Introduction to ROOT

Summer Students Lecture
6 July 2009

Fons Rademakers (PH/SFT)
Jan Fiete Grosse-Oetringhaus (PH/AIP)

http://root.cern.ch
ROOT in a Nutshell

- ROOT is a large Object-Oriented data handling and analysis framework
  - Efficient object store scaling from KB’s to PB’s
  - C++ interpreter
  - Extensive 2D+3D scientific data visualization capabilities
  - Extensive set of multi-dimensional histograming, data fitting, modeling and analysis methods
  - Complete set of GUI widgets
  - Classes for threading, shared memory, networking, etc.
  - Parallel version of analysis engine runs on clusters and multi-core
  - Fully cross platform, Unix/Linux, MacOS X and Windows

- The user interacts with ROOT via a graphical user interface, the command line or scripts
- The command and scripting language is C++, thanks to the embedded CINT C++ interpreter, and large scripts can be compiled and dynamically loaded
The ROOT Libraries

- Over 1500 classes
- 2,000,000 lines of code
- CORE (8 Mbytes)
- CINT (2 Mbytes)
- Green libraries linked on demand via plug-in manager (only a subset shown)
- 100 shared libs
ROOT: An Open Source Project

- The project was started in Jan 1995
- First release Nov 1995
- The project is developed as a collaboration between:
  - Full time developers:
    - 8 people full time at CERN (PH/SFT)
    - 2 developers at Fermilab/USA
  - Large number of part-time contributors (160 in CREDITS file)
  - A long list of users giving feedback, comments, bug fixes and many small contributions
    - 3870 registered to RootTalk forum
    - 10,000 posts per year
- An Open Source Project, source available under the LGPL license
- Used by all HEP experiments in the world
- Used in many other scientific fields and in commercial world
ROOT Stats

- ROOT binaries have been downloaded more than 650000 times since 1997
- The estimated user base is about 20000 people
ROOT: a Framework and a Library

- User classes
  - User can define new classes interactively
  - Either using calling API or sub-classing API
  - These classes can inherit from ROOT classes

- Dynamic linking
  - Interpreted code can call compiled code
  - Compiled code can call interpreted code
  - Macros can be dynamically compiled & linked

This is the normal operation mode

Interesting feature for GUIs & event displays

Script Compiler
root > .x file.C++
ROOT Application Domains

Data Analysis & Visualization

Data Storage: Local, Network
Three User Interfaces

- **GUI**
  windows, buttons, menus

- **Command line**
  CINT (C++ interpreter)

- **Macros, applications, libraries** (C++ compiler and interpreter)
CINT Interpreter
CINT in ROOT

- CINT is used in ROOT:
  - As command line interpreter
  - As script interpreter
  - To generate class dictionaries
  - To generate function/method calling stubs
  - Signals/Slots with the GUI
- The command line, script and programming language become the same
- Large scripts can be compiled for optimal performance
Compiled versus Interpreted

- Why compile?
  - Faster execution, CINT has some limitations...

- Why interpret?
  - Faster Edit → Run → Check result → Edit cycles ("rapid prototyping"). Scripting is sometimes just easier.

- Are Makefiles dead?
  - No! if you build/compile a very large application
  - Yes! ACLiC is even platform independent!
Running Code

To run function mycode() in file mycode.C:

```
root [0] .x mycode.C
```

Equivalent: load file and run function:

```
root [2] mycode()
```

All of CINT's commands (help):
```
root [3] .h
```
Running Code

Macro: file that is interpreted by CINT (.x)

```c
int mymacro(int value)
{
    int ret = 42;
    ret += value;
    return ret;
}
```

Execute with `.x mymacro.C(42)`
Unnamed Macros

No functions, just statements

```
{  
    float ret = 0.42;  
    return sin(ret);  
}
```

Execute with `.x mymacro.C`

No functions thus no arguments

Named macro recommended!
Compiler prefers it, too...
"Library": compiled code, shared library
CINT can call its functions!
Building a library from a macro: ACLiC
(Automatic Compiler of Libraries for CINT)
.x mymacro.C(42) +
Use "+" instead of writing a Makefile...

CINT knows all functions in mymacro_C.so/.dll
mymacro(42)
My First Session

```
root
root [0] 344+76.8
(const double)4.20800000000000010e+002
root [1] float x=89.7;
root [2] float y=567.8;
root [3] x+sqrt(y)
(double)1.13528550991510710e+002
root [4] float z = x+2*sqrt(y/6);
root [5] z
(float)1.09155929565429690e+002
root [6] .q
```

See file $HOME/.root_hist

```
root [0] try up and down arrows
```

Introduction to ROOT
My Second Session

Introduction to ROOT

root

root [0] .x session2.C
for N=100000, sum= 45908.6
root [1] sum
(double)4.59085828512453370e+004
Root [2] r.Rndm()
(Double)8.29029321670533560e-001
root [3] .q

session2.C

{ int N = 100000;
 TRandom r;
 double sum = 0;
 for (int i=0;i<N;i++) {
   sum += sin(r.Rndm());
 }
 printf("for N=%d, sum= %g\n",N,sum);
}

unnamed macro executes in global scope
My Third Session

```
root [0] .x session3.C
for N=100000, sum= 45908.6
root [1] sum
Error: Symbol sum is not defined in current scope
*** Interpreter error recovered ***
Root [2] .x session3.C(1000)
for N=1000, sum= 460.311
root [3] .q
```

```
void session3 (int N=100000) {
    TRandom r;
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += sin(r.Rndm());
    }
    printf("for N=%d, sum= %g\n",N,sum);
}
```

Named macro
Normal C++ scope rules

Introduction to ROOT
My Third Session with ACLIC

root [0] gROOT->Time();
root [1] .x session4.C(10000000)
for N=10000000, sum= 4.59765e+006
Real time 0:00:06, CP time 6.890
for N=10000000, sum= 4.59765e+006
Real time 0:00:09, CP time 1.062
root [3] session4(10000000)
for N=10000000, sum= 4.59765e+006
Real time 0:00:01, CP time 1.052
root [4] .q

File session4.C
Automatically compiled and linked by the native compiler.
Must be C++ compliant

#include "TRandom.h"
void session4 (int N) {
    TRandom r;
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += sin(r.Rndm());
    }
    printf("for N=%d, sum= %g\n",N,sum);
}
Macros With More Than One Function

root [0] .x session5.C >session5.log
root [1] .q

root [0] .L session5.C
root [1] session5(100); >session5.log
root [2] session5b(3)
sum(0) = 0
sum(1) = 1
sum(2) = 3
root [3] .q

session5.C

```c
void session5(int N=100) {
    session5a(N);
    session5b(N);
    gROOT->ProcessLine(".x session4.C+(1000)" );
}
void session5a(int N) {
    for (int i=0;i<N;i++) {
        printf("sqrt(%d) = %g
",i,sqrt(i));
    }
}
void session5b(int N) {
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += i;
        printf("sum(%d) = %g
",i,sum);
    }
}
```

.x session5.C executes the function session5 in session5.C

use gROOT->ProcessLine to execute a macro from a macro or from compiled code
Generating a Dictionary

```
rootcint -f MyDict.cxx -c MyClass.h
```

MyClass.h

MyDict.h

MyDict.cxx

compile and link

libMyClass.so
Graphics & GUI
TPad: Main Graphics Container

The `Draw` function adds the object to the list of primitives of the current pad.

If no pad exists, a pad is automatically created with a default range \([0,1]\).

When the pad needs to be drawn or redrawn, the object `Paint` function is called.

Only objects deriving from `TObject` may be drawn in a pad.

**ROOT Objects or User objects**
Basic Primitives

- TLine
- TArrow
- TEllipse
- TCurvyLine
- TPaveLabel
- TBox
- TText
- TMarker
- TCrown
- TCurlyArc
- TLatex
- $\alpha^2 + \beta^2$
- Test in a pad
- Hello CERN
- TPave
- TPaveText
- TPolyLine
Formula or diagrams can be edited with the mouse.
Graphs

- \text{TGraph}(n,x,y)
- \text{TCutG}(n,x,y)
- \text{TMultiGraph}

\text{TGraphErrors}(n,x,y,ex,ey)

\text{TGraphAsymmErrors}(n,x,y,exl,exh,eyl,eyh)

\text{gerrors2.C}
Introduction to ROOT

Graphics Examples

Concentration of elements derived from mixture Ca53+Sr78

\[ C_x = \frac{N_x(t)}{N_0(t=0)} = \sum a_i e^{-\lambda_i t} \]

Concentration of C14 derived elements

\[ C_x = \frac{N_x(t)}{N_0(t=0)} = \sum a_i e^{-\lambda_i t} \]

Time evolution of a population of radionuclides. The concentration of a nuclide X represents the ratio between the number of X nuclei and the number of nuclei of the top element of the decay from which X derives from at T=0.
More Graphics Examples
Even More Graphics Examples

- **Di-electron Invariant Mass Spectrum**
  - CDF Run II Preliminary
  - Integration: \( \int L \, dt = 1.3 \, \text{fb}^{-1} \)

- **Di-jet mass**
  - DØ Run II Preliminary
  - Signal window
  - Events with \( \geq 3 \) b-tagged jets

- **Signal window**
  - Data
  - Total bkgd.
  - \( m_A = 120 \, \text{GeV} \)
  - \( 260 \, \text{pb}^{-1} \)
Special Graphics

Graph polar drawn as a polygon

Error bars

Pie with offset and no colors

Pie with radial labels

Pie with tangential labels

Pie with verbose labels

Introduction to ROOT
ASImage: Image processor
GUI (Graphical User Interface)
Canvas tool bar/menus/help
Object Editor

Click on any object to show its editor
ROOT Browser

```c
// Example for fitting signal/background. This example can be executed with
// root > .x FittingDemo.C (using the CINT interpreter)
// root > .x FittingDemo.C+ (using the native compiler via ACLIC)
// Author: Rene Brun

#include "TH1.h"
#include "TMath.h"
#include "TF1.h"
#include "TLegend.h"
#include "TCanvas.h"

// Quadratic background function
Double_t background(Double_t *x, Double_t *par) {
  return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

// Lorentzian Peak function
Double_t lorentzianPeak(Double_t *x, Double_t *par) {
  return 0.5*par[0]*par[1]*TMath::P(0) /
         TMath::Max( 1.e-10,(x[0]-par[2])*(x[0]-par[2]) + .25*par[1]*par[1] );
}

// Sum of background and peak function
Double_t fitFunction(Double_t *x, Double_t *par) {
  return background(x, par) + lorentzianPeak(x, &par[3]);
}

void FittingDemo()
{
  // Bevington Exercise by Peter Malzacher, modified by Rene Brun
  const int nBins = 60;
  Double_t data[nBins] = { 6, 1.10,12, 6.13,23,22,15,21, 23, 26,36,25,27,35,40,44,66,81, 75, 57,43,45,46,41,35,36,53,32
                           ... 
  http://root.cern.ch/root/Benchmark.html
```
When pressing **ctrl+S** on any widget it is **saved as a C++ macro file** thanks to the `SavePrimitive` methods implemented in all GUI classes. The generated macro can be edited and then executed via CINT.

Executing the macro restores the complete original GUI as well as all created signal/slot connections in a global way.

```c++
// transient frame
TGRadioButton *frame4 = new TGRadioButton(frame3, "gaus", 10);
frame3->AddFrame(frame4);
```

`root [0] .x example.C`
The GUI Builder

- The GUI builder provides GUI tools for developing user interfaces based on the ROOT GUI classes. It includes over 30 advanced widgets and an automatic C++ code generator.
More GUI Examples

$ROOTSYS/tutorials/gui
$ROOTSYS/test/RootShower
$ROOTSYS/test/RootIDE
The **GEOMetry** package is used to model very complex detectors (LHC). It includes

- a visualization system
- a navigator (where am I, distances, overlaps, etc)
OpenGL

see $ROOTSYS/tutorials/geom
Math Libraries

**MathMore**
- Random Numbers
- Extra algorithms
- Extra Math functions
- GSL and more

**MathCore**
- Function interfaces
- Physics Vectors
- Basic algorithms
- Basic Math functions

**Histogram library**
- TH1
- TF1

**Statistical Libraries**
- Statistical Utilities
- TMVA
- MLP

**Fitting and Minimization**
- New Fitter
- Tminuit
- TFumili
- Linear Fitter
- Minuit2 (new C++ Minuit)

**Linear Algebra**
- TMatrix
- SMatrix

**libCore**
- TMath
- TRandom

Introduction to ROOT
Peak Finder + Deconvolutions

T Spectrum
Fitters

- Minuit
- Fumili
- LinearFitter
- RobustFitter
- RooFit

Lorentzian Peak on Quadratic Background

Data, Background fit, Signal fit, Global Fit
Fit Panel
RooFit: a Powerful Fitting Framework

see $ROOTSYS/tutorials/roofit/RoofitDemo.C
I/O

Object in Memory

Streamer: No need for transient / persistent classes

Buffer

sockets -> Net File
http -> Web File
XML -> XML File
SQL -> RDBMS

Local

File on disk

Introduction to ROOT
Object Oriented Concepts

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

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

![Class Hierarchy Diagram]

TObject

- IsA

Event

- HasA

Jets

- HasA

Momentum

- HasA

Segments

Tracks

- HasA

EvNum

- HasA

Charge
A TFile object may be divided in a hierarchy of directories, like a Unix file system.

Two I/O modes are supported

- **Key-mode (TKey)**. An object is identified by a name (key), like files in a Unix directory. OK to support up to a few thousand objects, like histograms, geometries, mag fields, etc.

- **TTree-mode** to store event data, when the number of events may be millions, billions.
Self-describing Files

- Dictionary for persistent classes written to the file
- ROOT files are self describing
- Support for Backward and Forward compatibility
- Files created in 2001 must be readable in 2015
- Classes (data objects) for all objects in a file can be regenerated via TFile::MakeProject

```
Root > TFile f("demo.root");
Root > f.MakeProject("dir", "*", "new++");
```
Example of Key Mode

void keywrite() {
    TFile f("keymode.root","new");
    TH1F h("hist","test",100,-3,3);
    h.FillRandom("gaus",1000);
    h.Write()
}

void keyRead() {
    TFile f("keymode.root");
    TH1F *h = (TH1F*)f.Get("hist");;
    h.Draw();
}
A Root file `pippa.root` with two levels of directories.

Objects in directory `/pippa/DM/CJ` e.g.: `/pippa/DM/CJ/h15`
LHC: How Much Data?

High Level-1 Trigger
(1 MHz)

High No. Channels
High Bandwidth
(500 Gbit/s)

1 billion people
surfing the Web

10^1 Rate
(Hz)

10^4
10^5
10^6

LHC

KLOE

HERA-B

CDF II

CDF

H1

ZEUS

UA1

LEP

ATLAS

CMS

High Data Archive
(5 PetaBytes/year)

10 Gbits/s in Data base

Event Size (bytes)

ALICE

ATLAS

CMS

10^2
10^3
10^4
10^5
10^6
10^7

LHCB

10^4
10^5
10^6
ROOT Trees
Why Trees?

- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the best).
- Trees have branches and leaves. One can read a subset of all branches.
- High level functions like `TTree::Draw` loop on all entries with selection expressions.
- Trees can be browsed via `TBrowser`.
- Trees can be analyzed via `TTreeViewer`. 
Memory $\leftrightarrow$ Tree
Each Node is a Branch in the Tree

T.Fill()
ROOT I/O -- *Split/Cluster*

**Tree version**

- **Tree entries**
- **Streamer**
- **Branches**

**Tree in memory**

**File**
Writing/Reading a Tree

class Event : public Something {
    Header fHeader;
    std::list<Vertex*> fVertices;
    std::vector<Track> fTracks;
    TOF fTOF;
    Calor *fCalor;
}

main() {
    Event *event = 0;
    TFile f("demo.root", "recreate");
    int split = 99;  //maximum split
    TTree *T = new TTree("T","demo Tree");
    T->Branch("event", "Event", &event, split);
    for (int ev = 0; ev < 1000; ev++) {
        event = new Event(...);
        T->Fill();
        delete event;
    }
    t->AutoSave();
}

main() {
    Event *event = 0;
    TFile f("demo.root");
    TTree *T = (TTree*)f.Get("T");
    T->SetBranchAddress("event", &event);
    Long64_t N = T->GetEntries();
    for (Long64_t ev = 0; ev < N; ev++) {
        T->GetEntry(ev);
        // do something with event
    }
}
Browsing a Tree

8 Branches of T

8 leaves of branch Electrons

A double-click to histogram the leaf
The TTreeViewer
TTree Selection Syntax

Print the first 8 variables of the tree.

```
MyTree->Scan();
```

Print all the variables of the tree.

```
MyTree->Scan("*");
```

Print the values of var1, var2 and var3.

```
MyTree->Scan("var1:var2:var3");
```

Print the values of var1, var2 and var3 for the entries where var1 is exactly 0.

```
MyTree->Scan("var1:var2:var3", "var1==0");
```
Data Volume & Organisation

A TFile typically contains 1 TTree

A TChain is a collection of TTrees or/and TChains

A TChain is typically the result of a query to the file catalogue
Chains of Trees

- A TChain is a collection of Trees.
- Same semantics for TChains and TTrees
  - `root > .x h1chain.C`
  - `root > chain.Process("h1analysis.C")`

```cpp
{
    //creates a TChain to be used by the h1analysis.C class
    //the symbol H1 must point to a directory where the H1 data sets
    //have been installed

    TChain chain("h42");
    chain.Add("$H1/dstarmb.root");
    chain.Add("$H1/dstarpla.root");
    chain.Add("$H1/dstarp1b.root");
    chain.Add("$H1/dstarp2.root");
}
```
Tree Friends

Entry # 8

Public read

Public read

User Write
GRIDs & Multi-Cores & PROOF
From the desktop to the GRID

Desktop
Online/Offline
Farms
Local/remote
Storage

New data analysis tools must be able to use in parallel remote CPUs, storage elements and networks in a transparent way for a user at a desktop.
GRID: Interactive Analysis
Case 1

- Data transfer to user’s laptop
- Optional Run/File catalog
- Optional GRID software

Analysis scripts are interpreted or compiled on the local machine.

Remote file server
e.g. xrootd

Optional run/File Catalog

Trees

Trees
GRID: Interactive Analysis Case 2

- Remote data processing
- Optional Run/File catalog
- Optional GRID software

Analysis scripts are interpreted or compiled on the remote master(s)

Commands, scripts

Remote data analyzer e.g. PROOF

Histograms, trees
Parallel ROOT Facility

- A system for running ROOT queries in parallel on a large number of distributed computers or many-core machines
- PROOF is designed to be a transparent, scalable and adaptable extension of the local interactive ROOT analysis session
- Extends the interactive model to long running “interactive batch” queries
- Uses xrootd for data access and communication infrastructure
- For optimal CPU load it needs fast data access (SSD, disk, network) as queries are often I/O bound
- Can also be used for pure CPU bound tasks like toy Monte Carlo’s for systematic studies or complex fits
Cluster perceived as extension of local PC
- *Same macro and syntax* as in local session
- More *dynamic* use of resources
- Real-time feedback
- Automated *splitting* and *merging*
In this talk, I presented the most basic classes typically used during Physics Analysis.

ROOT contains many more libraries, e.g.:
- FFT library
- Oracle, MySQL, etc interfaces
- XML drivers
- TMVA (Multi Variate Analysis)
- GRID, networking and thread classes
- Interfaces to Castor, Dcache, GFAL, xrootd
- Interfaces to Pythia, Geant3, Geant4, gdml
- Matrix packages, Fitting packages, etc
Users Guide and Reference Manuals are available at http://root.cern.ch

Tomorrow Jan Fiete will demo in his session many of the features I’ve presented today