Introduction to ROOT

Summer Students Lecture

21 July 2004

René.Brun@cern.ch

ROOT in a nutshell

- An efficient data storage and access system designed to support structured data sets in very large distributed data bases (Petabytes).
- A query system to extract information from these distributed data sets.
- The query system is able to use transparently parallel systems on the GRID (PROOF).
- A scientific visualisation system with 2-D and 3-D graphics.
- An advanced Graphical User Interface
- A C++ interpreter allowing calls to user defined classes.
- An Open Source Project
ROOT: An Open Source Project

- The project is developed as a collaboration between:

  - Full time developers:
    - 6 people full time at CERN
    - 1 key developer at FermiLab
    - 1 key developer in Japan (Agilent Technologies)
    - 1 key developer at MIT
    - 1 mathematician at CERN sponsored by a US Finance Company

  - Many contributors spending a substantial fraction of their time in specific areas (> 50).

  - Key developers in large experiments using ROOT as a framework.

  - Several thousand users given feedback and a very long list of small contributions.
The ROOT web pages

http://root.cern.ch

- General Information and News
- Download source and binaries
- Howto & tutorials
- User Guide & Reference Guides
- Roottalk Digest & Forum
Batch/Interactive models

Need experiment framework +widely available tools
 Batch
 Production
 Simulation
 reconstruction

Interactive batch model

Interactive Chaotic analysis

Need only widely available tools

Need only widely available tools
Data Volume & Processing Time
Using technology available in 2004

100MB 1GB 10GB 100GB 1TB 10TB 100TB 1PB

ROOT 1 Processor  P IV 2.4GHz  2004 : Time for one query using 10 per cent of data

Interactive

1” 10” 1’ 10’ 1h 10h 1day 1month

batch

PROOF 10 Processors

1” 1” 10” 1’ 10’ 1h 10h 1day 10days

PROOF 100 Processors

1” 1” 1” 10” 1’ 10’ 1h 10h 1day

PROOF/GLite 1000 Processors

1’ 10’ 1h 10h
Data Volume & Processing Time
Using technology available in 2010

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Interactive</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>100MB</td>
<td>1”</td>
<td>1h</td>
</tr>
<tr>
<td>1GB</td>
<td>1”</td>
<td>10h</td>
</tr>
<tr>
<td>10GB</td>
<td>10”</td>
<td>1day</td>
</tr>
<tr>
<td>100GB</td>
<td>1’</td>
<td>10days</td>
</tr>
<tr>
<td>1TB</td>
<td>10’</td>
<td></td>
</tr>
<tr>
<td>10TB</td>
<td>1h</td>
<td></td>
</tr>
<tr>
<td>100TB</td>
<td>10h</td>
<td></td>
</tr>
<tr>
<td>1PB</td>
<td>1day</td>
<td></td>
</tr>
</tbody>
</table>

ROOT 1 Processor XXXXX 2010: Time for one query using 10 per cent of data
1” 1” 10” 1’ 10’ 1h 10h 1day 10days

PROOF 10 Processors
1” 1” 1” 10” 1’ 10’ 1h 10h 1day

PROOF 100 Processors
1” 1” 1” 1” 10” 1’ 10’ 1h 10h

PROOF/GLite 1000 Processors
1’ 10’ 1h
ROOT Library Structure

- ROOT libraries are a layered structure
- The CORE classes are always required (support for RTTI, basic I/O and interpreter)
- The optional libraries (you load only what you use)
  Separation between data objects and the high level classes acting on these objects. Example, a batch job uses only the histogram library, no need to link histogram painter library.
- Shared libraries reduce the application link time
- Shared libraries reduce the application size
- ROOT shared libraries can be used with other class libraries
The Libraries

- Over 1000 classes
- 1250,000 lines of code
- CORE (12 Mbytes)
- CINT (3 Mbytes)
- Green libraries linked on demand via plug-in manager (only a subset shown)
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++
A Data Analysis & Visualisation tool
Introduction to ROOT

Graphics: 1, 2, 3-D functions
Full LateX support on screen and postscript

Formula or diagrams can be edited with the mouse

\[ \frac{2s}{\pi \alpha^2} \frac{d\sigma}{d\cos \theta} (e^+ e^- \rightarrow f \bar{f}) = \left| \frac{1}{1 - \Delta \alpha} \right|^2 (1 + \cos^2 \theta) \]

\[ + 4 \text{ Re} \left\{ \frac{2}{1 - \Delta \alpha} \chi(s) \left[ g_V g_V (1 + \cos^2 \theta) + 2 g_a g_a \cos \theta \right] \right\} \]

\[ + 16 \chi(s)^2 \left[ (g_a + g_V) (g_a + g_V) (1 + \cos^2 \theta) + 8 g_a g_a g_V g_V \cos \theta \right] \]

TCurlyArc
TCurlyLine
TWavyLine
and other building blocks for Feynmann diagrams
Introduction to ROOT

Alice

3 million nodes
ROOT + RDBMS Model

- ROOT files
- Oracle
- MySQL
- Event Store
  - histograms
  - Trees
  - Geometries
- Run/File Catalog
  - Calibrations
ROOT I/O : An Example

Program Writing

tfile f("example.root","new");
TH1F h("h","My histogram",100,-3,3);
h.FillRandom("gaus",5000);
h.Write();

Program Reading

tfile f("example.root");
TH1F *h = (TH1F*)f.Get("h");
h->Draw();
f.Map();

```
INTRODUCTION TO ROOT 17

ROOT I/O : An Example

TFile f("example.root","new");
TH1F h("h","My histogram",100,-3,3);
h.FillRandom("gaus",5000);
h.Write();

TFile f("example.root");
TH1F *h = (TH1F*)f.Get("h");
h->Draw();
f.Map();
```

20010831/171903 At: 64 N= 90 TFile
20010831/171941 At: 154 N= 453 TH1F CX =  2.09
20010831/171946 At: 607 N=2364 StreamerInfo CX =  3.25
20010831/171946 At: 2971 N=  96 KeysList
20010831/171946 At: 3067 N=  56 FreeSegments
20010831/171946 At: 3123 N=  1 END
Introduction to ROOT 18

A Root file `pippa.root` with two levels of directories:

Objects in directory `/pippa/DM/CJ`
- eg: `/pippa/DM/CJ/h15`

A Root file `pippa.root` with two levels of directories.
Memory <--> Tree
Each Node is a branch in the Tree
void demoe(int nevents) {
    //load shared lib with the Event class
    gSystem->Load("$ROOTSYS/test/libEvent");

    //create a new ROOT file
    TFile f("demoe.root","new");

    //Create a ROOT Tree with one single top level branch
    int split = 99; //try also split=1 and split=0
    int bufsize = 16000;
    Event *event = new Event;
    TTree T("T","Event demo tree");
    T.Branch("event","Event",&event,bufsize,split);

    //Build Event in a loop and fill the Tree
    for (int i=0;i<nevents;i++) {
        event->Build(i);
        T.Fill();
    }

    T.Print(); //Print Tree statistics
    T.Write(); //Write Tree header to the file
}
```cpp
void demoer() {
    //load shared lib with the Event class
    gSystem->Load("$ROOTSYS/test/libEvent");

    //connect ROOT file
    TFile *f = new TFile("demoe.root");

    //Read Tree header and set top branch address
    Event *event = 0;
    TTree *T = (TTree*)f->Get("T");
    T->SetBranchAddress("event", &event);

    //Loop on events and fill an histogram
    TH1F *h = new TH1F("hntrack","Number of tracks",100,580,620);
    int nevents = (int)T->GetEntries();
    for (int i=0;i<nevents;i++) {
        T->GetEntry(i);
        h->Fill(event->GetNtrack());
    }
    h->Draw();
}
```

Rebuild the full event in memory
Introduction to ROOT

8 Branches of T

8 leaves of branch Electrons

A double-click to histogram the leaf
The Tree Viewer & Analyzer

A very powerful class supporting complex cuts, event lists, 1-d, 2-d, 3-d views parallelism
Tree Friends

Entry # 8

Public read

Public read

User Write
Tree Friends

Processing time independent of the number of friends unlike table joins in RDBMS

Root > TFile f1("tree1.root");
Root > tree.AddFriend("tree2","tree2.root")
Root > tree.AddFriend("tree3","tree3.root");
Root > tree.Draw("x:a","k<c");
Root > tree.Draw("x:tree2.x","sqrt(p)<b");
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
GRID: Interactive Analysis
Case 2

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

Analysis scripts are interpreted or compiled on the remote machine

Optional run/File Catalog

Trees

Remote data analyzer eg proofd

Commands, scripts

histograms
GRID: Interactive Analysis
Case 3

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

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

Remote data analyzer eg proofd

Commands, scripts

Histograms, trees

Run/File Catalog

Trees

slaves

Trees

slaves

slaves

slaves

slaves
More and more GRID oriented data analysis
More and more experiment-independent software
Google: a good model
Make it simple

Web Images Groupe Annuaire Actualités
Rechercher
Recherche avancée
Reset
Web Images Groupe Annuaire Actualités
Rechercher
Recherche avancée
Reset

---

Introduction to ROOT

Google: a good model
Make it simple

Simple interface
Available everywhere
Hidden Parallelism
Distributed DB
Don’t know a priori
the data location
Fast

RTAG11 ARDA Documents

RTAG11 ARDA Documents

RTAG11 ARDA Documents
Playing with ROOT on lxplus

- Set the environment variables
  - `setenv ROOTSYS /afs/cern.ch/sw/root/v4.00.08/rh73_gcc32/root`
  - `setenv LD_LIBRARY_PATH $ROOTSYS/lib:$LD_LIBRARY_PATH`
  - `setenv PATH $ROOTSYS/bin:$PATH`

- Copy the ROOT tutorials
  - `cd $HOME`
  - `cp $ROOTSYS/tutorials .`
  - `cd tutorials`

- Run ROOT
  - `root`
Playing with ROOT on a PC

- Import a binary tar file from:
  - Eg: Intel x86 Linux for Redhat 7.3 and gcc 3.2, version 4.00/08
- Untar the file in your home directory
  - cd $HOME
  - tar zxvf root_v4.00.08.Linux.RH7.3.gcc32.tar.gz
- Set the environment variables
  - setenv ROOTSYS $HOME/root
  - setenv LD_LIBRARY_PATH $ROOTSYS/lib:$LD_LIBRARY_PATH
  - setenv PATH $ROOTSYS/bin:$PATH
- Go to the ROOT tutorials
  - cd $HOME
  - cp $ROOTSYS/tutorials .
  - cd tutorials
- Run ROOT
  - root
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

root

See file $HOME/.root_hist

root [0] try up and down arrows
My second session

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

unnamed macro executes in global scope

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);
}
My third session

root

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

Named macro
Normal C++ scope rules

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);
}

session3.C
My third session with ACLI C

```c
#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);
}
```

File session4.C
Automatically compiled and linked by the native compiler.
Must be C++ compliant
Macros with more than one function

```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\n", i, sqrt(i));
    }
}

void session5b(int N) {
    double sum = 0;
    for (int i=0; i<N; i++) {
        sum += i;
        printf("sum(%d) = %g\n", i, sum);
    }
}
```

.session5.C

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

.use gROOT->ProcessLine to execute a macro from a macro or from compiled code
Interactive Demo

You can import the tar file with all demos from 

root summershow.C

You can find more demos, examples, tests at

$ROOTSYS/ tutorials
$ROOTSYS/ test
$ROOTSYS/ test/ RootShower