ROOT

Tutorials

ROOT, Tutorials

Authors: René BRUN *(Rene.Brun@cern.ch)*
Nenad BUNCIC *(Nenad.Buncic@cern.ch)*
Valery FINE *(Valeri.Faine@cern.ch)*
Fons RADEMAKERS *(Fons.Rademakers@cern.ch)*

Cover Page: Marjana RAPAIC BUNCIC

Table of Contents

Tutorials

The log file of a simple interactive session

```
The output below is a direct copy of an interactive ROOT session.
The ROOT system identifies itself showing a logo and a version number.
At startup time, some environment variables are read from the
local file .rootrc (see an example of a .rootrc file below).
The .rootrc file references a logon macro and a logoff macro.
the logon macro is called rootlogon.C and an example of this macro
is also shown below. Same for the logoff macro called rootlogoff.C.
```

// Start Interactive session by invoking the module root.
// The only command typed in this session is ".q".

```
root
 *******************************************
 *                                         *
 *        W E L C O M E  to  R O O T       *
 *                                         *
 *   Version   0.90/09   3 December 1996   *
 *                                         *
 *  You are welcome to visit our Web site  *
 *          http://root.cern.ch            *
 *                                         *
 *******************************************
CINT/ROOT C/C++ Interpreter version 5.11.27, Nov 27 1996
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
Welcome to the ROOT tutorials
Root >  .q
This is the end of ROOT -- Goodbye
```

// End of the interactive session

// This is the file .rootrc
// The description of possible variables in .rootrc is given in TEnv.

```
cat .rootrc
# Path used by dynamic loader to find shared libraries
*.Root.DynamicPath:    .:~/rootlibs:$ROOTDIR/lib
# Activate memory statistics
Rint.Root.MemStat:       1
Rint.Load:               rootalias.C
Rint.Logon:              rootlogon.C
Rint.Logoff:             rootlogoff.C
Rint.Canvas.MoveOpaque:  false
Rint.Canvas.HighLightColor: 5
```

// This is the file rootlogon.C

```
{
  printf("Welcome to the ROOT tutorials");
}
```

// This is the file rootlogoff.C

```
{
  printf("This is the end of ROOT -- Goodbye");
}
```

The Hello World example

This example is very basic. It illustrates the power of the ROOT system. Start the interactive version of ROOT and execute the two following commands:

```
Root >  TPaveLabel hello(0.2,0.4,0.8,0.6,"Hello World");
Root >  hello.Draw();
```

The first command creates an object of class **TPaveLabel**. The second command will draw this object and will automatically create a graphics canvas. The default canvas created by the **Draw** command has a default X and Y range defined between 0 and 1. The first 4 parameters in the first command are the lower and upper coordinates of the **Pave**.

You can now use the mouse and try the following:

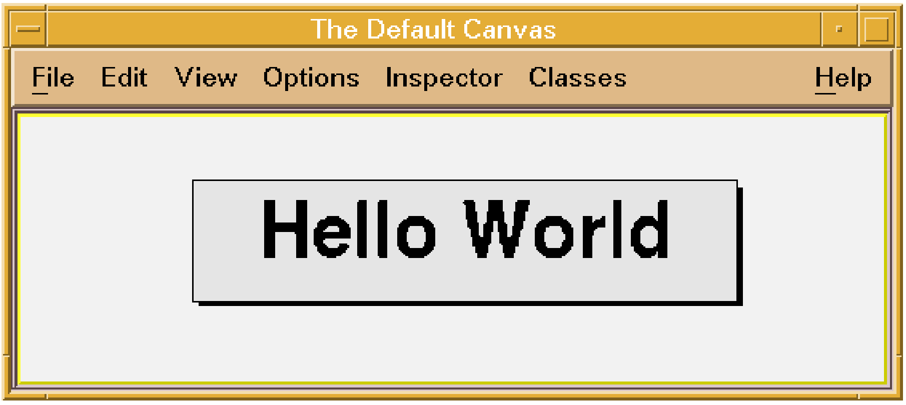
• Move the mouse on the Pave and press the left button. Keeping the left button pressed, you can move the Pave in the Canvas.

• Point to one of the corners of the Pave and grow or shrink the Pave.

• Move the mouse on the Pave and press the right button. You get a context pop-up menu showing a list of possible actions. Try to change for example the text color.

• Same as above, but select the action **Dump**. This will print the actual parameters of the object **hello** on standard output.

• Make a Postscript copy of the canvas by selecting the option in the **File** menu at the top of the canvas.



The Command Line interface

ROOT commands are executable C++ statements with a few extensions.

Note that:

• Commands are case sensitive.

• The semi colon at the end of a command may be omitted. However, note that the semi colon is mandatory at the end of a statement in a macro.

• ROOT does not make any difference between an object and a pointer to an object. **Object.Function** and **Object->Function** are equivalent.

To execute a macro, use the **.x** command. For example:

```
Root >  .x  macro.C;
```

a ROOT macro is a real and valid C++ file enclosed in curly brackets {}. ROOT accepts the following extensions:

• A statement like **TCanvas *c1 = new TCanvas(..)** can be abbreviated to: **c1 = new TCanvas(...)**.

• A function prototype is not required at the beginning of the macro.

• ROOT can automatically get in memory persistent objects from a file. For example, if we have an object called **hpx** in the current directory, one can type:

```
Root >  hpx.Draw();
```

All commands must be preceded by a . (dot), except for the evaluation statement { } and the ?.

```
==========================================================================
Dump:        n [file]  : create new readline dumpfile and start dump
            y [file]  : append readline dump to [file]
            z         : stop readline dump
            < [file]  : execute readline dumpfile
            > [file]  : output redirect to [file]
            2> [file] : error redirect to [file]
Help:        ?         : help
            help      : help
            /[keyword]: help information for keyword
Completion:  [nam][Tab]: complete symbol name start with [nam]
            [nam][Tab][Tab]: list up all symbol name start with [nam]]
Shell:       ![shell]  : execute shell command
Source:      v <[line]>: view source code
            V [stack] : view source code in function call stack
            t         : show function call stack
            f [file]  : view source file [file]
            trace <classname> : turn on trace mode for class
            deltrace <classname> : turn off trace mode for class
            break [classname] : break at the memberfunc
            delbreak [classname] : turn off memberfunc break
Evaluation:  p [expr]  : evaluate expression (no declaration/loop/condition)
            {[statements]}: evaluate statement (any kind)
            x [file]  : load [file] and evaluate {statements} in the file
            X [file]  : load [file] and execute function [file](minus extension)
            E <[file]>: open editor and evaluate {statements} in the file
Load/Unload: L [file]  : load [file]
            U [file]  : unload [file]
            reset     : reset interpreter environment
Monitor:     g <[var]> : list global variable
            l <[var]> : list local variable
            class <[name]>: show class definition (one level)
            Class <[name]>: show class definition (all level)
            typedef <name>  : show typedefs
            function  : show interpreted functions
            macro     : show macro functions
            template  : show templates
            include   : show include paths
            file      : show loaded files
            garbage   : show garbage collection buffer
            Garbage   : Do garbage collection
            cover [file]: save trace coverage
            return [val]: return undefined symbol value
Run:         S         : step over function/loop
            s         : step into function/loop
            i         : ignore and step over
            c <[line]>: continue <to [line]>
            e         : step out from function
            b [line]  : set break point
            db [line] : delete break point
```

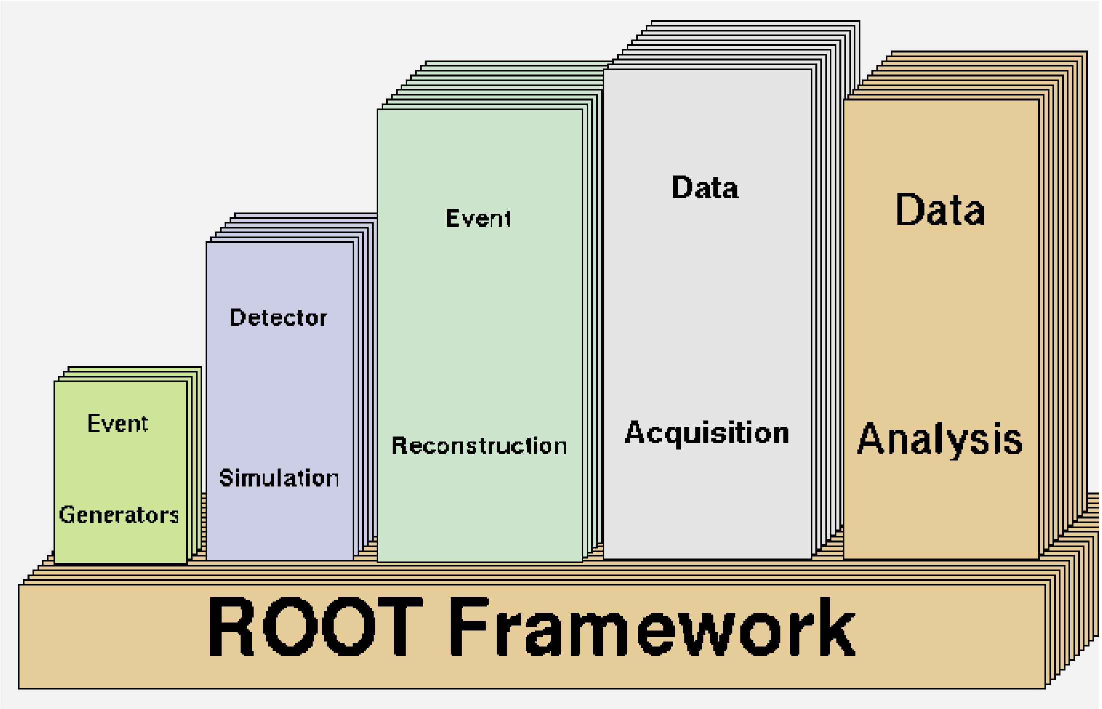
```
             a [assert]: break only if assertion is true
Quit:         q          : quit cint execution
             qqq        : quit anyway
```

If an object resides in the current directory, a reference to this object puts the object in memory.

Objects in the list of files, geometries, canvases may be accessed directly by name.

An Example of Object Oriented User Interface

```
{

// An example with basic graphics illustrating the Object Oriented
// User Interface of ROOT. The picture
// produced is the one shown in Mission Statement.

  gROOT->Reset();
  c1 = new TCanvas("c1","The ROOT Framework",200,10,700,500);
  c1->Range(0,0,19,12);

  TPavesText rootf(0.4,0.6,18,2.3,20,"tr");
  rootf.AddText("ROOT Framework");
  rootf.SetFillColor(42);
  rootf.Draw();

  TPavesText eventg(0.99,2.66,3.29,5.67,4,"tr");
  eventg.SetFillColor(38);
  eventg.AddText("Event");
  eventg.AddText("Generators");
  eventg.Draw();

  TPavesText simul(3.62,2.71,6.15,7.96,7,"tr");
  simul.SetFillColor(41);
  simul.AddText("Detector");
  simul.AddText("Simulation");
  simul.Draw();

  TPavesText recon(6.56,2.69,10.07,10.15,11,"tr");
  recon.SetFillColor(48);
  recon.AddText("Event");
  recon.AddText("Reconstruction");
  recon.Draw();

  TPavesText daq(10.43,2.74,14.0,10.81,11,"tr");
  daq.AddText("Data");
  daq.AddText("Acquisition");
  daq.Draw();

  TPavesText anal(14.55,2.72,17.9,10.31,11,"tr");
  anal.SetFillColor(42);
  anal.AddText("Data");
  anal.AddText("Analysis");
  anal.Draw();
  c1->Update();
}
```


Event
Generators
Detector
Simulation
Event
Reconstruction
Data
Acquisition
Data
Analysis
ROOT Framework

Reading an ascii file and making an ntuple

```
{

// example of macro to read data from an ascii file and
// create a root file with an histogram and an ntuple.

  gROOT->Reset();
  FILE *fp = fopen("/user/brun/root/basic.dat","r");
  Float_t x,y,z;
  Int_t ncols;
  Int_t nlines = 0;
  TFile *f = new TFile("basic.root","RECREATE");
  TH1F *h1 = new TH1F("h1","x distribution",100,-4,4);
  TNtuple *ntuple = new TNtuple("ntuple","data from ascii file","x:y:z");
  while (1) {
     ncols = fscanf(fp,"%f %f %f",&x, &y, &z);
     if (ncols < 0) break;
     if (nlines < 5) printf("x=%8f, y=%8f, z=%8f
",x,y,z);
     h1->Fill(x);
     ntuple->Fill(x,y,z);
     nlines++;
  }
  printf(" found %d points
",nlines);
  fclose(fp);
  f->Write();
}
```

Canvas and Pad Management

```
{
```

// One of the first actions in a ROOT session is the creation of a Canvas.
// Here we create a Canvas named "c1"

```
gROOT.Reset();
c1 = new TCanvas("c1","Canvas Example",200,10,600,480);
gBenchmark->Start("canvas");
```

// Inside this canvas, we create two pads

```
TPad pad1("pad1","This is pad1",0.05,0.52,0.95,0.97);
TPad pad2("pad2","This is pad2",0.05,0.02,0.95,0.47);
pad1.SetFillColor(11);
pad2.SetFillColor(11);
pad1.Draw();
pad2.Draw();
```

// A pad may contain other pads and graphics objects.
// We set the current pad to pad2.
// Note that the current pad is always highlighted.

```
pad2.cd();
TPad pad21("pad21","First subpad of pad2",0.02,0.05,0.48,0.95,17,3);
TPad pad22("pad22","Second subpad of pad2",0.52,0.05,0.98,0.95,17,3);
pad21.Draw();
pad22.Draw();
```

// We enter some primitives in the created pads and set some attributes

```
pad1.cd();
float xt1 = 0.5;
float yt1 = 0.1;
TText t1(0.5,yt1,"ROOT");
t1.SetTextAlign(22);
t1.SetTextSize(0.05);
t1.Draw();
TLine line1(0.05,0.05,0.80,0.70);
line1.SetLineWidth(8);
line1.SetLineColor(2);
line1.Draw();
line1.DrawLine(0.6,0.1,0.9,0.9);
TLine line2(0.05,0.70,0.50,0.10);
line2.SetLineWidth(4);
line2.SetLineColor(5);
line2.Draw();

pad21.cd();
TText t21(0.05,0.8,"This is pad21");
t21.SetTextSize(0.1);
t21.Draw();
float xp2 = 0.5;
float yp2 = 0.4;
TPavesText paves(0.1,0.1,xp2,yp2);
paves.AddText("This is a PavesText");
paves.AddText("You can add new lines");
paves.AddText("Text formatting is automatic");
paves.SetFillColor(43);
paves.Draw();
pad22.cd();
TText t22(0.05,0.8,"This is pad22");
t22.SetTextSize(0.1);
t22.Draw();
float xlc = 0.01;
float ylc = 0.01;
TPaveLabel label(xlc, ylc, xlc+0.8, ylc+0.1,"This is a PaveLabel");
label.SetFillColor(24);
label.Draw();
```

// Modify object attributes in a loop

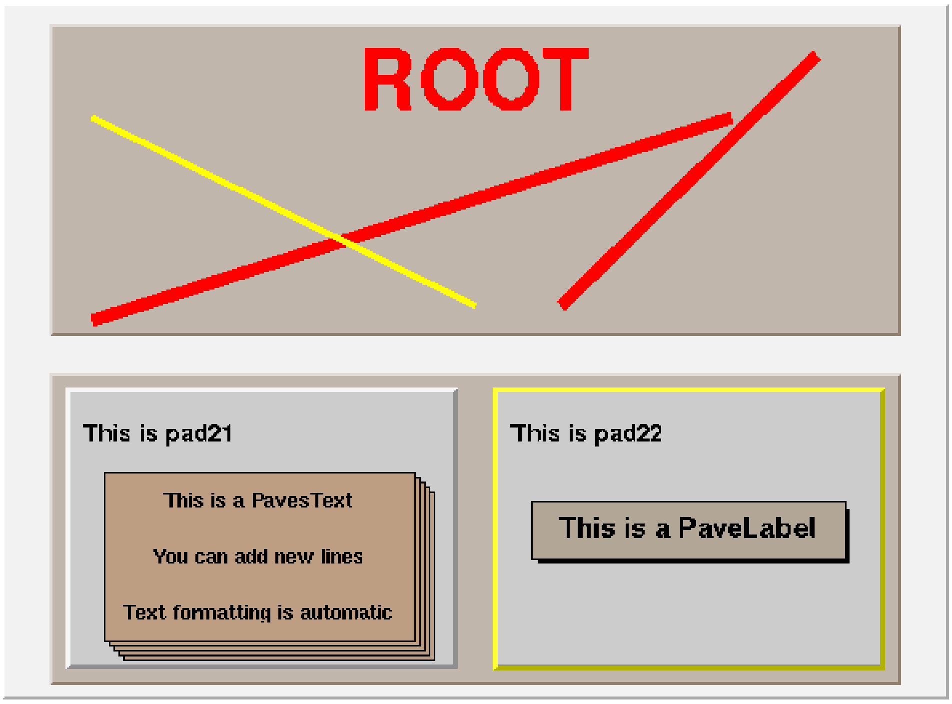
```
Int_t nloops = 50;
float dxp2   = (0.9-xp2)/nloops;
float dyp2   = (0.7-yp2)/nloops;
float dxlc   = (0.1-xlc)/nloops;
float dylc   = (0.4-xlc)/nloops;
float dxt1   = (0.5-xt1)/nloops;
```

```
float dyt1   = (0.8-yt1)/nloops;
float t10    = t1.GetTextSize();
float t1end  = 0.3;
float t1ds   = (t1end - t10)/nloops;
Int_t color  = 0;
for (int i=0;i<nloops;i++) {
   color++;
   color %= 8;
   line1.SetLineColor(color);
   t1.SetTextSize(t10 + t1ds*i);
   t1.SetTextColor(color);
   t1.SetX(xt1+dxt1*i);
   t1.SetY(yt1+dyt1*i);
   pad1.Modified();
   paves.SetX2NDC(xp2+dxp2*i);
   paves.SetY2NDC(yp2+dyp2*i);
   pad21.Modified();
   label.SetX1NDC(xlc+dxlc*i);
   label.SetY1NDC(ylc+dylc*i);
   label.SetX2NDC(xlc+dxlc*i+0.8);
   label.SetY2NDC(ylc+dylc*i+0.2);
   pad22.Modified();
   c1->Update();
}
gBenchmark->Show("canvas");
```

// Try now to point on any object on the screen: pad, text, lines, etc.
// When the cursor points to sensitive areas in an object, the cursor
// shape changes and suggests the type of action that can be applied.

// For example, one can move, grow,shrink a pad.
// A text can be moved.
// A line can be moved or its end points can be modified.
// One can move, grow and shrink PaveLabels and PavesText.
// Point to an object and click the right mouse button to change attributes.
// Try to change the canvas size.
// In the canvas "File" menu, select the option "Print" to produce
// a PostScript file with a copy of the canvas.

```
}
```



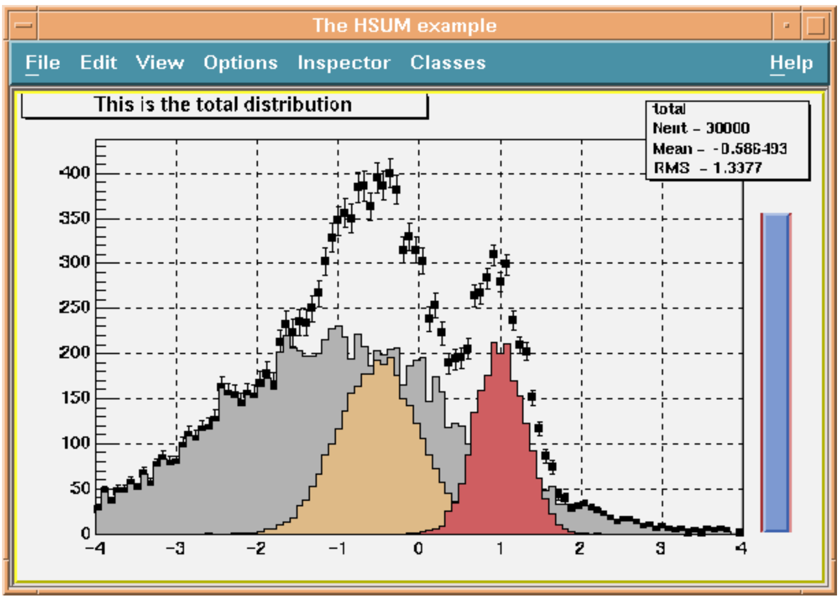
An Example Creating Histograms/Ntuples on a File

```
{
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
//*-*
//*-* This program creates :
//*-* - a one dimensional histogram
//*-* - a two dimensional histogram
//*-* - a profile histogram
//*-* - a memory-resident ntuple
//*-*
//*-* These objects are filled with some random numbers and saved on a file.
//*-*
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

 gROOT->Reset();

// Create a new canvas.

 c1 = new TCanvas("c1","Dynamic Filling Example",200,10,700,500);
 c1->SetFillColor(42);
 c1->GetFrame()->SetFillColor(21);
 c1->GetFrame()->SetBorderSize(6);
 c1->GetFrame()->SetBorderMode(-1);

// Create a new ROOT binary machine independent file.
// Note that this file may contain any kind of ROOT objects, histograms,
// pictures, graphics objects, detector geometries, tracks, events, etc..
// This file is now becoming the current directory.

 TFile *hfile = gROOT->FindObject("hsimple.root"); if (hfile) hfile->Close();
 hfile = new TFile("hsimple.root","RECREATE","Demo ROOT file with histograms");

// Create some histograms, a profile histogram and an ntuple

 hpx    = new TH1F("hpx","This is the px distribution",100,-4,4);
 hpxpy  = new TH2F("hpxpy","py vs px",40,-4,4,40,-4,4);
 hprof  = new TProfile("hprof","Profile of pz versus px",100,-4,4,0,20);
 ntuple = new TNtuple("ntuple","Demo ntuple","px:py:pz:random:i");

// Set canvas/frame attributes (save old attributes)

 hpx->SetFillColor(48);
 gBenchmark->Start("hsimple");

// Fill histograms randomly

 gRandom->SetSeed();
 Float_t px, py, pz;
 const Int_t kUPDATE = 1000;
 for ( Int_t i=0; i<25000; i++) {
    gRandom->Rannor(px,py);
    pz = px*px + py*py;
    Float_t random = gRandom->Rndm(1);
    hpx->Fill(px);
    hpxpy->Fill(px,py);
    hprof->Fill(px,pz);
    ntuple->Fill(px,py,pz,random,i);
    if (!i%kUPDATE) {
       if (i == kUPDATE) hpx->Draw();
       c1->Modified();
       c1->Update();
    }
 }
 gBenchmark->Show("hsimple");

// Save all objects in this file

 hpx->SetFillColor(0);
 hfile->Write();
 hpx->SetFillColor(48);

// Note that the file is automatically close when application terminates
```

```
// or when the file destructor is called.

}
```

Filling several histograms and some graphics options

```
{
// Simple example illustrating how to use the C++ interpreter
// to fill histograms in a loop and show the graphics results
 gROOT->Reset();
 c1 = new TCanvas("c1","The HSUM example",200,10,600,400);
 c1->SetGrid();
 gBenchmark->Start("hsum");

// Create some histograms.
 total  = new TH1F("total","This is the total distribution",100,-4,4);
 main   = new TH1F("main","Main contributor",100,-4,4);
 s1     = new TH1F("s1","This is the first signal",100,-4,4);
 s2     = new TH1F("s2","This is the second signal",100,-4,4);
 total->Sumw2();   // this makes sure that the sum of squares of weights will be stored
 total->SetMarkerStyle(21);
 total->SetMarkerSize(0.7);
 main->SetFillColor(16);
 s1->SetFillColor(42);
 s2->SetFillColor(46);
 TSlider *slider = 0;

// Fill histograms randomly
 gRandom->SetSeed();
 const Int_t kUPDATE = 500;
 Float_t xs1, xs2, xmain;
 for ( Int_t i=0; i<10000; i++) {
    xmain = gRandom->Gaus(-1,1.5);
    xs1   = gRandom->Gaus(-0.5,0.5);
    xs2   = gRandom->Gaus(1,0.3);
    main->Fill(xmain);
    s1->Fill(xs1,0.3);
    s2->Fill(xs2,0.2);
    total->Fill(xmain);
    total->Fill(xs1,0.3);
    total->Fill(xs2,0.2);
    if (i && (i%kUPDATE) == 0) {
       if (i == kUPDATE) {
          total->Draw("e1p");
          main->Draw("same");
          s1->Draw("same");
          s2->Draw("same");
          c1->Update();
          slider = new TSlider("slider","test",4.2,0,4.6,total->GetMaximum(),38);
          slider->SetFillColor(46);
       }
       if (slider) slider->SetRange(0,Float_t(i)/10000.);
       c1->Modified();
       c1->Update();
    }
 }
 slider->SetRange(0,1);
 c1->Modified();
 gBenchmark->Show("hsum");
}
```



Histograms with Random Numbers from a Function

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","The FillRandom example",200,10,700,900);
  c1->SetFillColor(18);
  pad1 = new TPad("pad1","The pad with the function",0.05,0.50,0.95,0.95,21);
  pad2 = new TPad("pad2","The pad with the histogram",0.05,0.05,0.95,0.45,21);
  pad1->Draw();
  pad2->Draw();
  pad1->cd();
  gBenchmark->Start("fillrandom");
```

// A function (any dimension) or a formula may reference
// an already defined formula

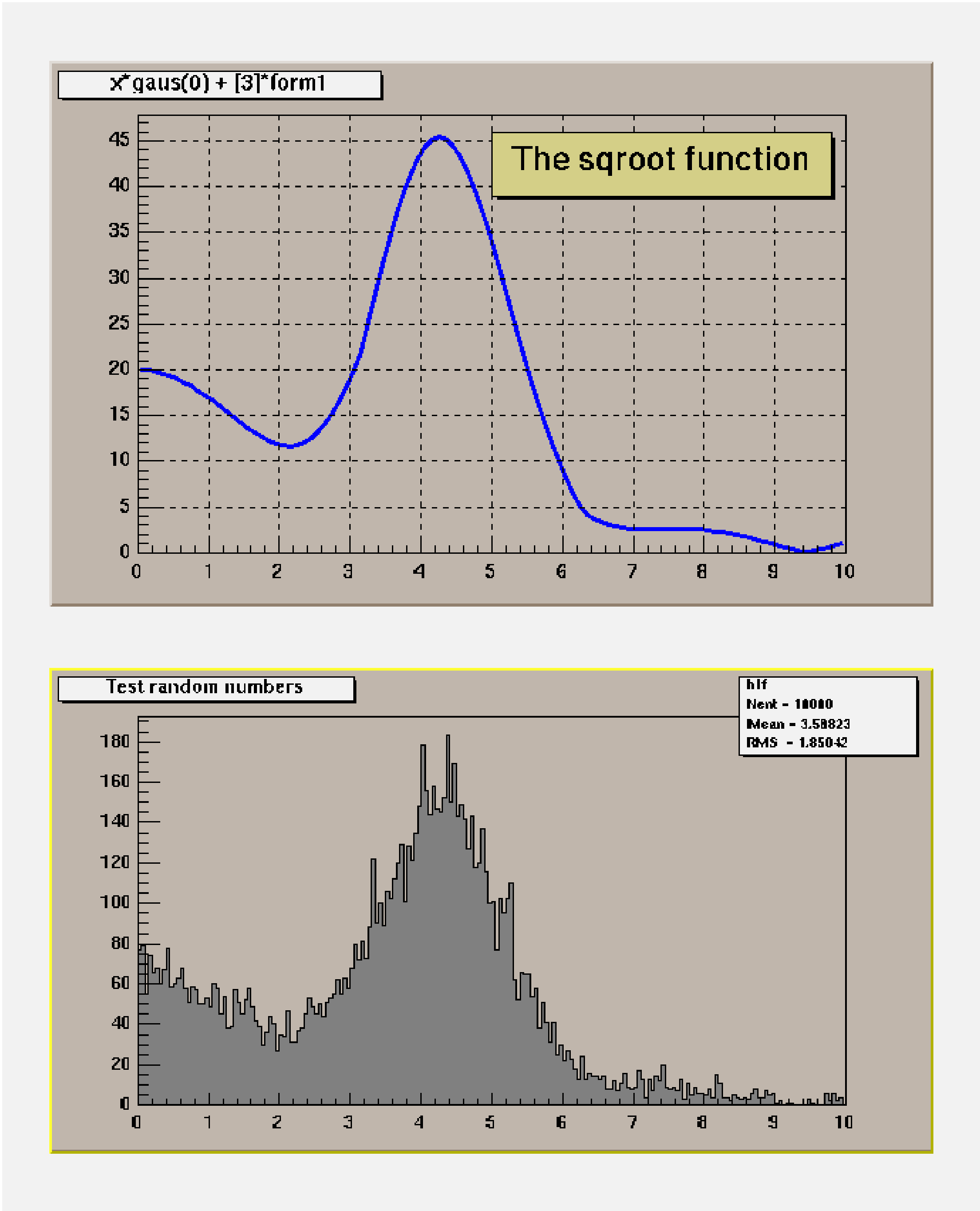
```
  form1 = new TFormula("form1","abs(sin(x)/x)");
  sqroot = new TF1("sqroot","x*gaus(0) + [3]*form1",0,10);
  sqroot->SetParameters(10,4,1,20);
  pad1->SetGridx();
  pad1->SetGridy();
  pad1->GetFrame()->SetFillColor(42);
  pad1->GetFrame()->SetBorderMode(-1);
  pad1->GetFrame()->SetBorderSize(5);
  sqroot->SetLineColor(4);
  sqroot->SetLineWidth(6);
  sqroot->Draw();
  lfunction = new TPaveLabel(5,39,9.8,46,"The sqroot function");
  lfunction->SetFillColor(41);
  lfunction->Draw();
  c1->Update();
```

// Create a one dimensional histogram (one float per bin)
// and fill it following the distribution in function sqroot.

```
  pad2->cd();
  pad2->GetFrame()->SetFillColor(42);
  pad2->GetFrame()->SetBorderMode(-1);
  pad2->GetFrame()->SetBorderSize(5);
  h1f = new TH1F("h1f","Test random numbers",200,0,10);
  h1f->SetFillColor(45);
  h1f->FillRandom("sqroot",10000);
  h1f->Draw();
  c1->Update();
```

// Open a ROOT file and save the formula, function and histogram

```
  TFile myfile("fillrandom.root","RECREATE");
  form1->Write();
  sqroot->Write();
  h1f->Write();
  myfile.Close();
  gBenchmark->Show("fillrandom");
}
```


x*gaus(0) + [3]*form1
The sqroot function
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
Test random numbers
hlf
Nent = 10000
Mean = 3.58823
RMS = 1.85042
180
160
140
120
100
80
60
40
20
0

Simple Formula and Functions

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","Example with Formula",200,10,700,500);
```

// We create a formula object and compute the value of this formula
// for two different values of the x variable.

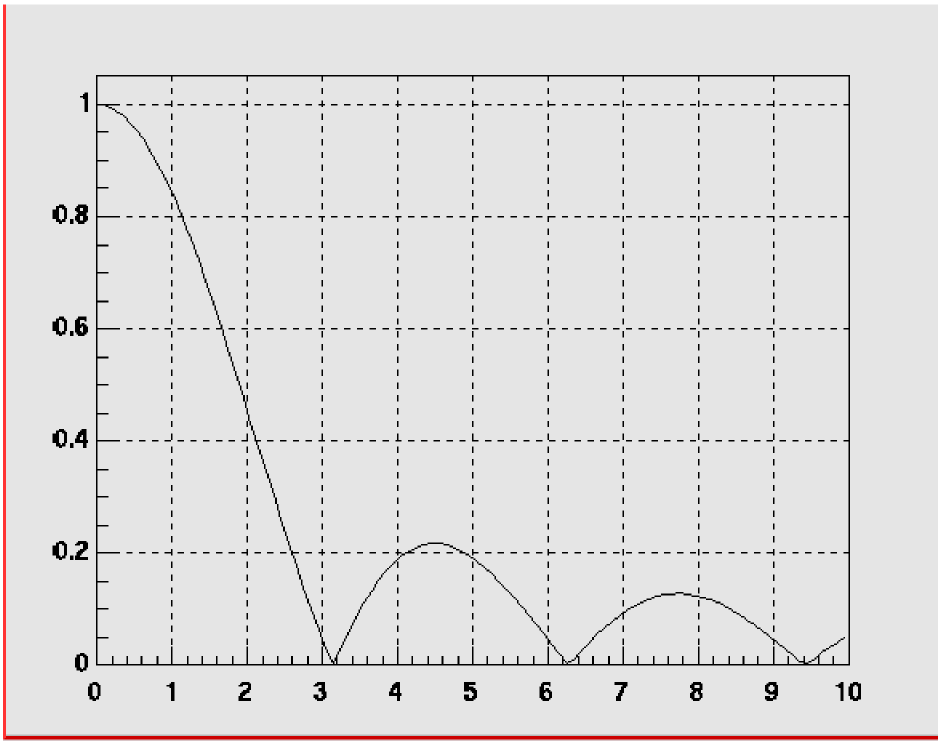
```
  form1 = new TFormula("form1","sqrt(abs(x))");
  form1->Eval(2);
  form1->Eval(-45);
```

// Create a one dimensional function and draw it

```
  fun1 = new TF1("fun1","abs(sin(x)/x)",0,10);
  c1->SetGridx();
  c1->SetGridy();
  fun1->Draw();
  c1->Update();
```

// Before leaving this demo, we print the list of objects known to ROOT

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



A Simple Fitting Example

```
{
  gROOT->Reset();
  c1 = new TCanvas("c1","The Fit Canvas",200,10,700,500);
  c1->SetGridx();
  c1->SetGridy();
  c1->GetFrame()->SetFillColor(21);
  c1->GetFrame()->SetBorderMode(-1);
  c1->GetFrame()->SetBorderSize(5);
  gBenchmark->Start("fit1");
```

// We connect the ROOT file generated in a previous tutorial
// (see Filling histograms with random numbers from a function)

```
  TFile fill("fillrandom.root");
```

// The function "ls()" lists the directory contents of this file

```
  fill.ls();
```

// Get object "sqroot" from the file. Undefined objects are searched
// for using gROOT->FindObject("xxx"), e.g.:
// TF1 *sqroot = (TF1*) gROOT.FindObject("sqroot")

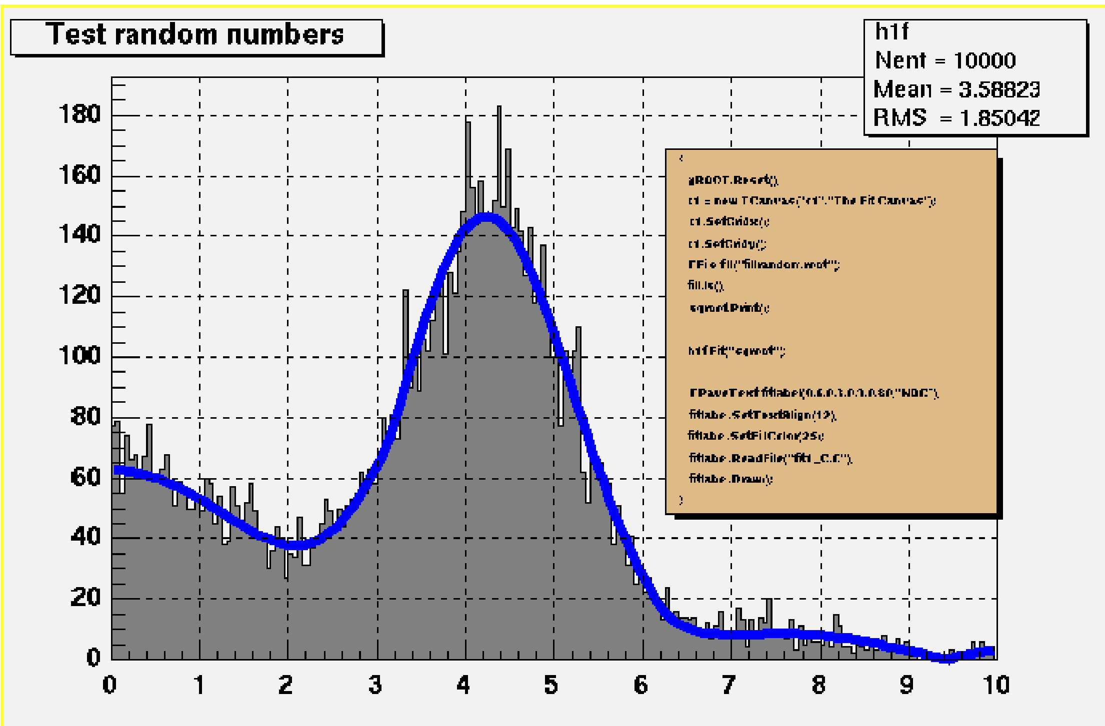
```
  sqroot->Print();
```

// Now fit histogram h1f with the function sqroot

```
  h1f->SetFillColor(45);
  h1f->Fit("sqroot");
```

// We now annotate the picture by creating a PaveText object
// and displaying the list of commands in this macro

```
  fitlabel = new TPaveText(0.6,0.3,0.9,0.80,"NDC");
  fitlabel->SetTextAlign(12);
  fitlabel->SetFillColor(42);
  fitlabel->ReadFile("fit1_C.C");
  fitlabel->Draw();
  c1->Update();
  gBenchmark->Show("fit1");
}
```



Example of a program to fit non-equidistant data points

// Example of a program to fit non-equidistant data points
// ===

// The fitting function fcn is a simple chisquare function
// The data consists of 5 data points (arrays x,y,z) + the errors in errorsz
// More details on the various functions or parameters for these functions
// can be obtained in an interactive ROOT session with:
// Root > TMinuit *minuit = new TMinuit(10);
// Root > minuit->mnhelp("*",0) to see the list of possible keywords
// Root > minuit->mnhelp("SET",0) explains most parameters

```
Float_t z[5],x[5],y[5],errorz[5];
```

//____

```
void fcn(Int_t &npar, Double_t *gin, Double_t &f, Double_t *par, Int_t iflag)
{
  const Int_t nbins = 5;
  Int_t i;
```

//calculate chisquare

```
  Double_t chisq = 0;
  Double_t delta;
  for (i=0;i<nbins; i++) {
    delta  = (z[i]-func(x[i],y[i],par))/errorz[i];
    chisq += delta*delta;
  }
  f = chisq;
}
```

//____

```
Double_t func(float x,float y,Double_t *par)
{
Double_t value=( (par[0]*par[0])/(x*x)-1)/ ( par[1]+par[2]*y-par[3]*y*y);
return value;
}
```

//____

```
void Ifit()
{
```

// The z values

```
z[0]=1;
z[1]=0.96;
z[2]=0.89;
z[3]=0.85;
z[4]=0.78;
```

// The errors on z values

```
       Float_t error = 0.01;
errorz[0]=error;
errorz[1]=error;
errorz[2]=error;
errorz[3]=error;
errorz[4]=error;
```

// the x values

```
x[0]=1.5751;
x[1]=1.5825;
x[2]=1.6069;
x[3]=1.6339;
x[4]=1.6706;
```

// the y values

```
y[0]=1.0642;
y[1]=0.97685;
y[2]=1.13168;
y[3]=1.128654;
y[4]=1.44016;
  TMinuit *gMinuit = new TMinuit(5);  //initialize TMinuit with a maximum of 5 params
  gMinuit->SetFCN(fcn);
  Double_t arglist[10];
  Int_t ierflg = 0;
  arglist[0] = 1;
  gMinuit->mnexcm("SET ERR", arglist ,1,ierflg);
```

// Set starting values and step sizes for parameters

```
  static Double_t vstart[4] = {3, 1 , 0.1 , 0.01};
  static Double_t step[4] = {0.1 , 0.1 , 0.01 , 0.001};
  gMinuit->mnparm(0, "a1", vstart[0], step[0], 0,0,ierflg);
  gMinuit->mnparm(1, "a2", vstart[1], step[1], 0,0,ierflg);
  gMinuit->mnparm(2, "a3", vstart[2], step[2], 0,0,ierflg);
  gMinuit->mnparm(3, "a4", vstart[3], step[3], 0,0,ierflg);
```

// Now ready for minimization step

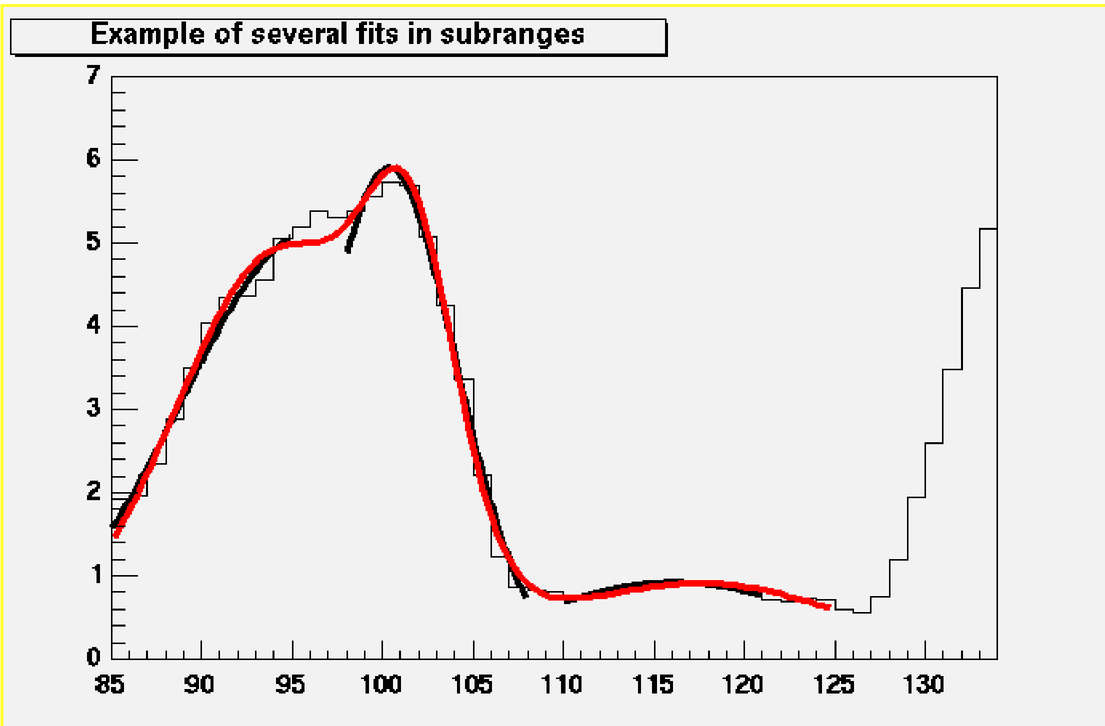
```
  arglist[0] = 500;
  arglist[1] = 1.;
  gMinuit->mnexcm("MIGRAD", arglist ,2,ierflg);
```

// Print results

```
  Double_t amin,edm,errdef;
  Int_t nvpar,nparx,icstat;
  gMinuit->mnstat(amin,edm,errdef,nvpar,nparx,icstat);
  gMinuit->mnprin(3,amin);
}
```

How to fit in a sub-range of an histogram

```
{
// Example showing how to fit in a sub-range of an histogram
// An histogram is created and filled with the bin contents and errors
// defined in the table below.
// 3 gaussians are fitted in sub-ranges of this histogram.
// A new function (a sum of 3 gaussians) is fitted on another subrange
// Note that when fitting simple functions, such as gaussians, the initial
// values of parameters are automatically computed by ROOT.
// In the more complicated case of the sum of 3 gaussians, the initial values
// of parameters must be given. In this particular case, the initial values
// are taken from the result of the individual fits.
   gROOT->Reset();
   const Int_t np = 49;
   Float_t x[np] = {1.913521, 1.953769, 2.347435, 2.883654, 3.493567,
                    4.047560, 4.337210, 4.364347, 4.563004, 5.054247,
                    5.194183, 5.380521, 5.303213, 5.384578, 5.563983,
                    5.728500, 5.685752, 5.080029, 4.251809, 3.372246,
                    2.207432, 1.227541, 0.8597788,0.8220503,0.8046592,
                    0.7684097,0.7469761,0.8019787,0.8362375,0.8744895,
                    0.9143721,0.9462768,0.9285364,0.8954604,0.8410891,
                    0.7853871,0.7100883,0.6938808,0.7363682,0.7032954,
                    0.6029015,0.5600163,0.7477068,1.188785, 1.938228,
                    2.602717, 3.472962, 4.465014, 5.177035};
   h = new TH1F("g1","Example of several fits in subranges",np,85,134);
   h->SetMaximum(7);
   for (int i=0;i<np;i++) {
      h->SetBinContent(i+1,x[i]);
   }
   Double_t par[9];
   g1    = new TF1("g1","gaus",85,95);
   g2    = new TF1("g2","gaus",98,108);
   g3    = new TF1("g3","gaus",110,121);
   total = new TF1("total","gaus(0)+gaus(3)+gaus(6)",85,125);
   total->SetLineColor(2);
   h->Fit("g1","R0");
   h->Fit("g2","R0+");
   h->Fit("g3","R0+");
   g1->GetParameters(&par[0]);
   g2->GetParameters(&par[3]);
   g3->GetParameters(&par[6]);
   total->SetParameters(par);
   h->Fit("total","R0+");
   h->Draw();
}
```



Fitting with a user-defined function

// This macro gets in memory an histogram from a root file
// and fits a user defined function.
// Note that a user defined function must always be defined
// as in this example:
// - first parameter: array of variables (in this example only 1-dimension)
// - second parameter: array of parameters
// Note also that in case of user defined functions, one must set
// an initial value for each parameter.

```
Double_t fitf(Double_t *x, Double_t *par)
{
  Double_t arg = 0;
  if (par[2]) arg = (x[0] - par[1])/par[2];
  Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
  return fitval;
}
void myfit()
{
  TFile *f = new TFile("hsimple.root");
  TCanvas *c1 = new TCanvas("c1","the fit canvas",500,400);
  TH1F *hpx = (TH1F*)f->Get("hpx");
```

// Creates a Root function based on function fitf above

```
  TF1 *func = new TF1("fitf",fitf,-2,2,3);
```

// Sets initial values and parameter names

```
  func->SetParameters(100,0,1);
  func->SetParNames("Constant","Mean_value","Sigma");
```

// Fit histogram in range defined by function

```
  hpx->Fit("fitf","r");
```

// Gets integral of function between fit limits

```
  printf("Integral of function = %g
",func->Integral(-2,2));
}
```

Drawing Options for 1D Histograms

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","Histogram Drawing Options",200,10,700,900);
  pad1 = new TPad("pad1","The pad with the function",0.03,0.62,0.50,0.92,21);
  pad2 = new TPad("pad2","The pad with the histogram",0.51,0.62,0.98,0.92,21);
  pad3 = new TPad("pad3","The pad with the histogram",0.03,0.02,0.97,0.57,21);
  pad1->Draw();
  pad2->Draw();
  pad3->Draw();
```

// We connect the ROOT file generated in a previous tutorial
// see An example creating/filling/saving histograms/ntuples on file

```
  TFile example("hsimple.root");
  example.ls();
```

// Draw a global picture title

```
  title = new TPaveLabel(0.1,0.94,0.9,0.98,
                   "Drawing options for one dimensional histograms");
  title->SetFillColor(16);
  title->Draw();
```

// Draw histogram hpx in first pad with the default option.

```
  pad1->cd();
  pad1->GetFrame()->SetFillColor(18);
  hpx->SetFillColor(45);
  hpx->DrawCopy();
  label1 = new TPaveLabel(-3.5,700,-1,800,"Default option");
  label1->SetFillColor(42);
  label1->Draw();
```

// Draw hpx as a lego. Clicking on the lego area will show
// a "transparent cube" to guide you rotating the lego in real time.

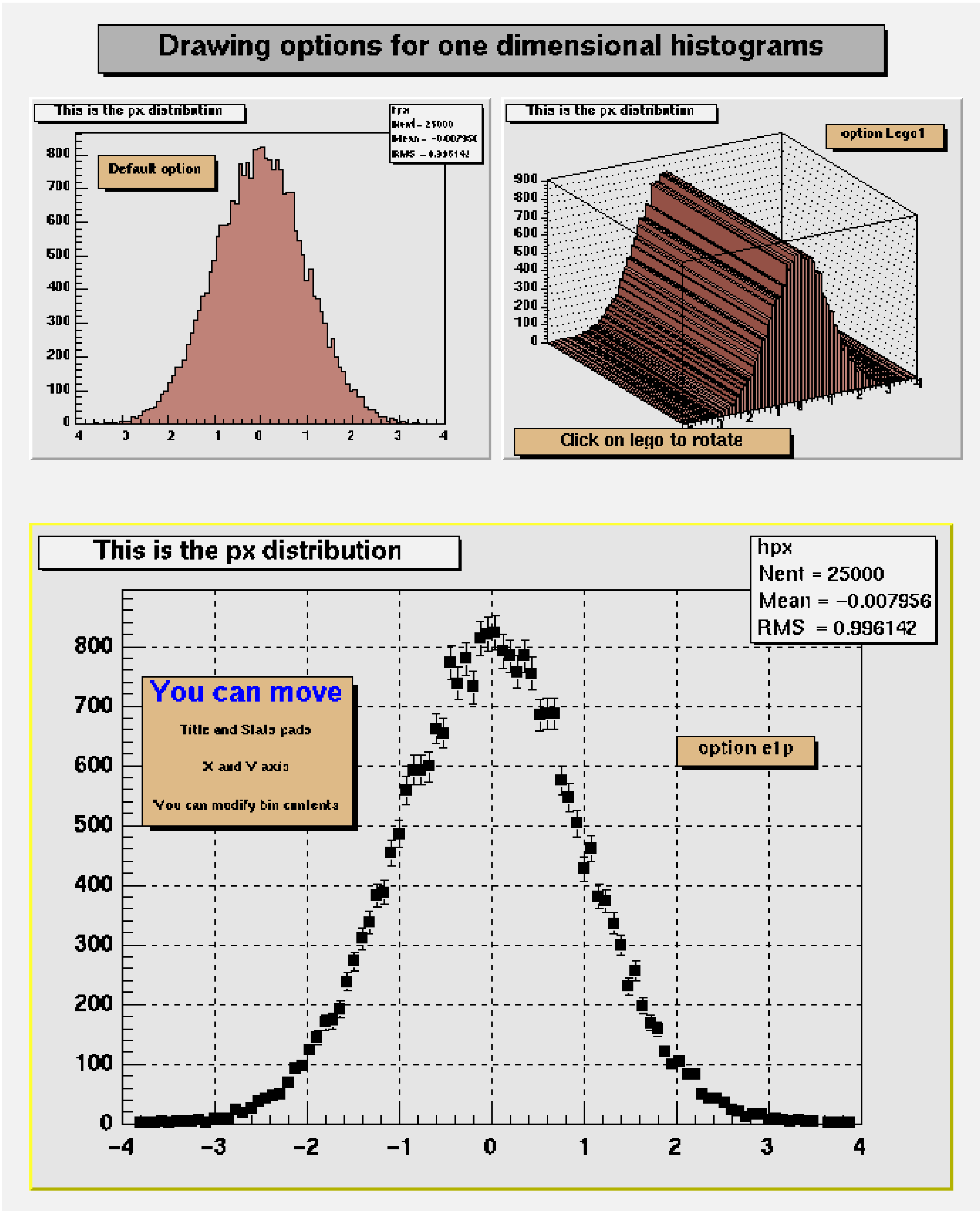
```
  pad2->cd();
  hpx->DrawCopy("lego1");
  label2 = new TPaveLabel(0.4,0.8,0.9,0.95,"option Lego1");
  label2->SetFillColor(42);
  label2->Draw();
  label2a = new TPaveLabel(-0.93,-1.08,0.25,-0.92,"Click on lego to rotate");
  label2a->SetFillColor(42);
  label2a->Draw();
```

// Draw hpx with its errors and a marker.

```
  pad3->cd();
  pad3->SetGridx();
  pad3->SetGridy();
  pad3->GetFrame()->SetFillColor(18);
  hpx->SetMarkerStyle(21);
  hpx->Draw("e1p");
  label3 = new TPaveLabel(2,600,3.5,650,"option e1p");
  label3->SetFillColor(42);
  label3->Draw();
```

// The following illustrates how to add comments using a PaveText.
// Attributes of text/lines/boxes added to a PaveText can be modified.
// The AddText function returns a pointer to the added object.

```
  pave = new TPaveText(-3.78,500,-1.5,750);
  pave->SetFillColor(42);
  TText *t1=pave->AddText("You can move");
  t1->SetTextColor(4);
  t1->SetTextSize(0.05);
  pave->AddText("Title and Stats pads");
  pave->AddText("X and Y axis");
  pave->AddText("You can modify bin contents");
  pave->Draw();
  c1->Update();
}
```



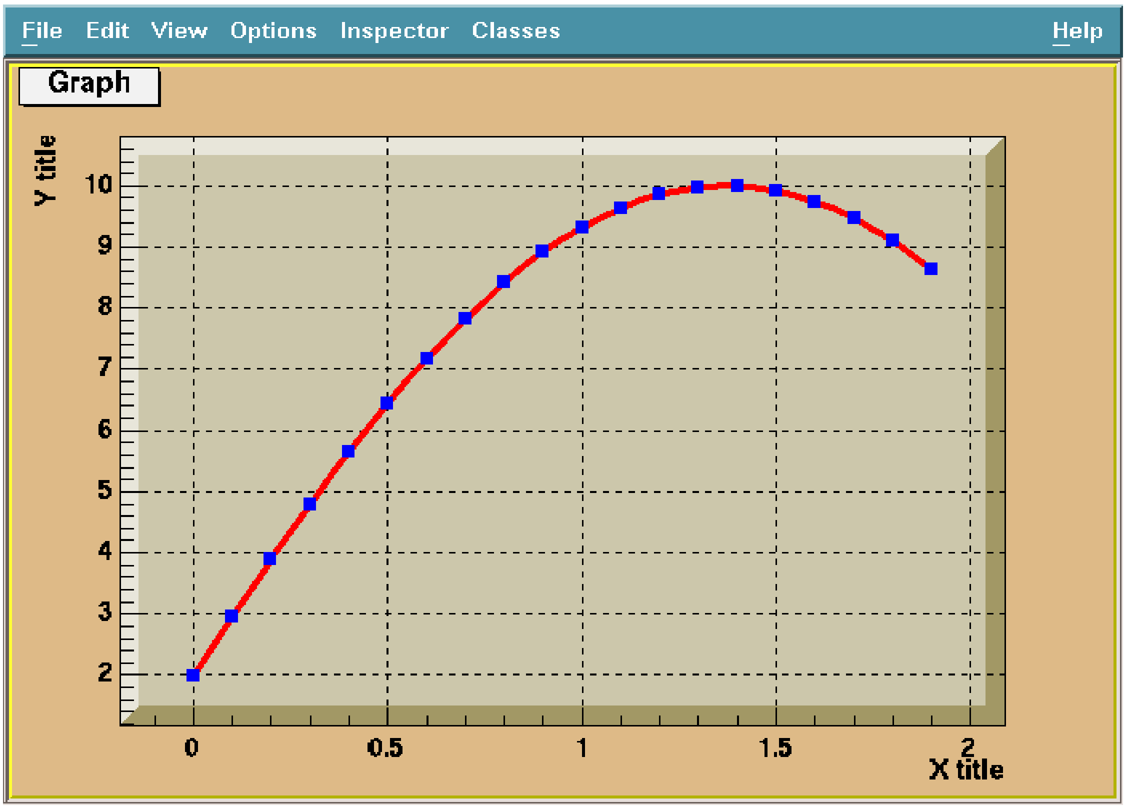
Drawing options for one dimensional histograms
This is the px distribution
Default option
This is the px distribution
option Lego1
Click on lego to rotate
This is the px distribution
hpx
Nent = 25000
Mean = −0.007956
RMS = 0.996142
You can move
Title and Stats pads
X and Y axis
You can modify bin contents
option e1p

A simple Graph with axis titles

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","A Simple Graph Example",200,10,700,500);
  c1->SetFillColor(42);
  c1->SetGridx();
  c1->SetGridy();
  c1->GetFrame()->SetFillColor(21);
  c1->GetFrame()->SetBorderSize(12);
  Int_t n = 20;
  Float_t x[n], y[n];
  for (Int_t i=0;i<n;i++) {
    x[i] = i*0.1;
    y[i] = 10*sin(x[i]+0.2);
    printf(" i %i %f %f
",i,x[i],y[i]);
  }
  gr = new TGraph(n,x,y);
  gr->SetFillColor(19);
  gr->SetLineColor(2);
  gr->SetLineWidth(4);
  gr->SetMarkerColor(4);
  gr->SetMarkerStyle(21);
  gr->Draw("ACP");
```

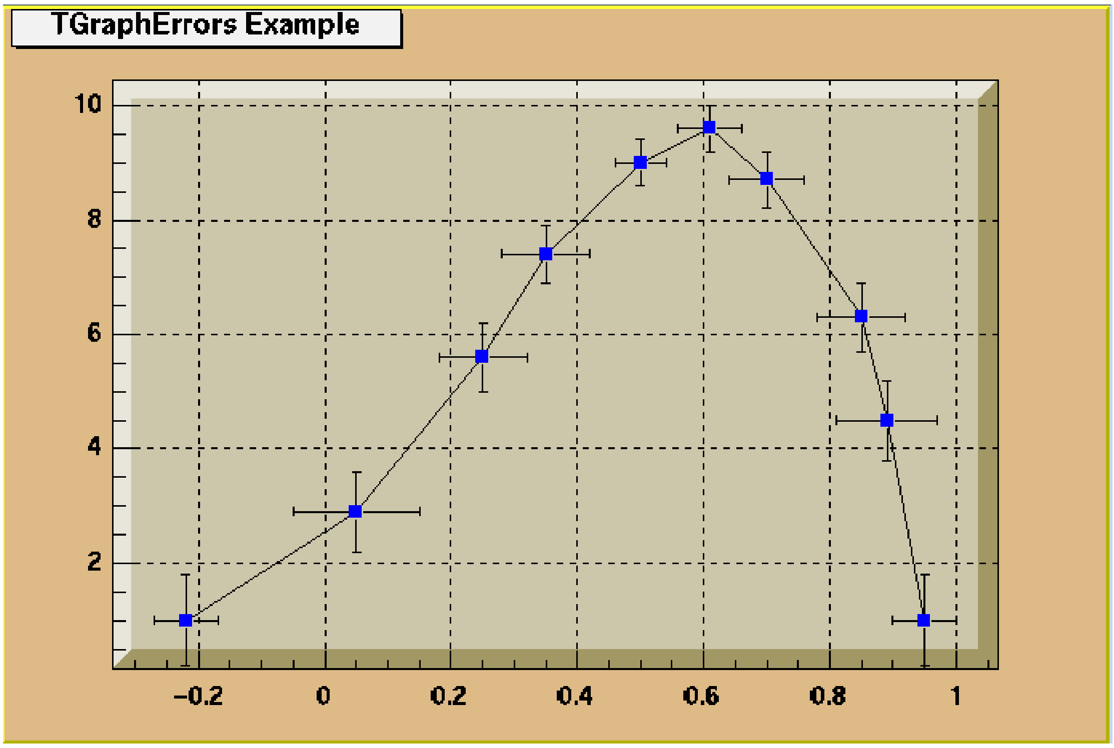
//Add axis titles.
//A graph is drawn using the services of the TH1F histogram class.
//The histogram is created by TGraph::Paint.
//TGraph::Paint is called by TCanvas::Update. This function is called by default
//when typing <CR> at the keyboard. In a macro, one must force TCanvas::Update.

```
  c1->Update();
  gr->GetHistogram()->SetXTitle("X title");
  gr->GetHistogram()->SetYTitle("Y title");
}
```



Examples of a Graph with error bars

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","A Simple Graph with error bars",200,10,700,500);
  c1->SetFillColor(42);
  c1->SetGrid();
  c1->GetFrame()->SetFillColor(21);
  c1->GetFrame()->SetBorderSize(12);
  Int_t n = 10;
  Float_t x[n]  = {-0.22, 0.05, 0.25, 0.35, 0.5, 0.61,0.7,0.85,0.89,0.95};
  Float_t y[n]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
  Float_t ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
  Float_t ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
  gr = new TGraphErrors(n,x,y,ex,ey);
  gr->SetTitle("TGraphErrors Example");
  gr->SetMarkerColor(4);
  gr->SetMarkerStyle(21);
  gr->Draw("ALP");
  c1->Update();
}
```



Surfaces drawing options

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","Surfaces Drawing Options",200,10,700,900);
  c1->SetFillColor(42);
  gStyle->SetFrameFillColor(42);
  title = new TPaveText(.2,0.96,.8,.995);
  title->SetFillColor(33);
  title->AddText("Examples of Surface options");
  title->Draw();
  pad1 = new TPad("pad1","Gouraud shading",0.03,0.50,0.98,0.95,21);
  pad2 = new TPad("pad2","Color mesh",0.03,0.02,0.98,0.48,21);
  pad1->Draw();
  pad2->Draw();
```

// We generate a 2-D function

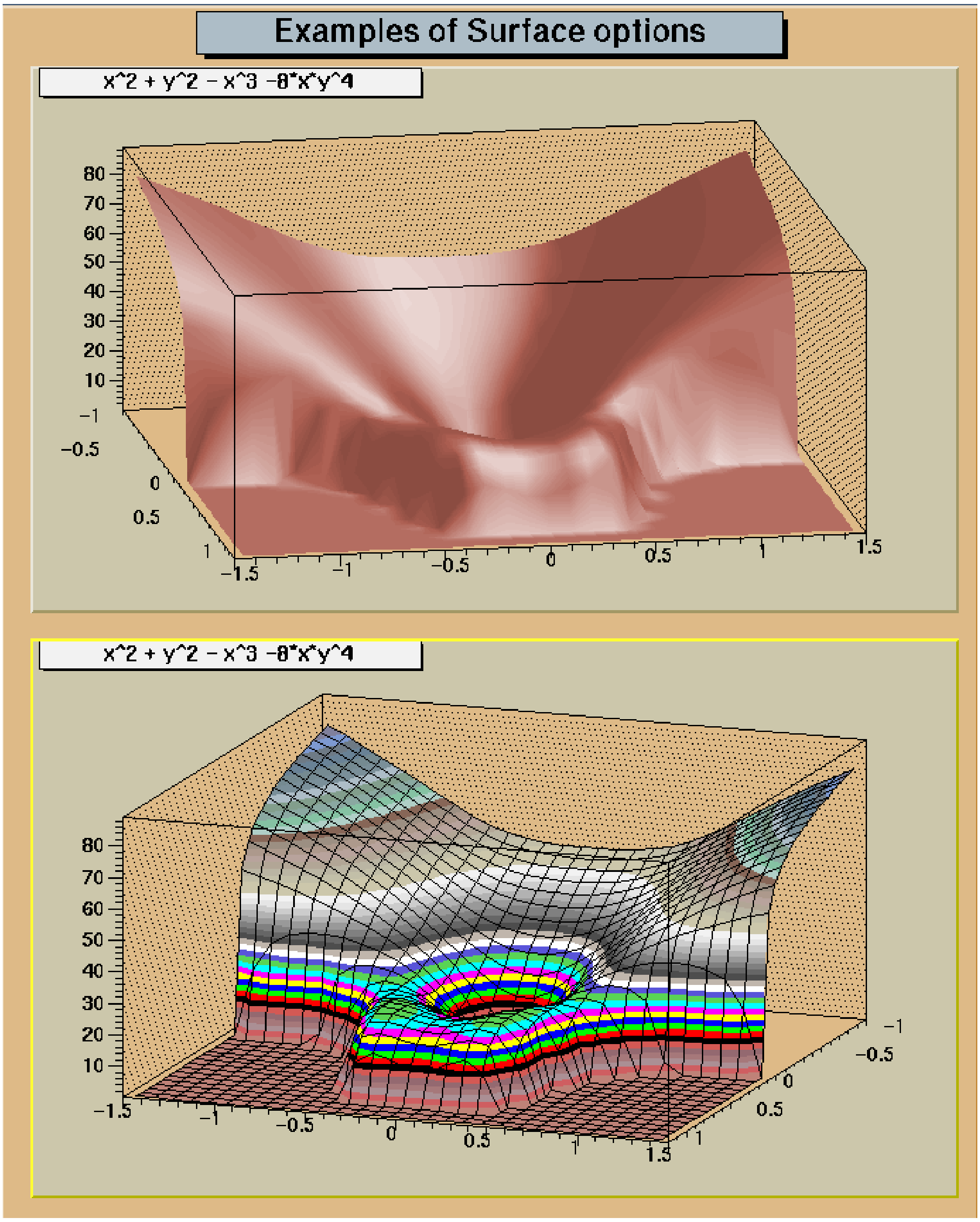
```
  TF2 *f2 = new TF2("f2","x^2 + y^2 - x^3 -8*x*y^4",-1,1.2,-1.5,1.5);
  f2->SetContour(48);
  f2->SetFillColor(45);
```

// Draw this function in pad1 with Gouraud shading option

```
  pad1->cd();
  pad1->SetPhi(-80);
  pad1->SetLogz();
  f2->Draw("surf4");
```

// Draw this function in pad2 with color mesh option

```
  pad2->cd();
  pad2->SetTheta(25);
  pad2->SetPhi(-110);
  pad2->SetLogz();
  f2->Draw("surf1");
}
```


Examples of Surface options
x^2 + y^2 - x^3 -8*x*y^4
x^2 + y^2 - x^3 -8*x*y^4

Examples of 3-D Polymarkers

```
{
   gROOT->Reset();
   gBenchmark->Start("tornado");
   double PI = 3.141592653;
   int d = 16;
   int numberOfPoints=200;
   int numberOfCircles=40;
```

// create and open a canvas

```
   sky = new TCanvas( "sky", "Tornado", 300, 10, 700, 500 );
   sky->SetFillColor(14);
```

// creating view

```
   TView *view = new TView(1);
   float range = numberOfCircles*d;
   view->SetRange( 0, 0, 0, 4.0*range, 2.0*range, range );
   for( int j = d; j < numberOfCircles*d; j += d ) {
```

// create a PolyMarker3D

```
       TPolyMarker3D *pm3d = new TPolyMarker3D( numberOfPoints );
       float x, y, z;
```

// set points

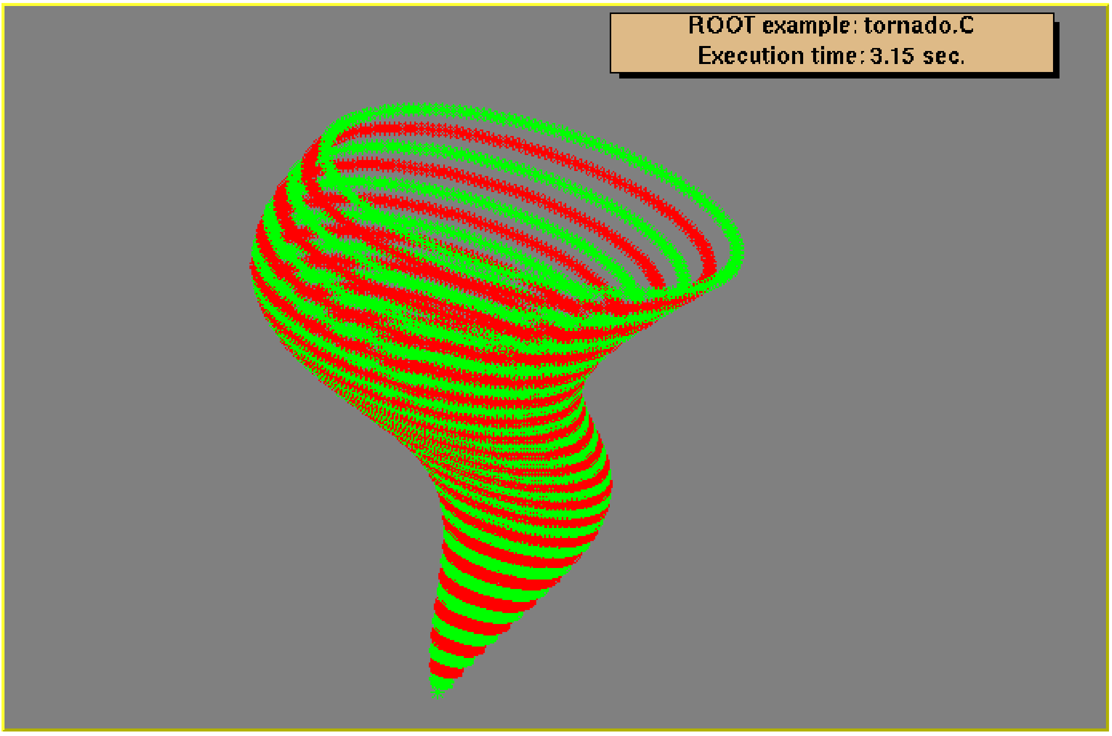
```
       for( int i = 1; i < numberOfPoints; i++ ) {
           float csin = sin(2*PI / (double)numberOfPoints  * (double)i) + 1;
           float ccos = cos(2*PI / (double)numberOfPoints  * (double)i) + 1;
           float esin = sin(2*PI / (double)(numberOfCircles*d) * (double)j) + 1;
           x = j * ( csin + esin );
           y = j * ccos;
           z = j;
           pm3d->SetPoint( i, x, y, z );
       }
```

// set marker size, color & style

```
       pm3d->SetMarkerSize( 1 );
       pm3d->SetMarkerColor( 2 + ( d == ( j & d ) ) );
       pm3d->SetMarkerStyle( 3 );
```

//draw

```
       pm3d->Draw();
   }
   char timeStr[60];
   gBenchmark->Show("tornado");
   Float_t ct = gBenchmark->GetCpuTime("tornado");
   sprintf( timeStr, "Execution time: %g sec.", ct);
   TPaveText *text = new TPaveText( 0.1, 0.81, 0.9, 0.97 );
   text->SetFillColor( 42 );
   text->AddText("ROOT example: tornado.C");
   text->AddText(timeStr);
   text.Draw();
   sky->Update();
}
```


ROOT example: tornado.C
Execution time: 3.15 sec.

The Geometry shapes

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","Geometry Shapes",200,10,700,500);
```

// Define some volumes

```
  brik = new TBRIK("BRIK","BRIK","void",200,150,150);
  trd1 = new TTRD1("TRD1","TRD1","void",200,50,100,100);
  trd2 = new TTRD2("TRD2","TRD2","void",200,50,200,50,100);
  trap = new TTRAP("TRAP","TRAP","void",190,0,0,60,40,90,15,120,80,180,15);
  para = new TPARA("PARA","PARA","void",100,200,200,15,30,30);
  gtra = new TGTRA("GTRA","GTRA","void",390,0,0,20,60,40,90,15,120,80,180,15);
  tube = new TTUBE("TUBE","TUBE","void",150,200,400);
  tubs = new TTUBS("TUBS","TUBS","void",80,100,100,90,235);
  cone = new TCONE("CONE","CONE","void",50,70,120,150,100);
  cons = new TCONS("CONS","CONS","void",50,100,100,200,300,90,270);
  pcon = new TPCON("PCON","PCON","void",180,270,4);
  pcon->DefineSection(0,50,100,-200);
  pcon->DefineSection(1,50,80,-50);
  pcon->DefineSection(2,50,80,50);
  pcon->DefineSection(3,50,100,200);
  pgon = new TPGON("PGON","PGON","void",180,270,8,4);
  pgon->DefineSection(0,50,100,-200);
  pgon->DefineSection(1,50,80,-50);
  pgon->DefineSection(2,50,80,50);
  pgon->DefineSection(3,50,100,200);
```

// Set shapes attributes

```
  brik->SetLineColor(1);
  trd1->SetLineColor(2);
  trd2->SetLineColor(3);
  trap->SetLineColor(4);
  para->SetLineColor(5);
  gtra->SetLineColor(7);
  tube->SetLineColor(6);
  tubs->SetLineColor(7);
  cone->SetLineColor(2);
  cons->SetLineColor(3);
  pcon->SetLineColor(6);
  pgon->SetLineColor(2);
```

// Build the geometry hierarchy

```
  node1 = new TNode("NODE1","NODE1","BRIK");
  node1->cd();
  node2  = new TNode("NODE2","NODE2","TRD1",0,0,-1000);
  node3  = new TNode("NODE3","NODE3","TRD2",0,0,1000);
  node4  = new TNode("NODE4","NODE4","TRAP",0,-1000,0);
  node5  = new TNode("NODE5","NODE5","PARA",0,1000,0);
  node6  = new TNode("NODE6","NODE6","TUBE",-1000,0,0);
  node7  = new TNode("NODE7","NODE7","TUBS",1000,0,0);
  node8  = new TNode("NODE8","NODE8","CONE",-300,-300,0);
  node9  = new TNode("NODE9","NODE9","CONS",300,300,0);
  node10 = new TNode("NODE10","NODE10","PCON",0,-1000,-1000);
  node11 = new TNode("NODE11","NODE11","PGON",0,1000,1000);
  node12 = new TNode("NODE12","NODE12","GTRA",0,-400,700);
```

// Draw this geometry in the current canvas

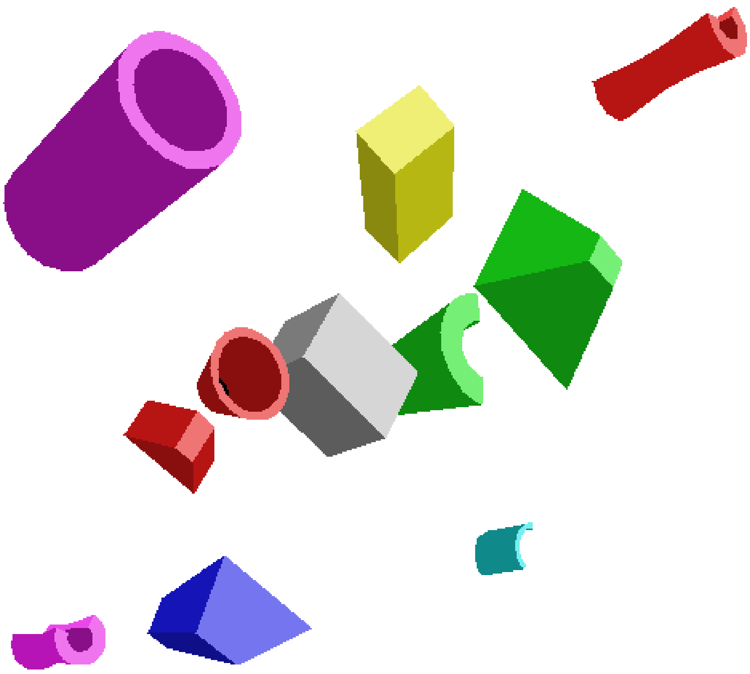
```
  node1->cd();
  node1->Draw();
  c1->Update();
```

// Draw the geometry using the x3d viewver.
// Note that this viewver may also be invoked from the "View" menu in
// the canvas tool bar

```
  c1->x3d();
```

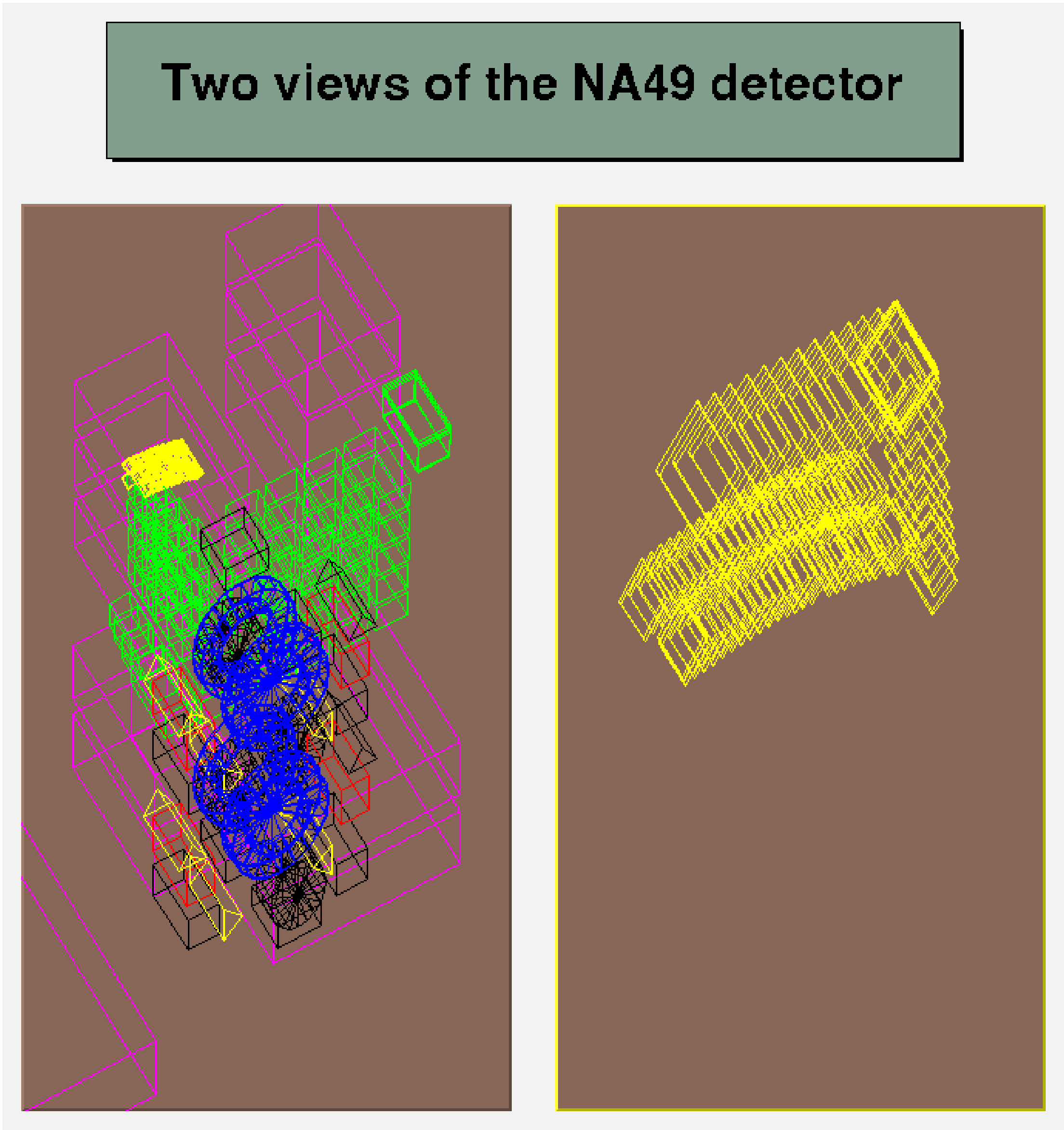
// once in x3d viewer, type m to see the menu.
// For example typing r will show a solid model of this geometry.

```
}
```



Creating and Viewing Geometries

```
{

// This macro generates
// a Canvas
// with 2 views of the NA49 detector.

   gROOT->Reset();
   c1 = new TCanvas("c1","The NA49 canvas",200,10,700,780);
   gBenchmark->Start("na49view");
   all = new TPad("all","A Global view of NA49",0.02,0.02,0.48,0.82,28);
   tof = new TPad("tof","One Time Of Flight element",0.52,0.02,0.98,0.82,28);
   all->Draw();
   tof->Draw();
   na49title = new TPaveLabel(0.1,0.86,0.9,0.98,"Two views of the NA49 detector");
   na49title->SetFillColor(32);
   na49title->Draw();

   nageom = new TFile("na49.root");
   TGeometry *n49 =(TGeometry*)gROOT->FindObject("na49");
   n49->SetBomb(1.2);
   n49->cd();     //Set current geometry
   all->cd();     //Set current pad
   n49->Draw();
   c1->Update();
   tof->cd();
   TOFR1->Draw();
   c1->Update();
   gBenchmark->Show("na49view");

// To have a better and dynamic view of any of these pads,
// you can click with the middle button of your mouse to select it.
// Then select "View with x3d" in the VIEW menu of the Canvas.
// Once in x3d, you are in wireframe mode by default.
// You can switch to:
// - Hidden Line mode by typing E
// - Solid mode by typing R
// - Wireframe mode by typing W
// - Stereo mode by clicking S (and you need special glasses)
// - To leave x3d type Q

}
```


Two views of the NA49 detector

Ntuples and Selections

```
{

  gROOT->Reset();
  c1 = new TCanvas("c1","The Ntuple canvas",200,10,700,780);
  gBenchmark->Start("ntuple1");
```

// Connect ROOT histogram/ntuple demonstration file
// generated by example hsimple.C.

```
  f1 = new TFile("hsimple.root");
```

// Inside this canvas, we create 4 pads

```
  pad1 = new TPad("pad1","This is pad1",0.02,0.52,0.48,0.98,21);
  pad2 = new TPad("pad2","This is pad2",0.52,0.52,0.98,0.98,21);
  pad3 = new TPad("pad3","This is pad3",0.02,0.02,0.48,0.48,21);
  pad4 = new TPad("pad4","This is pad4",0.52,0.02,0.98,0.48,1);
  pad1->Draw();
  pad2->Draw();
  pad3->Draw();
  pad4->Draw();
```

// Change default style for the statistics box

```
  gStyle->SetStatW(0.30);
  gStyle->SetStatH(0.20);
  gStyle->SetStatColor(42);
```

// Display a function of one ntuple column imposing a condition
// on another column.

```
  pad1->cd();
  pad1->SetGrid();
  pad1->SetLogy();
  pad1->GetFrame()->SetFillColor(15);
  ntuple->SetLineColor(1);
  ntuple->SetFillStyle(1001);
  ntuple->SetFillColor(45);
  ntuple->Draw("3*px+2","px**2+py**2>1");
  ntuple->SetFillColor(38);
  ntuple->Draw("2*px+2","pz>2","same");
  ntuple->SetFillColor(5);
  ntuple->Draw("1.3*px+2","(px^2+py^2>4) && py>0","same");
  c1->Update();
```

// Display the profile of two columns
// The profile histogram produced is saved in the current directory with
// the name hprofs

```
  pad2->cd();
  pad2->SetGrid();
  pad2->GetFrame()->SetFillColor(32);
  ntuple->Draw("pz:px>>hprofs","","profs");
  hprofs->SetMarkerColor(5);
  hprofs->SetMarkerSize(0.7);
  hprofs->SetMarkerStyle(21);
  hprofs->Fit("pol2","","same");
```

// Get pointer to fitted function and modify its attributes

```
  TF1 *fpol2 = hprofs->GetFunction("pol2");
  fpol2->SetLineWidth(4);
  fpol2->SetLineColor(2);
  c1->Update();
```

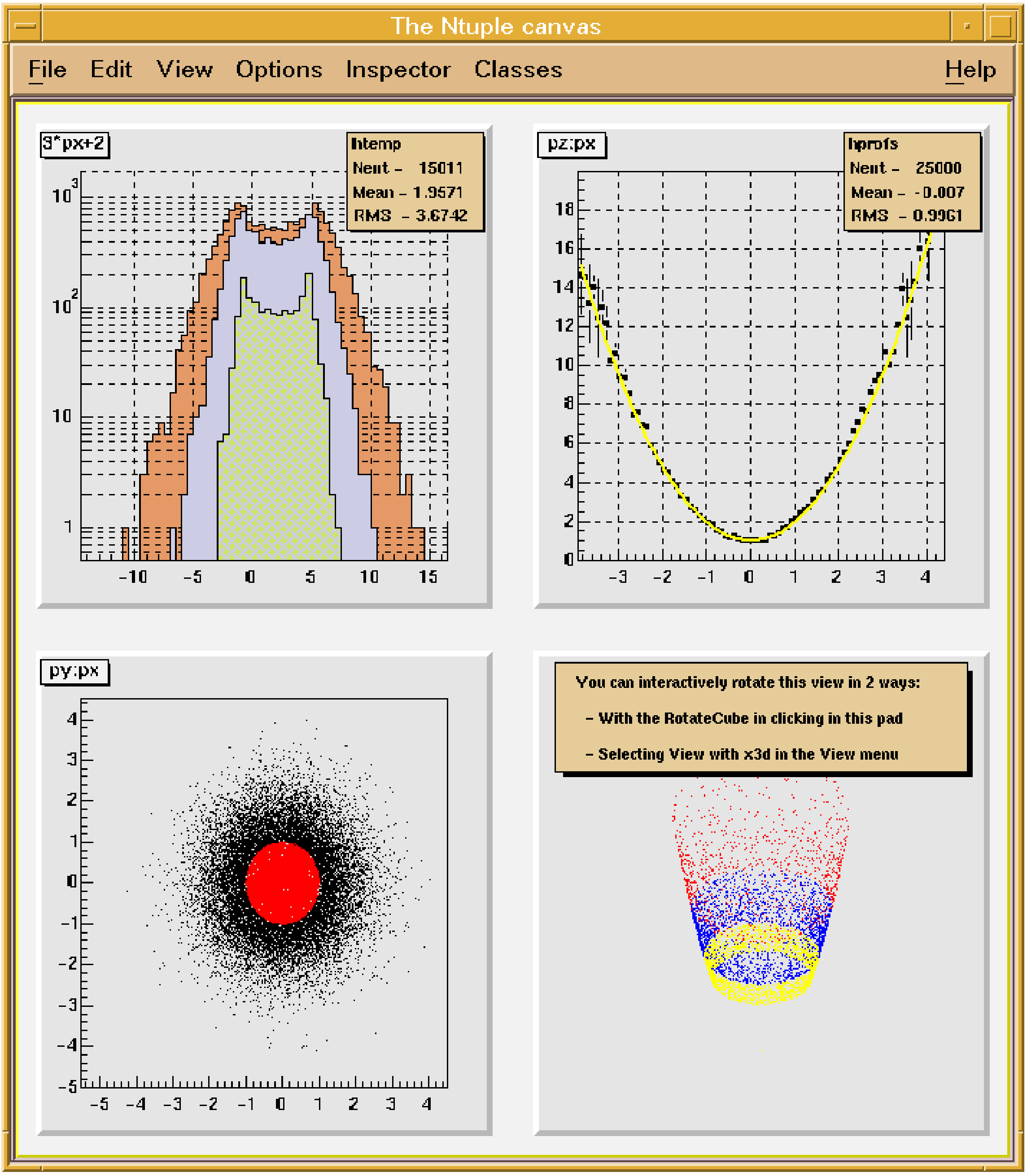
// Display a scatter plot of two columns with a selection.
// Superimpose the result of another cut with a different marker color

```
  pad3->cd();
  pad3->GetFrame()->SetFillColor(38);
  pad3->GetFrame()->SetBorderSize(8);
  ntuple->SetMarkerColor(1);
  ntuple->Draw("py:px","pz>1");
  ntuple->SetMarkerColor(2);
  ntuple->Draw("py:px","pz<1","same");
```

```
c1->Update();
```

// Display a 3-D scatter plot of 3 columns. Superimpose a different selection.

```
pad4->cd();
ntuple->Draw("pz:py:px","(pz<10 && pz>6)+(pz<4 && pz>3)");
ntuple->SetMarkerColor(4);
ntuple->Draw("pz:py:px","pz<6 && pz>4","same");
ntuple->SetMarkerColor(5);
ntuple->Draw("pz:py:px","pz<4 && pz>3","same");
l4 = new TPaveText(-0.9,0.5,0.9,0.95);
l4->SetFillColor(42);
l4->SetTextAlign(12);
l4->AddText("You can interactively rotate this view in 2 ways:");
l4->AddText("  - With the RotateCube in clicking in this pad");
l4->AddText("  - Selecting View with x3d in the View menu");
l4->Draw();

c1->cd();
c1->Update();
gStyle->SetStatColor(19);
gBenchmark->Show("ntuple1");
}
```



Creation of a ROOT Tree

```
//*CMZ : 1.00/07 04/04/97 10.13.50 by Ren\x8e Brun
//*-- Author : Valery Fine(fine@vxcern.cern.ch) 19/01/97

//______________________________________________________________________________
// A simple example with a ROOT tree
// =================================

// This program creates :
// - a ROOT file
// - a tree
// Additional arguments can be passed to the program to control the flow
// of execution. (see comments describing the arguments in the code).
// Event nevent comp split fill
// All arguments are optional: Default is
// Event 400 1 1 1

// In this example, the tree consists of one single "super branch"
// The statement ***tree->Branch("event", event, 64000,split);*** below
// will parse the structure described in Event.h and will make
// a new branch for each data member of the class if split is set to 1.
// - 5 branches corresponding to the basic types fNtrack,fNseg,fNvertex
// ,fFlag and fTemperature.
// - 3 branches corresponding to the members of the subobject EventHeader.
// - one branch for each data member of the class Track of TClonesArray.
// - one branch for the object fH (histogram of class TH1F).

// if split = 0 only one single branch is created and the complete event
// is serialized in one single buffer.
// if comp = 0 no compression at all.
// if comp = 1 event is compressed.
// if comp = 2 same as 1. In addition branches with floats in the TClonesArray
// are also compressed.
// The 4th argument fill can be set to 0 if one wants to time
// the percentage of time spent in creating the event structure and
// not write the event in the file.
// In this example, one loops over nevent events.
// The branch "event" is created at the first event.
// The branch address is set for all other events.
// For each event, the event header is filled and ntrack tracks
// are generated and added to the TClonesArray list.
// For each event the event histogram is saved as well as the list
// of all tracks.

// The number of events can be given as the first argument to the program.
// By default 400 events are generated.
// The compression option can be activated/deactivated via the second argument.

// ---Running/Linking instructions----
// This program consists of the following files and procedures.
// - Event.h event class description
// - Event.C event class implementation
// - MainEvent.C the main program to demo this class might be used (this file)
// - EventCint.C the CINT dictionary for the event and Track classes
// this file is automatically generated by rootcint (see Makefile),
// when the class definition in Event.h is modified.

// ---Analyzing the Event.root file with the interactive root
// example of a simple session
```

```
// Root > TFile f("Event.root")
// Root > T.Draw("fNtrack") //histogram the number of tracks per event
// Root > T.Draw("fPx") //histogram fPx for all tracks in all events
// Root > T.Draw("fXfirst:fYfirst","fNtrack>600")
// //scatter-plot for x versus y of first point of each track
// Root > T.Draw("fH.GetRMS()") //histogram of the RMS of the event histogram

// Look also in the same directory at the following macros:
// - eventa.C an example how to read the tree
// - eventb.C how to read events conditionally

//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

#include <stdlib.h>
#include "TROOT.h"
#include "TFile.h"
#include "TRandom.h"
#include "TTree.h"
#include "TBranch.h"
#include "TClonesArray.h"
#include "TStopwatch.h"
#include "Event.h"

//______________________________________________________________________________

main(int argc, char **argv)
{
  TROOT simple("simple","Example of creation of a tree");
  Int_t nevent = 400;     // by default create 400 events
  Int_t comp   = 1;       // by default file is compressed
  Int_t split  = 1;       // by default, split Event in sub branches
  Int_t write  = 1;       // by default the tree is filled
  Int_t hfill  = 0;       // by default histograms are not filled
  Int_t read   = 0;
  Int_t arg4   = 1;
  Int_t arg5   = 600;     //default number of tracks per event
  if (argc > 1)  nevent = atoi(argv[1]);
  if (argc > 2)  comp   = atoi(argv[2]);
  if (argc > 3)  split  = atoi(argv[3]);
  if (argc > 4)  arg4   = atoi(argv[4]);
  if (argc > 5)  arg5   = atoi(argv[5]);
  if (arg4 ==  0) { write = 0; hfill = 0;}
  if (arg4 ==  1) { write = 1; hfill = 0;}
  if (arg4 == 10) { write = 0; hfill = 1;}
  if (arg4 == 11) { write = 1; hfill = 1;}
  if (arg4 == 20) { write = 0; read  = 1;}  //read sequential
  if (arg4 == 25) { write = 0; read  = 2;}  //read random
  TFile *hfile;
  Event *event;
  TTree *tree;
  TBranch *b = 0;

// Fill event, header and tracks with some random numbers
// Create a timer object to benchmark this loop

  TStopwatch timer;
  timer.Start();
  Int_t nb = 0;
  Int_t ev;
  Int_t bufsize;
  Double_t told = 0;
  Double_t tnew = 0;
  Int_t printev = 100;
  if (arg5 < 100) printev = 1000;
  if (arg5 < 10)  printev = 10000;

// Read case

  if (read) {
     hfile = new TFile("Event.root");
     tree = (TTree*)hfile->Get("T");
     TBranch *branch = tree->GetBranch("event");
     event = new Event();
     branch->SetAddress(&event);
     Int_t nentries = (Int_t)tree->GetEntries();
     nevent = TMath::Max(nevent,nentries);
     if (read == 1) {  //read sequential
        for (ev = 0; ev < nevent; ev++) {
           if (ev%printev == 0) cout<<"event="<<ev<<endl;
           nb += tree->GetEvent(ev);        //read complete event in memory
```

```
         }
      } else {    //read random
         Int_t evrandom;
         for (ev = 0; ev < nevent; ev++) {
            if (ev%printev == 0) cout<<"event="<<ev<<endl;
            evrandom = nevent*gRandom->Rndm(1);
            nb += tree->GetEvent(evrandom);  //read complete event in memory
         }
      }
   } else {
```

// Write case
// Create a new ROOT binary machine independent file.
// Note that this file may contain any kind of ROOT objects, histograms,
// pictures, graphics objects, detector geometries, tracks, events, etc..
// This file is now becoming the current directory.

```
      hfile = new TFile("Event.root","RECREATE","TTree benchmark ROOT file");
      hfile->SetCompressionLevel(comp);
```

// Create histogram to show write_time in function of time

```
     Float_t curtime = -0.5;
     Int_t ntime = nevent/printev;
     TH1F *htime = new TH1F("htime","Real-Time to write versus time",ntime,0,ntime);
```

// Create one event

```
      event = new Event();
      if (hfill) event->Hcreate();
```

// Create a ROOT Tree and one superbranch

```
      TTree *tree = new TTree("T","An example of a ROOT tree");
      tree->SetAutoSave(1000000000);  // autosave when 1 Gbyte written
      bufsize = 256000;
      if (split)  bufsize /= 4;
      TBranch *b = tree->Branch("event", "Event", &event, bufsize,split);
      for (ev = 0; ev < nevent; ev++) {
         if (ev%printev == 0) {
            tnew = timer.RealTime();
            printf("event:%d, rtime=%f s
",ev,tnew-told);
            htime->Fill(curtime,tnew-told);
            curtime += 1;
            told=tnew;
            timer.Continue();
         }
         Float_t sigmat, sigmas;
         gRandom->Rannor(sigmat,sigmas);
         Int_t ntrack   = Int_t(arg5 +arg5*sigmat/120.);
         Float_t random = gRandom->Rndm(1);
         event->SetHeader(ev, 200, 960312, random);
         event->SetNseg(Int_t(10*ntrack+20*sigmas));
         event->SetNvertex(1);
         event->SetFlag(UInt_t(random+0.5));
         event->SetTemperature(random+20.);
```

// Create and Fill the Track objects

```
         for (Int_t t = 0; t < ntrack; t++) event->AddTrack(random);
         if (write) nb += tree->Fill();  //fill the tree
         if (hfill) event->Hfill();      //fill histograms
         event->GetTracks()->Clear();
      }
      if (write) { hfile->Write();   tree->Print();}
   }
```

// Stop timer and print results

```
   timer.Stop();
   Float_t mbytes = 0.000001*nb;
   Double_t rtime = timer.RealTime();
   Double_t ctime = timer.CpuTime();
   printf("
%d events and %d bytes processed.
",nevent,nb);
   printf("RealTime=%f seconds, CpuTime=%f seconds
",rtime,ctime);
   if (read) {
      printf("You read %f Mbytes/Realtime seconds
```

```
",mbytes/rtime);
     printf("You read %f Mbytes/Cputime seconds
",mbytes/ctime);
  } else {
     printf("compression level=%d, split=%d, arg4=%d
",comp,split,arg4);
     printf("You write %f Mbytes/Realtime seconds
",mbytes/rtime);
     printf("You write %f Mbytes/Cputime seconds
",mbytes/ctime);
```

***//printf("file compression factor = %f
",hfile.GetCompressionFactor());***

```
  }
  hfile->Close();
  return 0;
}
```

//*CMZ : 1.00/00 14/02/97 11.04.07 by Ren\x8e Brun
//*-- Author : René Brun 19/08/96
//____
// Event and Track classes
// =======================

// The Event class is a naive/simple example of an event structure.
// public:
// Int_t fNtrack;
// Int_t fNseg;
// Int_t fNvertex;
// UInt_t fFlag;
// Float_t fTemperature;
// EventHeader fEvtHdr;
// TClonesArray *fTracks;
// TH1F *fH;

// The EventHeader class has 3 data members (integers):
// public:
// Int_t fEvtNum;
// Int_t fRun;
// Int_t fDate;

// The Event data member fTracks is a pointer to a TClonesArray.
// It is an array of a variable number of tracks per event.
// Each element of the array is an object of class Track with the members:
// private:
// Float_t fPx; //X component of the momentum
// Float_t fPy; //Y component of the momentum
// Float_t fPz; //Z component of the momentum
// Float_t fRandom; //A random track quantity
// Float_t fMass2; //The mass square of this particle
// Float_t fBx; //X intercept at the vertex
// Float_t fBy; //Y intercept at the vertex
// Float_t fMeanCharge; //Mean charge deposition of all hits of this track
// Float_t fXfirst; //X coordinate of the first point
// Float_t fXlast; //X coordinate of the last point
// Float_t fYfirst; //Y coordinate of the first point
// Float_t fYlast; //Y coordinate of the last point
// Float_t fZfirst; //Z coordinate of the first point
// Float_t fZlast; //Z coordinate of the last point
// Float_t fCharge; //Charge of this track
// Int_t fNpoint; //Number of points for this track
// Short_t fValid; //Validity criterion

// An example of a batch program to use the Event/Track classes is given
// in this directory: MainEvent.
// Look also in the same directory at the following macros:
// - eventa.C an example how to read the tree

// - eventb.C how to read events conditionally

//*-*

```
#include "TRandom.h"
#include "Event.h"
ClassImp(EventHeader)
ClassImp(Event)
ClassImp(Track)
TClonesArray *gTracks;
TH1F *gHist;
TH1F *hNtrack,*hNseg, *hTemperature;
TH1F *hPx, *hPy, *hPz, *hRandom, *hMass2, *hBx, *hBy;
TH1F *hMeanCharge,*hXfirst,*hXlast,*hYfirst,*hYlast;
TH1F *hZfirst,*hZlast,*hCharge,*hNpoint,*hValid;
```

//____

```
Event::Event()
{
```

// Create one Event object
// When the constructor is invoked for the first time, the global
// variable gTracks is NULL. The TClonesArray gTracks is created.
// The histogram fH is created.

```
  fNtrack = 0;
  if (!gTracks) gTracks = new TClonesArray("Track", 1000);
  fTracks = gTracks;
  fH = 0;
  }
```

//____

```
Event::~Event()
{
  Clear();
}
```

//____

```
void Event::AddTrack(Float_t random)
{
```

// Add a new track to the list of tracks for this event.
// To avoid calling the very time consuming operator new for each track,
// the standard but not well know C++ operator "new with placement" is called.
// if tracks[i] is NULL, a new Track object will be created
// otherwise the previous Track[i] will be overwritten.

```
  TClonesArray &tracks = *fTracks;
  new(tracks[fNtrack++]) Track(random);
}
```

//____

```
void Event::Clear()
{
  fTracks->Clear();
  delete fH;
  fH = 0;
}
```

//____

```
void Event::Hcreate()
{
```

// Create histograms

```
  hNtrack      = new TH1F("hNtrack",     "Ntrack",100,575,625);
  hNseg        = new TH1F("hNseg",       "Nseg",100,5800,6200);
  hTemperature = new TH1F("hTemperature","Temperature",100,19.5,20.5);
  hPx          = new TH1F("hPx",         "Px",100,-4,4);
  hPy          = new TH1F("hPy",         "Py",100,-4,4);
  hPz          = new TH1F("hPz",         "Pz",100,0,5);
  hRandom      = new TH1F("hRandom",     "Random",100,0,1000);
```

```
  hMass2       = new TH1F("hMass2",     "Mass2",100,0,12);
  hBx          = new TH1F("hBx",        "Bx",100,-0.5,0.5);
  hBy          = new TH1F("hBy",        "By",100,-0.5,0.5);
  hMeanCharge  = new TH1F("hMeanCharge","MeanCharge",100,0,0.01);
  hXfirst      = new TH1F("hXfirst",    "Xfirst",100,-40,40);
  hXlast       = new TH1F("hXlast",     "Xlast",100,-40,40);
  hYfirst      = new TH1F("hYfirst",    "Yfirst",100,-40,40);
  hYlast       = new TH1F("hYlast",     "Ylast",100,-40,40);
  hZfirst      = new TH1F("hZfirst",    "Zfirst",100,0,80);
  hZlast       = new TH1F("hZlast",     "Zlast",100,0,250);
  hCharge      = new TH1F("hCharge",    "Charge",100,-1.5,1.5);
  hNpoint      = new TH1F("hNpoint",    "Npoint",100,50,80);
  hValid       = new TH1F("hValid",     "Valid",100,0,1.2);
}
```

//____

```
void Event::Hfill()
{
```

// Fill histograms

```
  hNtrack->Fill(fNtrack);
  hNseg->Fill(fNseg);
  hTemperature->Fill(fTemperature);
  for (Int_t itrack=0;itrack<fNtrack;itrack++) {
     Track *track = (Track*)fTracks->UncheckedAt(itrack);
     hPx->Fill(track->GetPx());
     hPy->Fill(track->GetPy());
     hPz->Fill(track->GetPz());
     hRandom->Fill(track->GetRandom());
     hMass2->Fill(track->GetMass2());
     hBx->Fill(track->GetBx());
     hBy->Fill(track->GetBy());
     hMeanCharge->Fill(track->GetMeanCharge());
     hXfirst->Fill(track->GetXfirst());
     hXlast->Fill(track->GetXlast());
     hYfirst->Fill(track->GetYfirst());
     hYlast->Fill(track->GetYlast());
     hZfirst->Fill(track->GetZfirst());
     hZlast->Fill(track->GetZlast());
     hCharge->Fill(track->GetCharge());
     hNpoint->Fill(track->GetNpoint());
     hValid->Fill(track->GetValid());
  }
}
```

//____

```
void Event::SetHeader(Int_t i, Int_t run, Int_t date, Float_t random)
{
  fNtrack = 0;
  fEvtHdr.Set(i, run, date);
  if (!gHist)   gHist   = new TH1F("hstat","Event Histogram",100,0,1);
  fH = gHist;
  fH->Fill(random);
}
```

//____

```
Track::Track(Float_t random) :TObject()
{
```

//*-*-*-*-*-*-*-*-*-*-*Track normal constructor*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
//*-* ======================
// Note that in this example, data members do not have any physical meaning

```
 Float_t a,b,sigmat,sigmas,px,py,pz;
 gRandom->Rannor(px,py);
 fPx = px;
 fPy = py;
 fPz = TMath::Sqrt(px*px+py*py);
 fRandom = 1000*random;
 if (fRandom < 10) fMass2 = 0.08;
 else if (fRandom < 100) fMass2 = 0.8;
 else if (fRandom < 500) fMass2 = 4.5;
 else if (fRandom < 900) fMass2 = 8.9;
 else  fMass2 = 9.8;
 gRandom->Rannor(a,b);
 fBx = 0.1*a;
 fBy = 0.1*b;
 fMeanCharge = 0.01*gRandom->Rndm(1);
 gRandom->Rannor(a,b);
```

```
 fXfirst = a*10;
 fXlast  = b*10;
 gRandom->Rannor(a,b);
 fYfirst = a*12;
 fYlast  = b*16;
 gRandom->Rannor(a,b);
 fZfirst = 50 + 5*a;
 fZlast  = 200 + 10*b;
 fCharge = Float_t(Int_t(3*gRandom->Rndm(1)) - 1);
 fNpoint = Int_t(60+10*gRandom->Rndm(1));
 fValid  = Int_t(0.6+gRandom->Rndm(1));
}
```

Reading all events of a ROOT Tree

//*CMZ : 1.00/05 20/03/97 09.05.34 by Ren\x8e Brun
//*-- Author : René Brun 10/01/97

```
{
```

// This macro read all events generated by the test program Event
// provided in $ROOTSYS/test.

// NOTE: Before executing this macro, you must have executed the macro eventload.

// This small program simply counts the number of bytes read and dump
// the first 3 events.

```
  gROOT->Reset();
```

// Connect file generated in $ROOTSYS/test

```
  TFile f("Event.root");
```

// Read Tree named "T" in memory. Tree pointer is assigned the same name

```
  TTree *T = (TTree*)f.Get("T");
```

// Create a timer object to benchmark this loop

```
  TStopwatch timer;
  timer.Start();
```

// Start main loop on all events

```
  Event *event = new Event();
  TClonesArray *tracks = event->GetTracks();
  TBranch *branch = T->GetBranch("event");
  branch->SetAddress(&event);
  Int_t nevent = T->GetEntries();
  Int_t nb = 0;
  for (Int_t i=0;i<nevent;i++) {
     if(i%50 == 0) printf("Event:%d
",i);
     nb += T->GetEvent(i);                 //read complete event in memory
     if (i < 3) event->Dump();             //dump the first 3 events
     event->Clear();                       //clear tracks array
  }
```

// Stop timer and print results

```
  timer.Stop();
  Float_t mbytes = 0.000001*nb;
  Double_t rtime = timer.RealTime();
  Double_t ctime = timer.CpuTime();
  printf("RealTime=%f seconds, CpuTime=%f seconds
",rtime,ctime);
  printf("You read %f Mbytes/Realtime seconds
",mbytes/rtime);
  printf("You read %f Mbytes/Cputime seconds
",mbytes/ctime);
  printf("%d events and %d bytes read.
",nevent,nb);
  f.Close();
}
```

Reading selected events of a ROOT Tree

//*CMZ : 1.00/05 20/03/97 09.10.53 by Ren\x8e Brun
//*-- Author : René Brun 10/01/97

```
{
```

// This macro is a variant of the macro eventa.

// NOTE: Before executing this macro, you must have executed the macro eventload.

// This small program loop on all events:
// - It reads the small branch containing the number of tracks per event
// - It reads the full event only for events having less than 587 tracks
// - It dumps the selected events.

```
  gROOT->Reset();
```

// Connect file generated in $ROOTSYS/test

```
  TFile f("Event.root");
```

// Read Tree named "T" in memory. Tree pointer is assigned the same name

```
  TTree *T = (TTree*)f.Get("T");
```

// Create a timer object to benchmark this loop

```
  TStopwatch timer;
  timer.Start();
```

// Start main loop on all events

```
  Event *event = new Event();   //we create the event object once outside the loop
  TClonesArray *tracks = event->GetTracks();
  TBranch *bntrack = T->GetBranch("fNtrack");
  TBranch *branch  = T->GetBranch("event");
  branch->SetAddress(&event);
  Int_t nevent = T->GetEntries();
  Int_t nselected = 0;
  Int_t nb = 0;
  for (Int_t i=0;i<nevent;i++) {
     if(i%50 == 0) printf("Event:%d
",i);
     bntrack->GetEvent(i);                   //read branch "fNtrack" only
     if (event->GetNtrack() > 587)continue; //reject events with more than 587 tracks
     nb += T->GetEvent(i);                   //read complete accepted event in memory
     nselected++;
     if (nselected == 1) event->Dump();     //dump the first accepted event
     event->Clear();                         //clear tracks array
  }
```

// Stop timer and print results

```
  timer.Stop();
  Float_t mbytes = T->GetTotBytes()*1.e-6;
  Double_t rtime = timer.RealTime();
  Double_t ctime = timer.CpuTime();
  printf("You have selected %d events out of %d
",nselected,nevent);
  printf("RealTime=%f seconds, CpuTime=%f seconds
",rtime,ctime);
  printf("You have scanned %f Mbytes/Realtime seconds
",mbytes/rtime);
  printf("You have scanned %f Mbytes/Cputime seconds
",mbytes/ctime);
  f.Close();
}
```

Example of analysis code with Chains

```
{
```

//
// This macro illustrates how to loop on events in a chain of files
// containing a TTree object
// The skeleton of this file has been automatically generated
// (Wed Jul 23 14:09:39 1997 by ROOT version 1.01/07)
// from TTree h3333/ATLFAST
// found on file: atlfast_whbb_lowlum_r04.root

// The analysis module and the data sets have been kindly provided
// by the ATLAS Atlfast team.

// The complete source of this macro, associated macros and files
// can be found at AtlFast massbb
//

//Reset ROOT

```
  gROOT->Reset();
```

//We intend to use this macro for benchmarking ROOT

```
  gBenchmark->Start("massbb");
```

//Load some utility functions and the event analysis code

```
  gROOT->LoadMacro("ElaUtil.C");
  gROOT->LoadMacro("ElaAnal.C");
```

// histos initialization. Create file and result histograms

```
  TFile histofile("atlf_massbb.root","RECREATE","WH-->bb mass distribution");
  TH1F Mbb_cru("Mbb_cru","m_bb_cru (GeV)",50,0,150);
  TH1F Ptj_cru("Ptj_cru","p_T^jet_cru (GeV)",50,0,100);
  TH1F Mbb("Mbb","m_bb (GeV)",50,0,150);
  TH1F Ptj("Ptj","p_T^jet (GeV)",50,0,100);
```

// Connecting calibration file

```
  TFile calibfile("atlf_calibration.root");
```

// Get histogram Kfac from file. We directly create a pointer
// because we intend to use this histogram when the current directory
// is not calibfile itself

```
  TH1F *Kfac = (TH1F*)calibfile.Get("Kfac");
```

// Create a chain of files. The parameter to the TChain constructor
// is the name of the Tree to be processed in each file of the chain.

```
  TChain chain("h3333");
  chain.Add("atlfast_whbb_lowlum_r04.root");
  chain.Add("atlfast_whbb_lowlum_r04.root");  // <== add a new file
```

//Declaration of leaves types

```
  UInt_t          Jrun;
  UInt_t          Jevnt;
  UInt_t          Jflag;
  UInt_t          Isub;
  UInt_t          Nel;
  UInt_t          Nmu;
  UInt_t          Nmux;
  UInt_t          Nph;
  UInt_t          Jetb;
  UInt_t          Jetc;
  UInt_t          Jetl;
  Float_t         Pmiss[2];
```

```
UInt_t          Nlep;
UInt_t          Kflep[12];
Float_t         Pxlep[12];
Float_t         Pylep[12];
Float_t         Pzlep[12];
Float_t         Eelep[12];
UInt_t          Npho;
UInt_t          Kfpho[12];
Float_t         Pxpho[12];
Float_t         Pypho[12];
Float_t         Pzpho[12];
Float_t         Eepho[12];
UInt_t          Njeta;
UInt_t          Kfjet[20];
Float_t         Pxjet[20];
Float_t         Pyjet[20];
Float_t         Pzjet[20];
Float_t         Eejet[20];
UInt_t          Nonmux;
UInt_t          Kfmux[12];
Float_t         Pxmux[12];
Float_t         Pymux[12];
Float_t         Pzmux[12];
Float_t         Eemux[12];
UInt_t          Npart;
UInt_t          Kfpar[40];
Float_t         Pxpar[40];
Float_t         Pypar[40];
Float_t         Pzpar[40];
Float_t         Eepar[40];
Float_t         Tgall;
Float_t         Tgem1;
Float_t         Tgph1;
Float_t         Tgem2;
Float_t         Tgmu1;
Float_t         Tgmu2;
Float_t         Tgemu;
Float_t         Tgjt1;
Float_t         Tgjt3;
Float_t         Tgjt4;
UInt_t          Ntra;
UInt_t          Kftra[100];
Float_t         Pxtra[100];
Float_t         Pytra[100];
Float_t         Pztra[100];
Float_t         Eetra[100];
```

//Set branch addresses

```
chain.SetBranchAddress("Jrun",&Jrun);
chain.SetBranchAddress("Jevnt",&Jevnt);
chain.SetBranchAddress("Jflag",&Jflag);
chain.SetBranchAddress("Isub",&Isub);
chain.SetBranchAddress("Nel",&Nel);
chain.SetBranchAddress("Nmu",&Nmu);
chain.SetBranchAddress("Nmux",&Nmux);
chain.SetBranchAddress("Nph",&Nph);
chain.SetBranchAddress("Jetb",&Jetb);
chain.SetBranchAddress("Jetc",&Jetc);
chain.SetBranchAddress("Jetl",&Jetl);
chain.SetBranchAddress("Pmiss",Pmiss);
chain.SetBranchAddress("Nlep",&Nlep);
chain.SetBranchAddress("Kflep",Kflep);
chain.SetBranchAddress("Pxlep",Pxlep);
chain.SetBranchAddress("Pylep",Pylep);
chain.SetBranchAddress("Pzlep",Pzlep);
chain.SetBranchAddress("Eelep",Eelep);
chain.SetBranchAddress("Npho",&Npho);
chain.SetBranchAddress("Kfpho",Kfpho);
chain.SetBranchAddress("Pxpho",Pxpho);
chain.SetBranchAddress("Pypho",Pypho);
chain.SetBranchAddress("Pzpho",Pzpho);
chain.SetBranchAddress("Eepho",Eepho);
chain.SetBranchAddress("Njeta",&Njeta);
chain.SetBranchAddress("Kfjet",Kfjet);
chain.SetBranchAddress("Pxjet",Pxjet);
chain.SetBranchAddress("Pyjet",Pyjet);
chain.SetBranchAddress("Pzjet",Pzjet);
chain.SetBranchAddress("Eejet",Eejet);
chain.SetBranchAddress("Nonmux",&Nonmux);
chain.SetBranchAddress("Kfmux",Kfmux);
chain.SetBranchAddress("Pxmux",Pxmux);
chain.SetBranchAddress("Pymux",Pymux);
chain.SetBranchAddress("Pzmux",Pzmux);
chain.SetBranchAddress("Eemux",Eemux);
chain.SetBranchAddress("Npart",&Npart);
```

```
  chain.SetBranchAddress("Kfpar",Kfpar);
  chain.SetBranchAddress("Pxpar",Pxpar);
  chain.SetBranchAddress("Pypar",Pypar);
  chain.SetBranchAddress("Pzpar",Pzpar);
  chain.SetBranchAddress("Eepar",Eepar);
  chain.SetBranchAddress("Tgall",&Tgall);
  chain.SetBranchAddress("Tgem1",&Tgem1);
  chain.SetBranchAddress("Tgph1",&Tgph1);
  chain.SetBranchAddress("Tgem2",&Tgem2);
  chain.SetBranchAddress("Tgmu1",&Tgmu1);
  chain.SetBranchAddress("Tgmu2",&Tgmu2);
  chain.SetBranchAddress("Tgemu",&Tgemu);
  chain.SetBranchAddress("Tgjt1",&Tgjt1);
  chain.SetBranchAddress("Tgjt3",&Tgjt3);
  chain.SetBranchAddress("Tgjt4",&Tgjt4);
  chain.SetBranchAddress("Ntra",&Ntra);
  chain.SetBranchAddress("Kftra",Kftra);
  chain.SetBranchAddress("Pxtra",Pxtra);
  chain.SetBranchAddress("Pytra",Pytra);
  chain.SetBranchAddress("Pztra",Pztra);
  chain.SetBranchAddress("Eetra",Eetra);
```

// This is the loop skeleton
// To read only selected branches, Insert statements like:
// chain.SetBranchStatus("*",0); // disable all branches
// chain.SetBranchStatus("branchname",1); // activate branchname

// Invoke analysis code

```
  ElaAnal();
```

//..... save histograms

```
  histofile.Write();
  histofile.Close();
  gBenchmark->Show("massbb");
  Float_t hp735     = 27;  //this macro gives 27 RootMarks on an HP735
  Float_t massbb_rt = gBenchmark->GetRealTime("massbb");
  Float_t massbb_ct = gBenchmark->GetCpuTime("massbb");
  Float_t rtmark    = hp735*(439/massbb_rt);
  Float_t cpmark    = hp735*(434/massbb_ct);
  printf("massbb    = %7.2f RealMARKS,  = %7.2f CpuMARKS
",rtmark,cpmark);
  Float_t rtmarks   = 0.5*(rtmark+cpmark); // take average of realtime and cputime
  printf("
");
  printf("****************************************************
");
  printf("* Your machine is estimated at %7.2f ROOTMARKS   *
",rtmarks);
  printf("****************************************************
");
}
```

Example of a Minimization program

```
//*CMZ : 1.00/06 23/03/97 11.54.04 by Ren\x8e Brun
//*-- Author : René Brun 22/08/95

//______________________________________________________________________________
//*-*-*-*-*-*-*-*-*-*-*-*The Minuit standard test program-*-*-*-*-*-*-*-*-*
//*-* ======================== *
//*-* *
//*-* This program is the translation to C++ of the minexam program *
//*-* distributed with the Minuit/Fortran source file. *
//*-* original author Fred James *
//*-* *
//*-* Fit randomly-generated leptonic K0 decays to the *
//*-* time distribution expected for interfering K1 and K2, *
//*-* with free parameters Re(X), Im(X), DeltaM, and GammaS. *
//*-* *
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

```
#include "TROOT.h"
#include "TMinuit.h"
#include "TMath.h"
#include "TStopwatch.h"
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
extern void fcnk0(Int_t &npar, Double_t *gin, Double_t &f, Double_t *x, Int_t iflag);
static ncount = 0;
```

```
//______________________________________________________________________________
```

```
main()
{
  TROOT minexam("TestMinuit","Test of Minuit");
  static Int_t nprm[5] = { 0   ,    1   ,     4    ,   9     ,  10};
  static Double_t vstrt[5] = {0.  ,    0.  ,    .535  ,   .892   ,  518.3};
  static Double_t stp[5] = {0.1 ,    0.1 ,     0.1  ,     0.   ,   0.};
  static Double_t p0=0;
  static Double_t p1=1;
  static Double_t p3=3;
  static Double_t p5=5;
  Int_t ierflg = 0;
  const Int_t FIVE=5;
  const Int_t THREE=3;
  TStopwatch timer;
  TMinuit *gMinuit = new TMinuit(5);  //initialize TMinuit with a maximum of 5 params
  cout<<"Starting timer"<<endl;
  timer.Start();
  gMinuit->mninit(5,6,7);
  gMinuit->SetFCN(fcnk0);
  gMinuit->mnparm(nprm[0], "Re(X)",    vstrt[0], stp[0], 0,0,ierflg);
  gMinuit->mnparm(nprm[1], "Im(X)",    vstrt[1], stp[1], 0,0,ierflg);
  gMinuit->mnparm(nprm[2], "Delta M",  vstrt[2], stp[2], 0,0,ierflg);
  gMinuit->mnparm(nprm[3], "T Kshort", vstrt[3], stp[3], 0,0,ierflg);
  gMinuit->mnparm(nprm[4], "T Klong",  vstrt[4], stp[4], 0,0,ierflg);
  if (ierflg) {
     Printf(" UNABLE TO DEFINE PARAMETER NO.");
     return ierflg;
  }
```

```
// gMinuit->mnseti("Time Distribution of Leptonic K0 Decays");
//*-*- Request FCN to read in (or generate random) data (IFLAG=1)
```

```
  gMinuit->mnexcm("CALL FCN", &p1 ,1,ierflg);
  gMinuit->mnexcm("FIX", &p5 ,1,ierflg);
  gMinuit->mnexcm("SET PRINT", &p0 ,1,ierflg);
  gMinuit->mnexcm("MIGRAD", &p0 ,0,ierflg);
  gMinuit->mnexcm("MINOS", &p0 ,0,ierflg);
  gMinuit->mnexcm("RELEASE", &p5 ,1,ierflg);
  gMinuit->mnexcm("MIGRAD", &p0 ,0,ierflg);
  gMinuit->mnexcm("MINOS",  &p0 ,0,ierflg);
  gMinuit->mnexcm("CALL FCN", &p3 , 1,ierflg);
  cout<<"Time at the end of job = "<<timer.CpuTime()<<" seconds"<<endl;
  return 0;
```

```
}
```

//____

```
void fcnk0(Int_t &npar, Double_t *gin, Double_t &f, Double_t *x, Int_t iflag)
{
  const Int_t MXBIN=50;
  static Double_t thplu[MXBIN],thmin[MXBIN],t[MXBIN];
  static Int_t nbins = 30;
  static Int_t nevtot = 250;
  static Double_t evtp[30] = {
             11.,  9., 13., 13., 17.,  9.,  1.,  7.,  8.,  9.,
              6.,  4.,  6.,  3.,  7.,  4.,  7.,  3.,  8.,  4.,
              6.,  5.,  7.,  2.,  7.,  1.,  4.,  1.,  4.,  5.};
  static Double_t evtm[30] = {
              0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,
              0.,  2.,  1.,  4.,  4.,  2.,  4.,  2.,  2.,  0.,
              2.,  3.,  7.,  2.,  3.,  6.,  2.,  4.,  1.,  5.};
  static Int_t i, nevplu, nevmin;
  static Double_t xre, xim, dm, gams, gaml,gamls,xr1,xr2,em,ep;
  static Double_t sthplu, sthmin, ehalf, th, sterm, sevtp, sevtm;
  static Double_t thplui, thmini, chisq, ti, thp, thm, evp, evm, chi1, chi2;
  xre   = x[0];
  xim   = x[1];
  dm    = x[4];
  gams  = 1/x[9];
  gaml  = 1/x[10];
  gamls = 0.5*(gaml+gams);
  if (iflag != 1)  goto L300;
```

// generate random data

```
  sthplu = sthmin = 0;
  for (i=1;i<=nbins; i++) {
     t[i-1] = 0.1*Double_t(i);
     ti = t[i-1];
     ehalf = TMath::Exp(-ti*gamls);
     xr1 = 1-xre;
     xr2 = 1+xre;
     th =      (xr1*xr1 + xim*xim) * TMath::Exp(-ti*gaml);
     th = th + (xr2*xr2 + xim*xim) * TMath::Exp(-ti*gams);
     th = th -               4*xim*TMath::Sin(dm*ti) * ehalf;
     sterm = 2*(1-xre*xre-xim*xim)*TMath::Cos(dm*ti) * ehalf;
     thplu[i-1] = th + sterm;
     thmin[i-1] = th - sterm;
     sthplu += thplu[i-1];
     sthmin += thmin[i-1];
  }
  nevplu = Int_t(Double_t(nevtot)*(sthplu/(sthplu+sthmin)));
  nevmin = Int_t(Double_t(nevtot)*(sthmin/(sthplu+sthmin)));
  cout<<"  LEPTONIC K ZERO DECAYS"<<endl;
  cout<<Form(" PLUS, MINUS, TOTAL=%5d %5d %5d",nevplu,nevmin,nevtot)<<endl;
  cout<<"0      TIME       THEOR+      EXPTL+      THEOR-      EXPTL-"<<endl;
  sevtp = sevtm = 0;
  for (i=1;i<=nbins; i++) {
     thplu[i-1] = thplu[i-1]*Double_t(nevplu) / sthplu;
     thmin[i-1] = thmin[i-1]*Double_t(nevmin) / sthmin;
     thplui = thplu[i-1];
     sevtp += evtp[i-1];
     thmini = thmin[i-1];
     sevtm += evtm[i-1];
     if (iflag != 4) {
        cout<<Form("%12.4f%12.4f%12.4f%12.4f%12.4f",t[i-1],thplu[i-1],evtp[i-1],thmin[i-1],evtm[i-
1])<<endl;
     }
  }
  cout<<Form(" DATA EVTS PLUS, MINUS= %10.2f%10.2f", sevtp,sevtm)<<endl;
```

// calculate chisquared

```
L300:
  chisq = sthplu = sthmin = 0;
  for (i=1;i<=nbins; i++) {
     ti = t[i-1];
     ehalf = TMath::Exp(-ti*gamls);
     xr1 = 1-xre;
     xr2 = 1+xre;
     th =      (xr1*xr1 + xim*xim) * TMath::Exp(-ti*gaml);
     th = th + (xr2*xr2 + xim*xim) * TMath::Exp(-ti*gams);
     th = th -               4*xim*TMath::Sin(dm*ti) * ehalf;
     sterm = 2*(1-xre*xre-xim*xim)*TMath::Cos(dm*ti) * ehalf;
     thplu[i-1] = th + sterm;
     thmin[i-1] = th - sterm;
     sthplu += thplu[i-1];
     sthmin += thmin[i-1];
```

```
 }
  thp = thm = evp = evm = 0;
  if (iflag != 4) cout<<"          POSITIVE LEPTONS                    ,NEGATIVE LEPTONS"<<endl;
  if (iflag != 4) {
     cout<<"      TIME    THEOR    EXPTL    chisq         TIME    THEOR    EXPTL    chisq"<<endl;
  }
  for (i=1;i<=nbins; i++) {
     thplu[i-1] = thplu[i-1]*sevtp / sthplu;
     thmin[i-1] = thmin[i-1]*sevtm / sthmin;
     thp += thplu[i-1];
     thm += thmin[i-1];
     evp += evtp[i-1];
     evm += evtm[i-1];
```

// Sum over bins until at least four events found

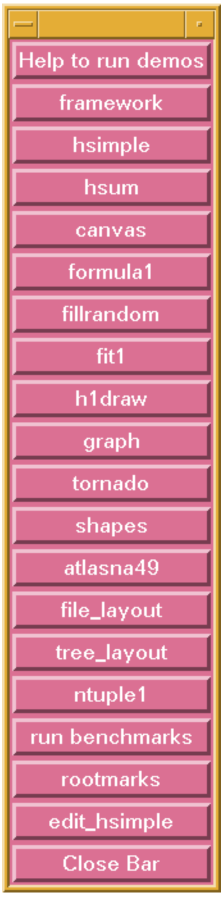
```
     if (evp > 3)  {
        ep   = evp-thp;
        chi1 = (ep*ep)/evp;
        chisq = chisq + chi1;
        if (iflag != 4) {
           cout<<Form(" %9.3f%9.3f%9.3f%9.3f",t[i-1],thp,evp,chi1)<<endl;
        }
        thp = evp = 0;
     }
     if (evm > 3) {
        em   = evm-thm;
        chi2 = (em*em)/evm;
        chisq = chisq + chi2;
        if (iflag != 4) {
           cout<<Form("                                          %9.3f%9.3f%9.3f%9.3f"
                          ,t[i-1],thm,evm,chi2)<<endl;
        }
        thm = evm = 0;
     }
 }
     f = chisq;
     ncount++;
```

// cout<<ncount<< " F= "<<f<<" xre="<<xre<<" xim="<<xim<<" dm="<<dm<<" gams="<<gams<<" gaml="<<gaml<<endl;

```
}
```

Example of a ControlBar menu

```
{
// To execute an item, click with the left mouse button.
// To see the HELP of a button, click on the right mouse button.
  gROOT.Reset("a");
  TControlBar bar("vertical");
  bar.AddButton("Help to run demos",".x demoshelp.C", " ");
  bar.AddButton("framework",   ".x framework.C", "An Example of Object Oriented User Interface");
  bar.AddButton("hsimple",     ".x hsimple.C",   "An Example Creating Histograms/Ntuples on File");
  bar.AddButton("hsum",        ".x hsum.C",      "Filling histograms and some graphics options");
  bar.AddButton("canvas",      ".x canvas.C",    "Canvas and Pad Management");
  bar.AddButton("formula1",    ".x formula1.C",  "Simple Formula and Functions");
  bar.AddButton("fillrandom",  ".x fillrandom.C","Histograms with Random Numbers from a Function");
  bar.AddButton("fit1",        ".x fit1.C",      "A Simple Fitting Example");
  bar.AddButton("h1draw",      ".x h1draw.C",    "Drawing Options for 1D Histograms");
  bar.AddButton("graph",       ".x graph.C",     "Examples of a simple graph");
  bar.AddButton("tornado",     ".x tornado.C",   "Examples of 3-D PolyMarkers");
  bar.AddButton("shapes",      ".x shapes.C",    "The Geometry Shapes");
  bar.AddButton("geomfile",    ".x geomfile.C",  "Creation of the NA49 geometry file");
  bar.AddButton("na49view",    ".x na49view.C",  "Two views of the NA49 detector geometry");
  bar.AddButton("file",        ".x file.C",      "The ROOT file format");
  bar.AddButton("fildir",      ".x fildir.C",    "The ROOT file, directories and keys");
  bar.AddButton("tree",        ".x tree.C",      "The Tree Data Structure");
  bar.AddButton("ntuple1",     ".x ntuple1.C",   "Ntuples and Selections");
  bar.AddButton("benchmarks",  ".x benchmarks.C", "Runs all the ROOT benchmarks");
  bar.AddButton("rootmarks",   ".x rootmarks.C", "Prints an estimated ROOTMARKS for your machine");
  bar.AddButton("Close Bar",   "gROOT.Reset("a")","Close ControlBar");
  bar.Show();
  gROOT.SaveContext();
}
```



The ROOT Collection Classes

Collections are a key feature of the ROOT system. Many, if not most, of the applications you write will use collections. If you have used polymorphic C++ collections before, some of this material will be review. However, much of this tutorial covers aspects of collections specific to the ROOT system. In this tutorial the following features will be demonstrated:

- How to create instances of collections
- The difference between lists, ordered collections, hashtables, maps, etc.
- How to add and remove elements of a collection
- How to search a collection for a specific element
- How to access and modify collection elements
- How to iterate over a collection
- How to manage memory for collections and collection elements
- How collection elements are tested for equality (`IsEqual()`)
- How collection elements are compared (`Compare()`) in case of sorted collections
- How collection elements are hashed (`Hash()`) in hash tables
- Understanding collections Using collections

Understanding Collections

A collection is a group of related objects. You will find it easier to manage a large number of items as a collection. For example, collections of points and lines might be managed by a graphics pad. A vertex will have a collection of tracks. A detector geometry contains collections of shapes, materials, rotation matrices and subdetectors. Collections act as flexible alternatives to traditional data structures of computer science such as arrays, lists, and trees.

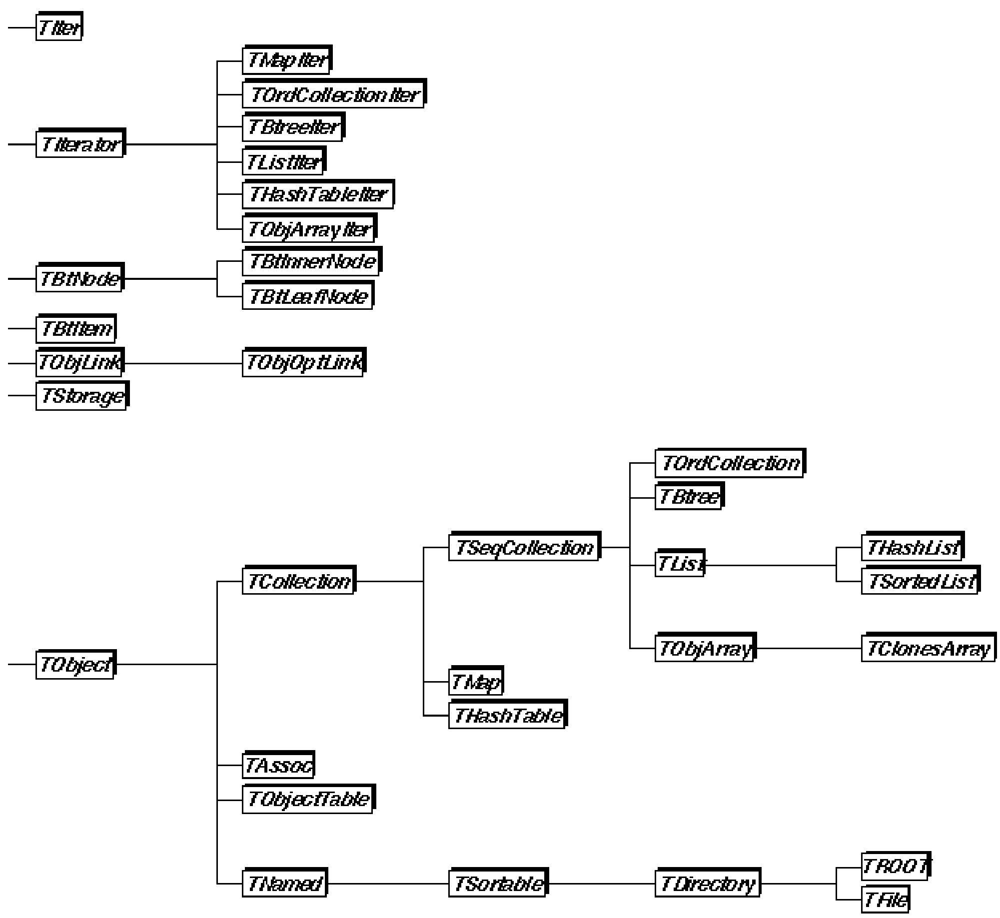
Collections can be thought of as polymorphic containers that can contain different types of elements. For this release of the ROOT system, elements to be placed in collections must be instances of classes. These may be classes defined by you or provided by ROOT. Collection elements must be instances of classes descending from TObject. The dependence of collections on TObject may disappear in the future when all C++ compilers used with the ROOT system fully support templates. In the mean time, knowing the role TObject plays in collections can be helpful.

In general you don't need to worry about TObject. Many ROOT classes have TObject as an ancestor. In fact, collections themselves are descendants of TObject. This makes it possible for collections to contain other collections (subcollections) in a tree structure. Such trees are used in the ROOT system to implement components of the graphics system (graphics pads containing pads), geometries (detectors in detectors), etc.

The basic protocol TObject defines for collection elements is shown below:

```
IsEqual()
Compare()
IsSortable()
Hash()
```

How to use and override these member functions is shown in the example program.



Types of Collections

The ROOT system implements the following type of collections: arrays, lists, sorted lists, B-trees, hashtables and maps. The figure below shows the inheritance hierarchy for the primary collection classes.

Ordered Collections (Sequences)

Sequences are collections that are externally ordered because they maintain internal elements according to the order in which they were added. The following sequence collections are available:

- TList
- THashList
- TOrdCollection
- TObjArray
- TClonesArray

Both a `TObjArray` as well as a `TOrdCollection` can be sorted using their `Sort()` member function (assuming the stored items are sortable).

Sorted Collections

Sorted collections are ordered by an internal (automatic) sorting mechanism. The following sorted collections are available:

- TSortedList
- TBtree

Unordered Collections

Unordered collections don't maintain the order in which the elements were added. I.e. when you iterate over an unordered collection, you are not likely to retrieve elements in the same order they were added to the collection. The following unordered collections are available:

- THashTable
- TMap

Adding Your Own Classes to ROOT

If you want to integrate your classes into the ROOT system, to enjoy features like, *extensive RTTI* and *ROOT object I/O*, you have to add the following line to your class header files (as part of the public part of the class definition):

ClassDef(*ClassName,ClassVersionID*) *//The class title*

See for example TLine.h. The **ClassVersionID** is used by the ROOT I/O system. It is written on the output stream and during reading you can check this version ID and take appropriate action depending on the value of the ID (see `Streamer`). Every time you change the layout of a class you should increase its **ClassVersionID** by one. The **ClassVersionID** should be **>=1**. Set **ClassVersionID=0** in case you don't need object I/O.

Similarly, in your implementation file you must add the statement:

ClassImp(ClassName)

See for example Line.C. Note that you *MUST* provide a default constructor for your classes, i.e. a constructor with zero parameters or with one or more parameters all with default values in case you want to use object I/O. If not you will get a compile time error.

The `ClassDef` and `ClassImp` macros are defined in the file Rtypes.h. This file is referenced by all ROOT include files, so you will automatically get them if you use a ROOT include file.

The `ClassDef` and `ClassImp` macros are necessary to link your classes to the dictionary generated by CINT to get access to the RTTI and object I/O features of ROOT. The RTTI system allows you to, a.o. find out to which class an object belongs, its baseclasses, its datamembers and methods, the method signatures, etc. This information is used to make advanced object browsers and by the automatic documentation generation system. The object I/O system allows you to store and retrieve objects (and arbitrarily complex object structures) from a ROOT database.

If you are ***only*** *interested in interactive access to your classes via the command line or macro processor you do not need to use the* `ClassDef` *and* `ClassImp` *macros.*

See "CINT as Dictionary Generator" for a description on how to generate a dictionary.

The CINT Dictionary Generator

To integrate your classes in the ROOT system and to benefit from the advanced RTTI and I/O features of ROOT and to get access to your classes via the C++ interpreter you have to generate a dictionary for your classes.

The dictionary will be generated using the program **rootcint** that comes with the RDK (Root Development Kit). Below follow two examples of how to use rootcint to generate a dictionary and how to compile and link this dictionary with your classes and the ROOT libraries.

Dictionary Generation for Interactive Access Without I/O and RTTI

The first example shows how a stand alone class, i.e. a class not inheriting from ROOT's `TObject` class can be made accessible to the interpreter for interactive manipulation.

We begin with a simple header file `MyClass.h` defining `class MyClass`:

```
//--------------------------------------------------
#ifndef __MyClass__
#define __MyClass__
class MyClass {
private:
  float   fX;     //x position in centimeters
  float   fY;     //y position in centimeters
public:
  MyClass();
  void    Print() const;
  void    SetX(float x) { fX = x; }
  void    SetY(float y) { fY = y; }
};
#endif //--------------------------------------------------
```

And its implementation file `MyClass.C`:

```
//--------------------------------------------------
#include <iostream.h>
#include "MyClass.h"
MyClass::MyClass()
{
  fX = -1;
  fY = -1;
}
void MyClass::Print() const
{
  cout << "fX = " << fX << ", fY = " << fY << endl;
} //--------------------------------------------------
```

To make this class accessible via the command line we need to link it with a small ROOT main program, `main.C`, that creates and calls the command line interpreter:

```
//--------------------------------------------------
#include "TROOT.h"
#include "TRint.h"
int Error; //left undefined by Motif
extern void  InitGui();  // initializer for GUI needed for interactive interface
VoidFuncPtr_t initfuncs[] = { InitGui, 0 };
// Initialize the ROOT system
TROOT root("Rint","The ROOT Interactive Interface", initfuncs);
int main(int argc, char **argv)
{
   // Create interactive interface
   TRint *theApp = new TRint("ROOT example", &argc, argv, NULL, 0);
   // Run interactive interface
   theApp->Run();
   return(0);
} //--------------------------------------------------
```

Using the following `Makefile` we can use `make` to build the program `myroot` that will give the user access to class `MyClass` via the C++ interpreter:

```
#--------------------------------------------------
CXXFLAGS      = -w -g +a1 -I$(ROOTSYS)/include
LDFLAGS       = -w -g +a1
LD            = CC
LIBS          = $(ROOTSYS)/lib/*.sl -lXm -lXt -lX11 -lm -lPW -ldld
HDRS          = MyClass.h
SRCS          = main.C MyClass.C mydict.C
OBJS          = main.o MyClass.o mydict.o
PROGRAM       = myroot
all:            $(PROGRAM)
```

```
$(PROGRAM):     $(OBJS)
               @echo "Linking $(PROGRAM) ..."
               @$(LD) $(LDFLAGS) $(OBJS) $(LIBS) -o $(PROGRAM)
               @echo "done"
clean:;         @rm -f $(OBJS) core
###
MyClass.o: MyClass.h
mydict.C: MyClass.h
       @echo "Generating dictionary ..."
       @rootcint mydict.C -c MyClass.h #-------------------------------------------------
```

This `Makefile` assumes that the environment variable `$ROOTSYS` has been correctly set (as described in the `AA_README` file that comes with the RDK) and that `$ROOTSYS/bin` has been added to `$PATH`. The compiler options are for HP-UX (for options for other platforms see the compile scripts in the `root/test` directory that comes with the RDK).

The line:

rootcint mydict.C -c MyClass.h

shows how **rootcint** is used to generate the dictionary files `mydict.h` and `mydict.C`. To get a full list and a description of all command line options supported by rootcint do:

rootcint -?

To see how to run `myroot` and to create and manipulate `MyClass` objects via the interpreter see: "CINT as Command Line and Macro Interpreter".

Dictionary Generation for Interactive Access Including I/O and RTTI

In this example we'll show how to fully integrate a set of classes into the ROOT system and how a shared library of these classes can be dynamically loaded into a running ROOTsession. We'll create an Event class which contains a list of Tracks. Here are the files `Event.h` and `Track.h`:

```
//--------------------------------------------------
#ifndef __Event__
#define __Event__
#include "TObject.h"
class TCollection;
class Track;
class Event : public TObject {
private:
  Int_t         fId;               //event sequential id
  Float_t       fTotalMom;         //total momentum
  TCollection  *fTracks;           //collection of tracks
public:
  Event() { fId = 0; fTracks = 0; }
  Event(Int_t id);
  ~Event();
  void    AddTrack(Track *t);
  Int_t   GetId() const { return fId; }
  Int_t   GetNoTracks() const;
  void    Print(Option_t *opt="");
  Float_t TotalMomentum();
  ClassDef(Event,1)  //Simple event class
};
#endif //--------------------------------------------------
//--------------------------------------------------
#ifndef __Track__
#define __Track__
#include "TObject.h"
class Event;
class Track : public TObject {
private:
  Int_t    fId;       //track sequential id
  Event   *fEvent;    //event to which track belongs
  Float_t  fPx;       //x part of track momentum
  Float_t  fPy;       //y part of track momentum
  Float_t  fPz;       //z part of track momentum
public:
  Track() { fId = 0; fEvent = 0; fPx = fPy = fPz = 0; }
  Track(Int_t id, Event *ev, Float_t px, Float_t py, Float_t pz);
  Float_t  Momentum() const;
  Event   *GetEvent() const { return fEvent; }
  void     Print(Option_t *opt="");
  ClassDef(Track,1)  //Simple track class
};
#endif //--------------------------------------------------
```

The first things to notice in these header files are the usage of the `ClassDef` macro and the default contructors of the `Event` and `Track` classes. Also notice the usage of comments to describe the data members and the comment after the `ClassDef` macro to describe the class. The intended usage of these classes is that one creates and event object with a certain id and then add tracks to the event. As one can see the track objects contain a pointer to the event to which they belong. This to show that the I/O system will correctly handle circular references. As an aside, note that although both header files contain references to each others objects there is no need to include the complete header files. A simple class declaration is enough ("`class Track;`"). This does not seem important now, but when a system grows it can save a lot of time during compilation when not every file that includes `Event.h` forces also the reading of `Track.h` or vice versa.

Next the implementation of these two classes. `Event.C`:

```
//----------------------------------------------------
#include <iostream.h>
#include "TOrdCollection.h"
#include "Event.h"
#include "Track.h"
ClassImp(Event)
Event::Event(Int_t id)
{
  fId = id;
  fTracks = new TOrdCollection;
}
Event::~Event()
{
  delete fTracks;
}
void Event::AddTrack(Track *t)
{
  fTracks->Add(t);
}
Int_t Event::GetNoTracks() const
{
  return fTracks->GetSize();
}
Float_t Event::TotalMomentum()
{
  TIter next(fTracks);
  Track *t;
  while (t = (Track *)next())
     fTotalMom += t->Momentum();
  return fTotalMom;
}
void Event::Print(Option_t *)
{
  cout << "*** Event=" << fId << " No of tracks=" << GetNoTracks() << endl;
  fTracks->Print();
} //----------------------------------------------------
```

And `Track.C`:

```
//----------------------------------------------------
#include <iostream.h>
#include "TMath.h"
#include "Track.h"
#include "Event.h"
ClassImp(Track)
Track::Track(Int_t id, Event *ev, Float_t px, Float_t py, Float_t pz)
{
  fId    = id;
  fEvent = ev;
  fPx    = px;
  fPy    = py;
  fPz    = pz;
}
Float_t Track::Momentum() const
{
  return TMath::Sqrt(fPx*fPx+fPy*fPy+fPz*fPz);
}
void Track::Print(Option_t *)
{
  cout << "id=" << fId << " event#=" << fEvent->GetId() << " px=" << fPx
       << " py=" << fPy << " pz=" << fPz << endl;
} //----------------------------------------------------
```

In the implementation files we notice the `ClassImp` macro's. Further, in `Event.C`, we see how we create a container class, `TOrdCollection`, and how we iterate over the collection using a `TIter` object. Note also how in `Event.h` we did not specify what kind of container we were going to use. Since all containers inherit from `TCollection` we are free to choose in the implementation file the collection with the right properties for the job. We don't have to change the header in case we want to use another container class.

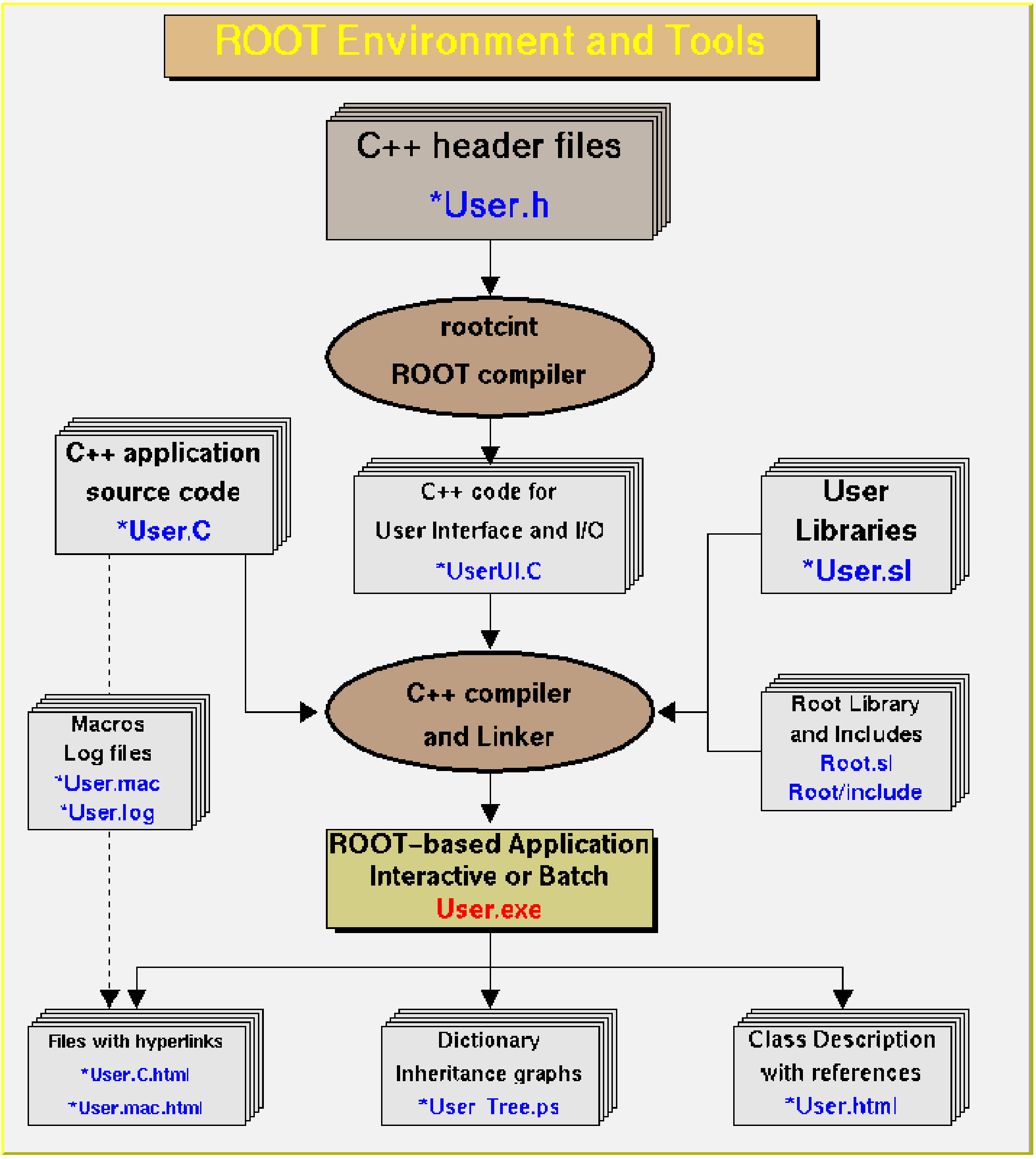
To create the `event.sl` shared library on HP-UX we use the following `Makefile`:

```
#---------------------------------------------------
CXXFLAGS      = -w -g +a1 +Z -I$(ROOTSYS)/include
LDFLAGS       = -w -g +a1 -b
LD            = CC
HDRS          = Event.h Track.h eventdict.h
SRCS          = Event.C Track.C eventdict.C
OBJS          = Event.o Track.o eventdict.o
PROGRAM       = event.sl
all:            $(PROGRAM)
$(PROGRAM):     $(OBJS)
               @echo "Linking $(PROGRAM) ..."
               @/bin/rm -f $(PROGRAM)
               @$(LD) $(LDFLAGS) $(OBJS) -o $(PROGRAM)
               @chmod 555 $(PROGRAM)
               @echo "done"
clean:;         @rm -f $(OBJS) core
###
Event.o: Event.h
Track.o: Track.h
eventdict.C: Event.h Track.h
       @echo "Generating dictionary ..."
       @rootcint eventdict.C -c Event.h Track.h #---------------------------------------------------
```

After running `make` have a look at the file `eventdict.C`. At the bottom of this file we see the `TBuffer &operator>>()`, `Streamer()` and `ShowMembers()` methods for our two classes, all automatically generated by **rootcint**. This is all there is to get extensive RTTI and full ROOT object I/O. No need for seperate IDL files or meta header files describing the class structure.

To see how to load `event.sl` in a running ROOT process and how to create, write and read events and tracks see: "Extending ROOT with Shared Libraries and an Example of Object I/O"

The figure below depicts the different stages and files involved in the dictionary creation process described above.


ROOT Environment and Tools
C++ header files
*User.h
rootcint
ROOT compiler
C++ application
source code
*User.C
C++ code for
User Interface and I/O
*UserUI.C
User
Libraries
*User.sl
Macros
Log files
*User.mac
*User.log
C++ compiler
and Linker
Root Library
and Includes
Root.sl
Root/include
ROOT-based Application
Interactive or Batch
User.exe
Files with hyperlinks
*User.C.html
*User.mac.html
Dictionary
Inheritance graphs
*User_Tree.ps
Class Description
with references
*User.html

The CINT Interpreter Interface

CINT as embedded in ROOT can be used as command line interpreter and as macro processor, where macros are "small" (up to at least 60000 loc) C++ programs. Thanks to CINT the ROOT system can present the user a single language environment: C++ as implementation, macro and command line language.

The advantages of a single language model are clear. Especially when writing macros. Typically macros start as small prototypes that need frequent modification. While execution speed is not important a short edit-execute cycle is. However, once macros have grown to full programs and have become stable, the need for fast execution in production jobs becomes important. Thanks to the fact that the macros are in "standard" C++ we can simply compile and dynamically link the macros with the ROOT system and execute them at full speed.

First we will show how to use the ROOT/CINT command line interpreter and next how to execute C++ macros. Along the way we'll describe the very few extensions made to CINT's C++ to facilitate a smooth integration with ROOT.

CINT as Command Line Interpreter

Lets run the program `myroot` that we created in "CINT as Dictionary Generator" and see how we can create and manipulate `MyClass` objects via the interpreter (for brevity's sake some non essential output has been removed):

```
$ myroot
 *******************************************
 *                                         *
 *        W E L C O M E  to  R O O T       *
 *                                         *
 *   Version   0.90/12  18 January 1997    *
 *                                         *
 *  You are welcome to visit our Web site  *
 *          http://root.cern.ch            *
 *                                         *
 *******************************************
CINT/ROOT C/C++ Interpreter version 5.11.29, Dec 5 1996
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] MyClass a
root [1] a.Print()
fX = -1, fY = -1
root [2] a.SetX(10)
root [3] a.SetY(11)
root [4] a.Print()
fX = 10, fY = 11
root [5] .g
...
...
0x4045c7f8 class MyClass a , size=8
 0x0        private: float fX //x position in centimeters
 0x0        private: float fY //y position in centimeters
root [6] .class MyClass
===============================================================
class MyClass
size=0x8
(tagnum=438,voffset=-1,isabstract=0,parent=-1,gcomp=0,=~cd=0)
List of base class--------------------------------------------------------
List of member variable---------------------------------------------------
Defined in MyClass
0x0        private: float fX //x position in centimeters
0x0        private: float fY //y position in centimeters
List of member function---------------------------------------------------
Defined in MyClass
filename       line:size busy function type and name
(compiled)        0:0    0 public: class MyClass MyClass(void);
(compiled)        0:0    0 public: void Print(void);
(compiled)        0:0    0 public: void SetX(float x);
(compiled)        0:0    0 public: void SetY(float y);
(compiled)        0:0    0 public: class MyClass MyClass(class MyClass&);
(compiled)        0:0    0 public: void ~MyClass(void); root [7] .q
```

In the above example we first create an object `a` of class `MyClass`. Next we call the `Print()` method (and notice that the constructor has been executed correctly), set some variables and call again the `Print()` method. Everything works as expected. Notice that when issueing single line commands we may omit the trailing "`;`".

Next we execute a "raw" interpreter command `.g` to list all global symbols currently defined in the interpreter followed by the command `.class` *classname* to show the definition of `MyClass` as known to the interpreter. We exit the session by typing `.q`. Here we notice that all interpreter commands start with a "`.`", except for the "`?`" command which you can use to get a list of all available interpreter commands.

The ROOT command line interface supports emacs style command line editing. Also the command history is saved between sessions.

Now lets execute a multi line command. We start again `myroot` and type:

```
root [0] {
end with '}'> MyClass a;
end with '}'> for (int i = 0; i < 5; i++) {
end with '}'>    a.SetX(i);
end with '}'>    a.SetY(i+1);
end with '}'>    a.Print();
end with '}'> }
end with '}'> }
fX = 0, fY = 1
fX = 1, fY = 2
fX = 2, fY = 3
fX = 3, fY = 4
fX = 4, fY = 5 root [1] .q
```

A multi line command starts always with a "`{`" and ends with a "`}`". As long as your are in multi line input mode you are prompted with "`end with '}'>`". After typing the closing "`}`" the command will be executed. Notice that now you have to terminate every line, like in C++, with a "`;`". Inserting many lines in this way requires typing without making any errors. There is no way to correct a once entered line. Therefore it is much easier to put such a sequence of commands in a macro file.

CINT as Macro Processor

CINT macro files contain pure C++ code. They can contain a simple sequence of statements like in the multi command line example given above, but also arbitrarily complex class and function definitions.

Lets start with a macro containing a simple list of statements (like the multi command line example given in the previous section). This type of macro must start with a "`{`" and end with a "`}`". Assume the file is called `macro1.C`:

```
//--------------------------------------------------
{
#include <iostream.h>
  cout << " Hello" << endl;
  float x = 3.;
  float y = 5.;
  int   i = 101;
  cout << " x = " << x << " y = " << y << " i = " << i << endl;
  return 0;
} //--------------------------------------------------
```

To execute the stream of statements in `macro1.C` do:

```
root [1] .x macro1.C       // loads the contents of file macro1.C and
                          // executes all statements in the global scope
```

One can re-execute the statements by re-issueing "`.x macro1.C`" (since there is no function entry point).

Now copy file `macro1.C` to `macro2.C` and add a function statement. Like this:

```
//--------------------------------------------------
#include <iostream.h>
int main()
{
  cout << " Hello" << endl;
  float x = 3.;
  float y = 5.;
  int   i= 101;
  cout << " x = " << x << " y = " << y << " i = " << i << endl;
  return 0;
} //--------------------------------------------------
```

Notice that no surrounding "`{`" and "`}`" are required in this case.

To execute function `main()` in `macro2.C` do:

```
root [1] .L macro2.C       // load the contents of file macro2.C in memory
root [2] main()            // execute entry point main()
Hello
x = 3 y = 5 i = 101
(int)0
root [3] main()            // execute main() again
```

```
Hello
x = 3 y = 5 i = 101
(int)0
root [4] .func             // list all functions known by CINT
filename       line:size busy function type and name
...
... macro2.C 4:9 0 public: int main();
```

The last command shows that `main()` has been loaded from file `macro2.C`, that the function `main()` starts on line 4 and is 9 lines long. Notice that once a function has been loaded it becomes part of the system just like a compiled function.

Now we copy file `macro2.C` to `macro3.C` and change the function name from `main()` to `macro3(int j = 10)`:

```
//------------------------------------------------
#include <iostream.h>
int macro3(int j = 10)
{
  cout << " Hello" << endl;
  float x = 3.;
  float y = 5.;
  int   i = j;
  cout << " x = " << x << " y = " << y << " i = " << i << endl;
  return 0;
} //------------------------------------------------
```

To execute `macro3()` in `macro3.C` type:

```
root [1] .x macro3.C(8)    // loads the contents of file macro3.C and // executes entry point
macro3(8)
```

Note that the above only works when the filename (minus extension) and function entry point are both the same. This shortcut is mainly interesting to run large one-off macros where one wants to pass arguments at the same time (instead of doing the "`.L`" step first). Function `macro3()` can still be executed multiple times:

```
root [2] macro3()
Hello
x = 3 y = 5 i = 10
(int)0
root [3] macro3(33)
Hello
x = 3 y = 5 i = 33 (int)0
```

A Macro Containing a Class Definition

As an example of a macro containing class definitions we take the small class `MyClass` that we used in the " CINT as Dictionary Generator" chapter and we introduce a subclass `Child`:

```
//------------------------------------------------
#include <iostream.h>
class MyClass {
private:
  float   fX;     //x position in centimeters
  float   fY;     //y position in centimeters
public:
  MyClass() { fX = fY = -1; }
  virtual void Print() const;
  void         SetX(float x) { fX = x; }
  void         SetY(float y) { fY = y; }
};
void MyClass::Print() const
{
  cout << "fX = " << fX << ", fY = " << fY << endl;
}
class Child : public MyClass {
public:
  void Print() const;
};
void Child::Print() const
{
  cout << "This is Child::Print()" << endl;
  MyClass::Print();
} //------------------------------------------------
```

The only changes we made was to put `MyClass.h` and the `MyClass.C` files together in file `macro4.C` and add the `Child` class. To make things a little less trivial we made `MyClass::Print()` virtual and overrode it in the `Child` class.

To execute `macro4.C` do:

```
root [0] .L macro4.C
```

```
root [1] MyClass *a = new Child
root [2] a.Print()
This is Child::Print()
fX = -1, fY = -1
root [3] a.SetX(10)
root [4] a.SetY(12)
root [5] a.Print()
This is Child::Print()
fX = 10, fY = 12
root [6] .class MyClass
==========================================================================
class MyClass
size=0x8 FILE:macro4.C LINE:3
(tagnum=459,voffset=-1,isabstract=0,parent=-1,gcomp=0,=~cd=1)
List of base class--------------------------------------------------------
List of member variable---------------------------------------------------
Defined in MyClass
0x0        private: float fX
0x4        private: float fY
List of member function---------------------------------------------------
Defined in MyClass
filename       line:size busy function type and name
macro4.C         16:5    0 public: class MyClass MyClass(void);
macro4.C         22:4    0 public: void Print(void);
macro4.C         12:1    0 public: void SetX(float x);
macro4.C         13:1    0 public: void SetY(float y); root [7] .q
```

As you can see an interpreted class behaves just like a compiled class (e.g. like `MyClass` in `myroot`).

Reseting the Interpreter Environment

Loading files and executing functions will in general result in many objects being created on the interpreter's internal stack. Once in a while it might be necessary to reset the interpreter environment. There are basically two way to reset the environment:

• Using the interpreter command `.reset` or

• Using the ROOT function `gROOT.Reset()`

The first method clears the complete interpreter environment, while the second method clears the environment to the status just before executing the last macro. Therefore multi command line macros often start with the statement `gROOT.Reset()` (where `gROOT` is a pointer to the global ROOT object, see also `TROOT`).

Debugging Macros

A powerful feature of CINT is the ablity to debug interpreted methods and functions by means of setting breakpoints and being able to single step through the code and print variable values on the way. Assume we have `macro4.C` still loaded, we can then do:

```
root [1] .b Child::Print
Break point set to line 26 macro4.C
root [2] a.Print()
26   Child::Print() const
27   {
28      cout << "This is Child::Print()" <<endl;
FILE:macro4.C LINE:28 cint> .s
311  operator<<(ostream& ostr,G__CINT_ENDL& i) {return(endl(ostr));
FILE:iostream.h LINE:311 cint> .s
}
This is Child::Print()
29      MyClass::Print();
FILE:macro4.C LINE:29 cint> .s
16   MyClass::Print() const
17   {
18      cout << "fX = " << fX << ", fY = " << fY << endl;
FILE:macro4.C LINE:18 cint> .p fX
(float)1.000000000000e+01
FILE:macro4.C LINE:18 cint> .s
311  operator<<(ostream& ostr,G__CINT_ENDL& i) {return(endl(ostr));
FILE:iostream.h LINE:311 cint> .s
}
fX = 10, fY = 12
19   }
30   }
2    } root [3] .q
```

In the above debug session we first set a breakpoint on the `Child::Print()` method. Then we execute the method. When we hit the breakpoint CINT returns a prompt and we can issue any of the debug commands (for a full list of debug commands, type a `?` at the ROOT prompt). In the above example we single step till we are in `MyClass::Print()` where we examine the value of `fX`. We can only set breakpoints on or in functions, therefore macros without function statement can not be debugged (like simple multi command line macros). In such cases just wrap the sequence of statements in a function.

ROOT/CINT Extensions to C++

In the next example we demonstrate three of the most important extensions ROOT/CINT makes to C++. Start basic `root` or `myroot` in the directory `root/tutorials` that comes with the RDK (make sure the file `hsimple.root` is there):

```
root [1] f = new TFile("hsimple.root")
(class TFile*)0x4045e690
root [2] f.ls()
TFile**         hsimple.root
TFile*         hsimple.root
 KEY: TH1F     hpx;1   This is the px distribution
 KEY: TH2F     hpxpy;1 py ps px
 KEY: THProfile        hprof;1 Profile of pz versus px
 KEY: TNtuple  ntuple;1        Demo ntuple
root [3] hpx.Draw()
NULL
Warning in <MakeDefCanvas>: creating a default canvas with name c1 root [4] .q
```

The first command shows the first extension; the declaration of "`f`" may be omitted when "`new`" is used. CINT will correctly create "`f`" as object of class "`TFile`".

The second extension is shown in the second command. Although "`f`" is a pointer to "`TFile`" we don't have to use the pointer dereferencing syntax "`->`" but can use the simple "." notation (since every object created by CINT is a heap object so CINT does not need the distinction between heap and stack based objects).

The third extension is more important. In case CINT can not find an object being referenced it will ask ROOT to search for an object with an identical name in the search path defined by `TROOT::FindObject()`. If ROOT finds the object it returns CINT a pointer to this object and a pointer to its class definition and CINT will execute the requested member function. This shortcut is quite natural for an interactive system and saves a lot of typing, e.g.:

```
root [4] TH1 *hpx = (TH1*)gROOT.FindObject("hpx") root [5] hpx.Draw()
```

Of course when writing large macros, it is best to stay away from these shortcuts since otherwise you will later have problems compiling your macros using a real C++ compiler.

Example of Client-Server communication: the Client

```
{

// Client program which creates and fills a histogram. Every 1000 fills
// the histogram is send to the server which displays the histogram.

// To run this demo do the following:
// - Open three windows
// - Start ROOT in all three windows
// - Execute in the first window: .x hserv.C
// - Execute in the second and third windows: .x hclient.C
// If you want to run the hserv.C on a different host, just change
// "localhost" in the TSocket ctor below to the desried hostname.

  gROOT->Reset();
  gBenchmark->Start("hclient");

// Open connection to server

  TSocket *sock = new TSocket("localhost", 9090);

// Wait till we get the start message

  char str[32];
  sock->Recv(str, 32);

// server tells us who we are

  int idx = !strcmp(str, "go 0") ? 0 : 1;
  if (idx == 0) {

// Create the histogram

     hpx = new TH1F("hpx","This is the px distribution",100,-4,4);
     hpx->SetFillColor(48);  // set nice fillcolor
  } else {
     hpx = new TH2F("hpxpy","py vs px",40,-4,4,40,-4,4);
  }
  TMessage mess(kMESS_OBJECT);

//TMessage mess(kMESS_OBJECT | kMESS_ACK);

// Fill histogram randomly

  gRandom->SetSeed();
  Float_t px, py;
  const int kUPDATE = 1000;
  for (int i = 0; i < 25000; i++) {
     gRandom->Rannor(px,py);
     if (idx == 0)
        hpx->Fill(px);
     else
        hpx->Fill(px,py);
     if (!i%kUPDATE) {
        mess.Reset();              // re-use TMessage object
        mess.WriteObject(hpx);     // write object in message buffer
        sock->Send(mess);          // send message
     }
  }
  sock->Send("Finished");          // tell server we are finished
  gBenchmark->Show("hclient");

// Close the socket

  sock->Close();
}
```

Example of Server-Server communication: the Server

```
{
```

// Server program which waits for two clients to connect. It then monitors
// the sockets and displays the objects it receives.

// To run this demo do the following:
// - Open three windows
// - Start ROOT in all three windows
// - Execute in the first window: .x hserv.C
// - Execute in the second and third windows: .x hclient.C
// If you want to run the hserv.C on a different host, just change
// "localhost" in the TSocket ctor below to the desried hostname.

// Open a server socket looking for connections on a named service or
// on a specified port.
//TServerSocket *ss = new TServerSocket("rootserv", kTRUE);

```
  TServerSocket *ss = new TServerSocket(9090, kTRUE);
```

// Accept a connection and return a full-duplex communication socket.

```
  TSocket *s0 = ss->Accept();
  TSocket *s1 = ss->Accept();
```

// tell the clients to start

```
  s0->Send("go 0");
  s1->Send("go 1");
```

// Close the server socket (unless we will use it later to wait for
// another connection).

```
  ss->Close();
```

// Check some options of socket 0.

```
  int val;
  s0->GetOption(kSendBuffer, val);
  printf("sendbuffer size: %d
", val);
  s0->GetOption(kRecvBuffer, val);
  printf("recvbuffer size: %d
", val);
```

// Get the remote addresses (informational only).

```
  TInetAddress adr = s0->GetInetAddress();
  adr.Print();
  adr = s1->GetInetAddress();
  adr.Print();
```

// Create canvas and pads to display the histograms

```
  TCanvas *c1 = new TCanvas("c1","The Ntuple canvas",200,10,700,780);
  TPad *pad1 = new TPad("pad1","This is pad1",0.02,0.52,0.98,0.98,21);
  TPad *pad2 = new TPad("pad2","This is pad2",0.02,0.02,0.98,0.48,21);
  pad1->Draw();
  pad2->Draw();
  TMonitor *mon = new TMonitor;
  mon->Add(s0);
  mon->Add(s1);
  while (1) {
     TMessage *mess;
     TSocket  *s;
     s = mon->Select();
     s->Recv(mess);
     if (mess->What() == kMESS_STRING) {
        char str[64];
        mess->ReadString(str, 64);
```

//printf("Client %d: %s

", s==s0 ? 0 : 1, str);

```
        printf("Client %d
", s==s0 ? 0 : 1);
        printf("Client %s
", str);
        mon->Remove(s);
        if (mon->GetActive() == 0) {
           printf("No more active clients... stopping
");
           break;
        }
     } else if (mess->What() == kMESS_OBJECT) {
```

//printf("got object of class: %s
", mess->GetClass()->GetName());

```
        TH1 *h = (TH1 *)mess->ReadObject(mess->GetClass());
        if (s == s0)
           pad1->cd();
        else
           pad2->cd();
        h->Print();
        h->DrawCopy();  // draw a copy of the histogram, not the histo itself
        c1->Modified();
        c1->Update();
        delete h;       // delete histogram
     } else {
        printf("*** Unexpected message ***
");
     }
     delete mess;
  }
```

// Close the socket.

```
  s0->Close();
  s1->Close();
}
```

Example of Shared memory: Producer

```
{

// Histogram producer script. This script creates a memory mapped
// file and stores three histogram objects in it (a TH1F, a TH2F and a
// TProfile). It then fills, in an infinite loop (so use ctrl-c to
// stop this script), the three histogram objects with random numbers.
// Every 10 fills the objects are updated in shared memory.
// Use the hcons.C script to map this file and display the histograms.

  gROOT->Reset();

// Create a new memory mapped file. The memory mapped file can be
// opened in an other process on the same machine and the objects
// stored in it can be accessed.

  mfile = TMapFile::Create("hsimple.map","RECREATE", 100000,
                           "Demo memory mapped file with histograms");

// Create a 1d, a 2d and a profile histogram. These objects will
// be automatically added to the current directory, i.e. mfile.

  hpx    = new TH1F("hpx","This is the px distribution",100,-4,4);
  hpxpy  = new TH2F("hpxpy","py vs px",40,-4,4,40,-4,4);
  hprof  = new TProfile("hprof","Profile of pz versus px",100,-4,4,0,20);

// Set a fill color for the TH1F

  hpx->SetFillColor(48);

// Print status of mapped file

  mfile->Print();

// Endless loop filling histograms with random numbers

  Float_t px, py, pz;
  int ii = 0;
  while (1) {
     gRandom->Rannor(px,py);
     pz = px*px + py*py;
     hpx->Fill(px);
     hpxpy->Fill(px,py);
     hprof->Fill(px,pz);
     if (!(ii % 10)) {
        mfile->Update();       // updates all objects in shared memory
        if (!ii) mfile->ls();  // print contents of mapped file after
     }                         // first update
     ii++;
  }
}
```

Example of Shared memory: Consumer

```
{
```

// Histogram consumer script. Create a canvas and 3 pads. Connect
// to memory mapped file "hsimple.map", that was created by hprod.C.
// It reads the histograms from shared memory and displays them
// in the pads (sleeping for 0.1 seconds before starting a new read-out
// cycle). This script runs in an infinite loop, so use ctrl-c to stop it.

```
  gROOT->Reset();
```

// Create a new canvas and 3 pads

```
  TCanvas *c1;
  TPad *pad1, *pad2, *pad3;
  if (!gROOT->IsBatch()) {
     c1 = new TCanvas("c1","Shared Memory Consumer Example",200,10,700,780);
     pad1 = new TPad("pad1","This is pad1",0.02,0.52,0.98,0.98,21);
     pad2 = new TPad("pad2","This is pad2",0.02,0.02,0.48,0.48,21);
     pad3 = new TPad("pad3","This is pad3",0.52,0.02,0.98,0.48,21);
     pad1->Draw();
     pad2->Draw();
     pad3->Draw();
  }
```

// Open the memory mapped file "hsimple.map" in "READ" (default) mode.

```
  mfile = TMapFile::Create("hsimple.map");
```

// Print status of mapped file and list its contents

```
  mfile->Print();
  mfile->ls();
```

// Create pointers to the objects in shared memory.

```
  TH1F     *hpx    = 0;
  TH2F     *hpxpy  = 0;
  TProfile *hprof  = 0;
```

// Loop displaying the histograms. Once the producer stops this
// script will break out of the loop.

```
  Int_t oldentries = 0;
  while (1) {
     hpx    = (TH1F *) mfile->Get("hpx", hpx);
     hpxpy  = (TH2F *) mfile->Get("hpxpy", hpxpy);
     hprof  = (TProfile *) mfile->Get("hprof", hprof);
     if (hpx->GetEntries() == oldentries) break;
     oldentries = hpx->GetEntries();
     if (!gROOT->IsBatch()) {
        pad1->cd();
        hpx->Draw();
        pad2->cd();
        hprof->Draw();
        pad3->cd();
        hpxpy->Draw("cont");
        c1->Modified();
        c1->Update();
     } else {
        printf("Entries, hpx=%d, Mean=%g, RMS=%g
",hpx->GetEntries(),hpx->GetMean(),hpx->GetRMS());
     }
     gSystem->Sleep(100);   // sleep for 0.1 seconds
  }
}
```

Extending ROOT with Shared Libraries

and an Example of Object I/O

In this tutorial we will show how to extend a running ROOT module with shared libraries. Also we'll show how to create a ROOT database and how to save and retrieve your own object hierachies.

This tutorial uses the shared library `event.sl` containing the `Event` and `Track` classes created in "The CINT Dictionary Generator" page.

Loading a shared library in a running ROOT session is trivial, first start `root`, then type:

```
root [0] gSystem.Load("event.sl") (int)0
```

This is all there is to it to load a shared library into a running ROOT session. A return value of 0 means that the load was successful. Now lets verify that the classes `Event` and `Track` can be seen by the system:

```
root [1] .class Event
===========================================================================
class Event //Simple event class
size=0x18
(tagnum=443,voffset=-1,isabstract=0,parent=-1,gcomp=0,=~cd=0)
List of base class--------------------------------------------------------
0x0        public: TObject //Basic ROOT object
List of member variable---------------------------------------------------
Defined in Event
0x0        private: Int_t fId //event sequential id
0x0        private: Float_t fTotalMom //total momentum
0x0        private: class TCollection* fTracks //collection of tracks
0x7ad8fa80 static class TClass* fgIsA
List of member function---------------------------------------------------
Defined in Event
filename       line:size busy function type and name
(compiled)        0:0    0 public: class Event Event(void);
(compiled)        0:0    0 public: class Event Event(const Int_t id);
(compiled)        0:0    0 public: void AddTrack(class Track *const t);
(compiled)        0:0    0 public: Int_t GetId(void) const;
(compiled)        0:0    0 public: Int_t GetNoTracks(void) const;
(compiled)        0:0    0 public: virtual void Print(Option_t* opt);
(compiled)        0:0    0 public: Float_t TotalMomentum(void);
(compiled)        0:0    0 public: const char* DeclFileName(void);
(compiled)        0:0    0 public: int DeclFileLine(void);
(compiled)        0:0    0 public: const char* ImplFileName(void);
(compiled)        0:0    0 public: int ImplFileLine(void);
(compiled)        0:0    0 public: Version_t Class_Version(void);
(compiled)        0:0    0 public: virtual class TClass* IsA(void) const;
(compiled)        0:0    0 public: virtual void ShowMembers(class TMemberInspector& insp,char*
parent);
(compiled)        0:0    0 public: virtual void Streamer(class TBuffer& b);
(compiled)        0:0    0 public: void Dictionary(void);
(compiled)        0:0    0 public: class Event Event(class Event&);
(compiled)        0:0    0 public: void ~Event(void);
root [2] gClassTable.Print()
Defined classes
class                          version  initialized
===================================================
Event                                1       No
... ...
```

The command "`.class` *classname*" shows a class definition as known by the builtin CINT interpreter. To check that the classes were properly added to the ROOT system (i.e. not only known to the interpreter, but also to ROOT) we can print the contents of the global class table using the command "`gClassTable.Print()`". If the classes appear in the `.class` listing but not in the class table change is that you forgot to insert the `ClassImp` macro in the class implementation file. In the output of the `.class` command we see also how the system uses the comments you provided in the class definition.

When loading a shared library using `gSystem.Load()` the system will look for the library in the `Root.DynamicPath` search path. On how to set this search path in your `.rootrc` config file is described in the `TEnv` class description.

Now lets play a bit with our Event and Track classes.

René Brun, Nenad Buncic & Fons Rademakers Last update 28/8/96 by FR

Automatic HTML document generation

```
{
```

// All ROOT tutorials as well as the class descriptions have been
// generated automatically by ROOT itself via the services of
// the THtml class. Please read this class description and Coding Conventions.
// The following macro illustrates how to generate the html code
// for one class using the Make function.
// This example also shows how to convert to html a macro, including
// the generation of a "gif" file produced by the macro.

// How to generate HTML files for a single class
// (in this example class name is TBRIK), ...

```
gHtml->MakeClass("TBRIK")
```

// and how to generate html code for all classes, including an index.

//gHtml->MakeAll();

// execute a macro

```
.x something.C
```

// Invoke the TSystem class to execute a shell script.
// Here we call the "xpick" program to capture the graphics window
// into a gif file.

```
gSystem->Exec("xpick html/gif/shapes.gif")
```

// Convert this macro into html

```
gHtml->Convert("htmlex.C","Automatic HTML document generation")
```

// The following line is an example of comment showing how
// to include HTML instructions in a comment line.

// here is the output

Example of use of collection classes

//*CMZ : 0.90/09 03/12/96 17.46.36 by Ren\x8e Brun
//*-- Author : Fons Rademakers 19/08/96

```
#include <stdlib.h>
#include <iostream.h>
#include "TROOT.h"
#include "TString.h"
#include "TObjString.h"
#include "TSortedList.h"
#include "TObjArray.h"
#include "TOrdCollection.h"
#include "THashTable.h"
#include "TBtree.h"
#include "TStopwatch.h"
```

// To focus on basic collection protocol, this sample program uses
// simple classes inheriting from TObject. One class, TObjString, is a
// collectable string class (a TString wrapped in a TObject) provided
// by the ROOT system. The other class we define below, is an integer
// wrapped in a TObject, just like TObjString.

// TObjNum is a simple container for an integer.

```
class TObjNum : public TObject {
private:
  int  num;
public:
  TObjNum(int i = 0) : num(i) { }
  ~TObjNum() { Printf("~TObjNum = %d", num); }
  void    SetNum(int i) { num = i; }
  int     GetNum() { return num; }
  void    Print(Option_t *) { Printf("TObjNum = %d", num); }
  ULong_t  Hash() { return num; }
  Bool_t IsEqual(TObject *obj) { return num == ((TObjNum*)obj)->num; }
  Bool_t  IsSortable() const { return kTRUE; }
  Int_t   Compare(TObject *obj) { if (num == ((TObjNum*)obj)->num)
                                     return 0;
                                  else if (num < ((TObjNum*)obj)->num)
                                     return -1;
                                  else
                                     return 1; }
};
void Test_TObjArray()
{
  Printf(
  "///////////////////////////////////////////////////////////////////
"
  "// Test of TObjArray                                             //
"
  "///////////////////////////////////////////////////////////////////"
  );
```

// Initialize the ROOT framework

```
  TROOT tcollection("Collection","Test collection classes");
```

// Array of capacity 10 automatically expanded if necessary.

```
  TObjArray  a(10);
  Printf("Filling TObjArray");
  a.Add(new TObjNum(1));            // add at next free slot, pos 0
  a[1] = new TObjNum(2);            // use operator[], put at pos 1
  TObjNum *n3 = new TObjNum(3);
  a.AddAt(n3,2);                    // add at position 2
  a.Add(new TObjNum(4));            // add at next free slot, pos 3
  a.AddLast(new TObjNum(10));       // add at pos 4
  TObjNum n6(6);                    // stack based TObjNum
  a.AddAt(&n6,5);                   // add at pos 5
  a[6] = new TObjNum(5);            // add at respective positions
  a[7] = new TObjNum(8);
  a[8] = new TObjNum(7);
```

// a[10] = &n6; // gives out-of-bound error

```
  Printf("Print array");
  a.Print();                        // invoke Print() of all objects
  Printf("Sort array");
  a.Sort();
  for (int i = 0; i < a.Capacity(); i++)  // typical way of iterating over array
     if (a[i])
        a[i]->Print();      // can also use operator[] to access elements
     else
        Printf("%d empty slot", i);
  Printf("Use binary search to get position of number 6");
  Printf("6 is at position %d", a.BinarySearch(&n6));
  Printf("Find number before 6");
  a.Before(&n6)->Print();
  Printf("Find number after 3");
  a.After(n3)->Print();
  Printf("Remove 3 and print list again");
  a.Remove(n3);
  delete n3;
  a.Print();
  Printf("Iterate forward over list and remove 4 and 7");
```

// TIter encapsulates the actual class iterator. The type of iterator // used depends on the type of the collection.

```
  TIter next(&a);
  TObjNum *obj;
  while (obj = (TObjNum*)next())     // iterator skips empty slots
     if (obj->GetNum() == 4) {
        a.Remove(obj);
        delete obj;
     }
```

// Reset the iterator and loop again

```
  next.Reset();
  while (obj = (TObjNum*)next())
     if (obj->GetNum() == 7) {
        a.Remove(obj);
        delete obj;
     }
  Printf("Iterate backward over list and remove 2");
  TIter next1(&a, kIterBackward);
  while (obj = (TObjNum*)next1())
     if (obj->GetNum() == 2) {
        a.Remove(obj);
        delete obj;
     }
  Printf("Delete remainder of list: 1,5,8,10 (6 not deleted since not on heap)");
```

// Delete heap objects and clear list. Attention: do this only when you // own all objects stored in the collection. When you stored aliases to // the actual objects (i.e. you did not create the objects) use Clear() // instead.

```
  a.Delete();
  Printf("Delete stack based objects (6)");
}
void Test_TOrdCollection()
{
  Printf(
  "//////////////////////////////////////////////////////////////////
"
  "// Test of TOrdCollection                                      //
"
  "//////////////////////////////////////////////////////////////////"
  );
```

// Create collection with default size, Add() will automatically expand // the collection if necessary.

```
  TOrdCollection  c;
  Printf("Filling TOrdCollection");
  c.Add(new TObjString("anton"));      // add at next free slot, pos 0
  c.AddFirst(new TObjString("bobo"));  // put at pos 0, bump anton to pos 1
  TObjString *s3 = new TObjString("damon");
  c.AddAt(s3,1);                       // add at position 1, bump anton to pos 2
  c.Add(new TObjString("cassius"));    // add at next free slot, pos 3
  c.AddLast(new TObjString("enigma")); // add at pos 4
  TObjString s6("fons");               // stack based TObjString
  c.AddBefore(s3,&s6);                 // add at pos 1
  c.AddAfter(s3, new TObjString("gaia"));
  Printf("Print collection");
```

```
  c.Print();                              // invoke Print() of all objects
  Printf("Sort collection");
  c.Sort();
  c.Print();
  Printf("Use binary search to get position of string damon");
  Printf("damon is at position %d", c.BinarySearch(s3));
  Printf("Find str before fons");
  c.Before(&s6)->Print();
  Printf("Find string after damon");
  c.After(s3)->Print();
  Printf("Remove damon and print list again");
  c.Remove(s3);
  delete s3;
  c.Print();
  Printf("Iterate forward over list and remove cassius");
  TObjString *objs;
  TIter next(&c);
  while (objs = (TObjString*)next())     // iterator skips empty slots
     if (objs->String() == "cassius") {
        c.Remove(objs);
        delete objs;
     }
  Printf("Iterate backward over list and remove gaia");
  TIter next1(&c, kIterBackward);
  while (objs = (TObjString*)next1())
     if (objs->String() == "gaia") {
        c.Remove(objs);
        delete objs;
     }
  Printf("Delete remainder of list: anton,bobo,enigma (fons not deleted since not on heap)");
  c.Delete();                           // delete heap objects and clear list
  Printf("Delete stack based objects (fons)");
}
void Test_TList()
{
  Printf(
  "////////////////////////////////////////////////////////////////
"
  "// Test of TList                                              //
"
  "////////////////////////////////////////////////////////////////"
  );
```

// Create a doubly linked list.

```
  TList l;
  Printf("Filling TList");
  TObjNum *n3 = new TObjNum(3);
  l.Add(n3);
  l.AddBefore(n3, new TObjNum(5));
  l.AddAfter(n3, new TObjNum(2));
  l.Add(new TObjNum(1));
  l.AddBefore(n3, new TObjNum(4));
  TObjNum n6(6);                        // stack based TObjNum
  l.AddFirst(&n6);
  Printf("Print list");
  l.Print();
  Printf("Remove 3 and print list again");
  l.Remove(n3);
  delete n3;
  l.Print();
  Printf("Iterate forward over list and remove 4");
  TObjNum *obj;
  TIter next(&l);
  while (obj = (TObjNum*)next())
     if (obj->GetNum() == 4) l.Remove(obj);
  Printf("Iterate backward over list and remove 2");
  TIter next1(&l, kIterBackward);
  while (obj = (TObjNum*)next1())
     if (obj->GetNum() == 2) {
        l.Remove(obj);
        delete obj;
     }
  Printf("Delete remainder of list: 1, 5 (6 not deleted since not on heap)");
  l.Delete();
  Printf("Delete stack based objects (6)");
}
void Test_TSortedList()
{
  Printf(
  "////////////////////////////////////////////////////////////////
"
  "// Test of TSortedList                                        //
"
  "////////////////////////////////////////////////////////////////"
  );
```

// Create a sorted doubly linked list.

```
  TSortedList sl;
  Printf("Filling TSortedList");
  TObjNum *n3 = new TObjNum(3);
  sl.Add(n3);
  sl.AddBefore(n3,new TObjNum(5));
  sl.AddAfter(n3, new TObjNum(2));
  sl.Add(new TObjNum(1));
  sl.AddBefore(n3, new TObjNum(4));
  TObjNum n6(6);                     // stack based TObjNum
  sl.AddFirst(&n6);
  Printf("Print list");
  sl.Print();
  Printf("Delete all heap based objects (6 not deleted since not on heap)");
  sl.Delete();
  Printf("Delete stack based objects (6)");
}
void Test_THashTable()
{
  Printf(
  "///////////////////////////////////////////////////////////////////
"
  "// Test of THashTable                                          //
"
  "///////////////////////////////////////////////////////////////////"
  );
  int i;
```

// Create a hash table with an initial size of 20 (actually the next prime
// above 20). No automatic rehashing.

```
  THashTable ht(20);
  Printf("Filling THashTable");
  Printf("Number of slots before filling: %d", ht.Capacity());
  for (i = 0; i < 1000; i++)
     ht.Add(new TObject);
  Printf("Average collisions: %f", ht.AverageCollisions());
```

// rehash the hash table to reduce the collission rate

```
  ht.Rehash(ht.GetSize());
  Printf("Number of slots after rehash: %d", ht.Capacity());
  Printf("Average collisions after rehash: %f", ht.AverageCollisions());
  ht.Delete();
```

// Create a hash table and trigger automatic rehashing when average
// collision rate becomes larger than 5.

```
  THashTable ht2(20,5);
  Printf("Filling THashTable with automatic rehash when AverageCollisions>5");
  Printf("Number of slots before filling: %d", ht2.Capacity());
  for (i = 0; i < 1000; i++)
     ht2.Add(new TObject);
  Printf("Number of slots after filling: %d", ht2.Capacity());
  Printf("Average collisions: %f", ht2.AverageCollisions());
  Printf("
Delete all heap based objects");
  ht2.Delete();
}
void Test_TBtree()
{
  Printf(
  "///////////////////////////////////////////////////////////////////
"
  "// Test of TBtree                                              //
"
  "///////////////////////////////////////////////////////////////////"
  );
  TStopwatch timer;      // create a timer
  TBtree     l;          // btree of order 3
  Printf("Filling TBtree");
  TObjNum *n3 = new TObjNum(3);
  l.Add(n3);
  l.AddBefore(n3,new TObjNum(5));
  l.AddAfter(n3, new TObjNum(2));
  l.Add(new TObjNum(1));
  l.AddBefore(n3, new TObjNum(4));
  TObjNum n6(6);                     // stack based TObjNum
  l.AddFirst(&n6);
  timer.Start();
  for (int i = 0; i < 50; i++)
```

```
      l.Add(new TObjNum(i));
   timer.Print();
   Printf("Print TBtree");
   l.Print();
   Printf("Remove 3 and print TBtree again");
   l.Remove(n3);
   l.Print();
   Printf("Iterate forward over TBtree and remove 4 from tree");
   TIter next(&l);
   TObjNum *obj;
   while (obj = (TObjNum*)next())
      if (obj->GetNum() == 4) l.Remove(obj);
   Printf("Iterate backward over TBtree and remove 2 from tree");
   TIter next1(&l, kIterBackward);
   while (obj = (TObjNum*)next1())
      if (obj->GetNum() == 2) l.Remove(obj);
   Printf("
Delete all heap based objects");
   l.Delete();
   Printf("Delete stack based objects (6)");
}
int main()
{
   Test_TObjArray();
   Test_TOrdCollection();
   Test_TList();
   Test_TSortedList();
   Test_THashTable();
   Test_TBtree();
   return 0;
}
```