter
- TClonesArray
  - Container class for same class, fixed-size object
    - Add
      new(clone[i]) TObject

root/array/tobjarray.C, tclonesarray.C

Class Schedule Day 3:

- Building Root Trees
- Reading Root Trees
- Using Trees in Analysis
  - The Draw method
  - The MakeClass method
- Add your class to ROOT
  - With the Interpreter
  - With a compiler (Shared Library)
  - With ACLiC

A Class From a Shared Library

Step 1:

define your class as a descendent of TObject and write the implementation.

SClass.h

```
#include <iostream.h>
#include "TObject.h"
class SClass : public TObject {
…
```


A
Ste
d
T
SCla
#incl
#incl
class
…
26-May

A Class From a Shared Library

Step 2:

ClassDef(*ClassName,ClassVersionID*)

At the end of the class definition

ClassImp(*ClassName*)

At the beginning of the implementation file

ClassDef and ClassImp*

ClassDef and ClassImp needed for Object I/O :

These macros can automatically create:

- Streamer method needed for writing to ROOT files and Trees.
- ShowMembers()
- >> operator overload

The LinkDef file

Step 3:

create a LinkDef.h file

```
#ifdef __CINT__
#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;
#pragma link C++ class SClass;
#endif
```


Ste
crea
#i
#p
#p
#p
#p
#e
26-May

The LinkDef Options:

- "-" do not generate a streamer.
  - `#pragma link C++ class SClass-;`
  - Use for objects with customized streamers
- "!" do not generate the operator >>
  - `#pragma link C++ class SClass-!;`
  - Use for classes not inheriting from TObject.
- "+" use the byte count check
  - `#pragma link C++ class SClass+;`
  - !! In ROOT 3 the "+" turns on the new ROOT IO.

Makefile and rootcint

Step 4:

Write a Makefile and call rootcint to add your class to the dictionary:

```
SClassDict.cxx SClass.h LinkDef.h
   $(ROOTSYS)/bin/rootcint -f
     SClassDict.cxx -c SClass.h LinkDef.h
```

rootcint …

- LinkDef.h must be the **last** argument on the rootcint command line.
- The LinkDef file name **MUST** contain the string:
  - LinkDef.h or linkdef.h, i.e. NA49_LinkDef.h is fine just like, mylinkdef.h.
  -

Compile and Load

Compile the class using the Makefile

```
gmake -f Makefile.sgikcc
```

Load the shared library

```
root [0] .L SClass.so
root [1] SClass *sc = new SClass()
root [2] TFile *f = new TFile("Afile.root",
"UPDATE");
root [3] sc->Write();
```

More information on ROOT

- ROOT home page http://root.cern.ch/
  - Root class categories http://root.cern.ch/root/Categories.html
  - Class Index http://root.cern.ch/root/html304/ClassIndex.html
  - Root How to http://root.cern.ch/root/Howto.html

- $ROOTSYS/tutorial

  To run demo programs, do

  $ root demos.C