CSU NUPAC Tutorials
2019

Gabriel Palacino

I’m borrowing a lot of the material from Harinder Bawa (CSU Fresno)
Why Should You Use a Tree?

• We saw how objects can be saved in ROOT files. In case you want to store large quantities of same-class objects, ROOT has designed the TTree and TNtuple classes specifically for that purpose.

• The TTree class is optimized to reduce disk space and enhance access speed. A TNtuple is a TTree that is limited to only hold floating-point numbers; a TTree on the other hand can hold all kind of data, such as objects or arrays in addition to all the simple types.
What is a Tree?

A Tree is like a large and wide table:

- A Tree is an array of ‘entries’ or ‘events’, similar to a row of a table.
- Within each entry, there are independent ‘branches’. Each branch can contain an object or sub-branches. This can be compared to a column of a table.
- Within each branch, there are ‘leaves’, which hold the member-variables of complicated classes. These are the final values.
Using a tree

Graphically (GUI) & Command Line
For Practice.

http://zimmer.csufresno.edu/~hbawa/tree_struc.root
The graphical interface

a->StartViewer()

Drawing options

Drag and drop leaves here to draw the histograms

The tree variables (leaves)

Cuts

Expression boxes

Drawing button

Commands history
Plotting a 1D histogram

1. Choose the variable
2. Click here
3. That's it!
Plotting a 2D histogram

1. Choose the variables
2. Set the drawing option
3. Click here
4. That's it! (SetLogz !!)
Using cuts

1. Double-click on an empty expression box E().

2. Type the cut condition and the name of its alias.

3. Drag and drop the expression box in the cut box.
Using cuts (2)

- Drag and drop the `Th_part` variable on the x axis
- Drag and drop the cut in the "scissors box"
- Double-click on the "scissors box" to disable the cut selection (red line)
- Draw the histogram **without the cut selection**
- Enable the cut selection
- Type "**same**" in the drawing option field
- Draw the histogram **with the cut selection**
- Record the display
- Perfect the presentation of the figure!
Save it...

1. Right-click here

2. Choose this item

3. Type the file name

MaSessionC
Everything is not lost...

1. Click here
2. Choose this item
3. Choose the file
The contents of the tree can be drawn from the command-line:

```
Draw(const char *exp, const char *cut, Option_t *option, Long64_t nent, Long64_t first)
```

- **exp**: expression describing what to draw e.g. "y:x", or "sqrt(x/y*z*z)". For 2-D(or 3-D) plots, expressions are separated by a ":". Convention: z:y:x. Statement "x>>histoname" will save to predefined histogram.
- **cut**: expression describing some conditions, e.g. "z>0"
- **option**: drawing option (see histograms)

Example:
```
ROOT[0] TTree* mytree=new TTree("mytree", "Test Tree");
ROOT[1] Double_t values[3];
ROOT[4] mytree->Draw("sqrt(z):x*y", "z>0","SURF4",1000,10);
```
Tree on the Command Line (Cont’d)

• The structure of the Tree can be printed:

```
mytree->Print();
```

```
*************** Tree: TestTree
*Entries: 2 : Total = 1281 bytes File Size = 0
* : : Tree compression factor = 1.00
***************
*Br 0 : b2 : a/D:b:c:d:e
*Entries: 2 : Total Size= 1001 bytes One basket in memory
*Baskets: 0 : Basket Size= 32000 bytes Compression= 1.00
```

• You can get a list of (part of) the contents:

```c
mytree->Scan("a:b");
```
• You can also inspect the contents of a specific entry numerically:

```c
mytree->Show(1);
```

```
=====> EVENT: 1
  a = 1
  b = 0
  c = 0
  d = -3.14
  e = 0
```
Accessing the tree data

- Selecting the events and print variables values:

\[ a \rightarrow \text{Scan("Mult:Z[30]:Energie[30]","Mult>30","",1000,0)} \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>46</td>
<td>32</td>
<td>2</td>
<td>47.778400</td>
</tr>
<tr>
<td>*</td>
<td>95</td>
<td>31</td>
<td>2</td>
<td>48.006801</td>
</tr>
<tr>
<td>*</td>
<td>399</td>
<td>31</td>
<td>1</td>
<td>28.520700</td>
</tr>
<tr>
<td>*</td>
<td>461</td>
<td>31</td>
<td>2</td>
<td>67.939399</td>
</tr>
<tr>
<td>*</td>
<td>628</td>
<td>32</td>
<td>2</td>
<td>69.046302</td>
</tr>
</tbody>
</table>

===> 5 selected entries
### Various commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-&gt;Print();</td>
<td>Prints the content of the tree</td>
</tr>
<tr>
<td>T-&gt;Scan();</td>
<td>Scans the rows and columns</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;x&quot;);</td>
<td>Draw a branch of tree</td>
</tr>
<tr>
<td>How to apply cuts:</td>
<td></td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;x&quot;,&quot;x&gt;0&quot;);</td>
<td>Draw “x” when “x&gt;0”</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;x&quot;,&quot;x&gt;0 &amp;&amp; y&gt;0&quot;);</td>
<td>Draw “x” when both x &gt;0 and y &gt;0</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;y&quot;,&quot;&quot;;&quot;;&quot;;&quot;same&quot;);</td>
<td>Superimpose “y” on “x”</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;y:x&quot;);</td>
<td>Make “y vs x” 2d scatter plot</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;z:y:x&quot;);</td>
<td>Make “z:y:x” 3d plot</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;sqrt(x<em>x+y</em>y)&quot; );</td>
<td>Plot calculated quantity</td>
</tr>
<tr>
<td>T-&gt;Draw(&quot;x&gt;&gt;h1&quot;);</td>
<td>Dump a root branch to a histogram</td>
</tr>
</tbody>
</table>
Example: Command Line Analyses

```c
#include "TRandom.h"
#include "TFile.h"
#include "TTree.h"

void ExampleTree(const char * filename= "tree.root") {
    TTree data("tree","Example TTree");
    double x, y, z, t;
    data.Branch("x",&x,"x/D");
    data.Branch("y",&y,"y/D");
    data.Branch("z",&z,"z/D");
    data.Branch("t",&t,"t/D");

    // fill it with random data
    for (int i = 0; i<10000; ++i) {
        x = gRandom->Uniform(-10,10);
        y = gRandom->Gaus(0,5);
        z = gRandom->Exp(10);
        t = gRandom->Landau(0,2);
        data.Fill();
    }

    // write in a file
    TFile f(filename,"RECREATE");
    data.Write();
    f.Close();
}
```

Scanning and Browsing a Tree

```
[harinder@harinder-OptiPlex-755 Exercises]$ root -l
root [0] TFile f("tree.root")
root [1] f.ls()
TFile** tree = f.tree.root.
TFile* root [1] tree:1 Example TTree
KEY: TTree tree:1 Example TTree
root [2] tree->Scan()

Row | x | y | z | t
--- |---|---|---|---
 0  | 9.9948349| -2.173821| 0.5424387| -0.612444
 1  | -0.306527| 4.1213184| 2.9530392| 3.3590962
 2  | 4.7990596| 0.0395619| 4.1758331| 0.3248392
 3  | 6.0880662| -4.925330| 17.803898| 2.3403682
 4  | -2.153720| -2.868095| 34.954463| 0.5382660
 5  | -6.117022| 3.8369828| 5.4484498| 22.924656
 6  | 3.112728| 10.364519| 12.156219| -1.950261
 7  | -8.741646| -2.271012| 3.2180651| 5.3128856
 8  | 4.2777012| -9.2215651| 3.5772235| -2.977455
 9  | -7.415144| 11.302227| 24.868109| -1.812724
10 | 0.9474282| 7.6234604| 12.305172| 21.559762
11 | -5.457643| -6.487339| 9.153188| 6.6488874
12 | -6.586581| -5.453365| 19.603690| 8.6652320
13 | -6.351783| -12.29079| 18.275389| 2.1263476
14 | 2.5567294| -1.661886| 3.5925466| 2.3901384
15 | -8.197848| -5.952137| 8.2887077| 2.3339561
16 | -1.615081| 0.4038978| 4.7962692| 169.15314
17 | 3.501395| 1.593703| 19.603690| 8.6652320
18 | 6.5947936| 1.2195639| 10.592969| -3.074369
19 | 2.2613264| -3.939634| 6.9536157| 7.8710863
20 | -7.168989| -6.082385| 11.868474| -1.895370
21 | -5.150696| -8.598714| 5.116594| 5.4938359
22 | -1.347475| -4.175319| 3.3296098| 5.4394669
23 | 3.7155345| 8.9958217| 20.961120| -2.862777
24 | 6.4014408| -1.842549| 17.116538| -1.637794
```

Type <CR> to continue or q to quit => q

(Long64_t)25
```
tv__tree->Draw("y","\"","\"", 10000, 0);

tv__tree->Draw("y","y>0","\"", 10000, 0);

y

<table>
<thead>
<tr>
<th>Entries</th>
<th>Mean</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>-0.03648</td>
<td>4.081</td>
</tr>
</tbody>
</table>

y (y>0)

<table>
<thead>
<tr>
<th>Entries</th>
<th>Mean</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4916</td>
<td>3.154</td>
<td>3.015</td>
</tr>
</tbody>
</table>

Tv__tree->Draw("y","x","\"","colz", 10000, 0);

tv__tree->Draw("y","x","x>2\&y<3","LEGO", 10000, 0);

y:x {x>2\&y<3}
Doing an Analysis

• The TBrowser is not of much use in a typical analysis
• Usually we want to
  - Throw away events we are not interested in
  - Select only certain objects within an event we are interested in
  - Calculate and plot quantities of interest in an event (i.e. energy ratios)
  - Calculate and plot properties of objects in an event (electron pT)
Doing an Analysis

- To do this we need to Loop over each event (entry) in the TTree
- For each event we need to
  - Read all of the variables for the event and store them in memory
  - Perform our selection on these variables
  - Calculate and plot the quantities of interest
- This can be simple if you only care about a few variables
- But most analyses will need >100 of variables
- ROOT has a built in method for looping over a TTree or TChain called a MakeClass/MakeSelector
MakeClass Option
Example: Creating tree

```cpp
#include <iostream>
#include "TTree.h"
#include "TMath.h"
#include "TFile.h"
#include "TRandom3.h"

int run(){
    TRandom3 random;

    //First open a file
    TFile *f1=new TFile("kinematics3.root","RECREATE");
    //Create a TTree object
    TTree* tree=new TTree("tree","An example tree");
    float px,py,pz,E,m;
    //Create branches
    tree->Branch("pxval",&px,"px/F");
    tree->Branch("pyval",&py,"py/F");
    tree->Branch("pzval",&pz,"pz/F");
    tree->Branch("Eval",&E,"E/F");
    tree->Branch("mval",&m,"m/F");
    for(int i=0; i<10000; i++){
        px=random.Gaus(5,1);
        py=random.Poisson(6);
        pz=random.Exp(3);
        E=random.Exp(9);
        m=random.Gaus(10,1);
        tree->Fill();
    }
    f1->Write();
    f1->Close();
    return 0;
}
```
Output files

• [http://zimmer.csufresno.edu/~hbawa/kinematics1.root](http://zimmer.csufresno.edu/~hbawa/kinematics1.root)
• [http://zimmer.csufresno.edu/~hbawa/kinematics2.root](http://zimmer.csufresno.edu/~hbawa/kinematics2.root)
• [http://zimmer.csufresno.edu/~hbawa/kinematics3.root](http://zimmer.csufresno.edu/~hbawa/kinematics3.root)
Analysing tree: MakeClass

[bawa@t3head 2017]$ ls
kinematics1.root  kinematics3.root  run_C_ACLiC_dict_rdict.pcm  run_C.so
kinematics2.root  run.C             run_C.d

[bawa@t3head 2017]$ root -l -b
root [0] TFile f("kinematics1.root")
(TFile &) @0x7f1b87391028
root [1] f.ls()
TFile** kinematics1.root
TFile*  kinematics1.root
KEY: TTree tree;1 An example tree
root [2] tree->MakeClass("Analyze")
Info in <TTreePlayer::MakeClass>: Files: Analyze.h and Analyze.C generated from TTree: tree
(Int_t) 0
root [3] .q
[bawa@t3head 2017]$
```c
#define Analyze_oxx
#include "Analyze.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void Analyze::Loop()
{
    // In a Root session, you can do:
    // Root > .L Analyze.C
    // Root > Analyze t
    // Root > t.GetEntry(12); // Fill t data members with entry number 12
    // Root > t.Show(); // Show values of entry 12
    // Root > t.Show(16); // Read and show values of entry 16
    // Root > t.Loop(); // Loop on all entries

    // This is the loop skeleton
    // To read only selected branches, insert statements like:

    // METHOD1:
    // fChain->SetBranchStatus("*",0); // disable all branches
    // fChain->SetBranchStatus("branchname",1); // activate branchname

    // METHOD2: replace line
    // fChain->GetEntry(i); // read all branches
    // by b branchname->GetEntry(i); // read only this branch
    if (fChain == 0) return;

    Long64_t nentries = fChain->GetEntries();
    Long64_t nbytes = 0, nb = 0;
    for (Long64_t jentry=0; jentry<nentries;jentry++) {
        Long64_t ientry = fChain->LoadTree(jentry);
        nb = fChain->GetEntry(jentry); nbytes += nb;
        // if (Cut(ientry) < 0) continue;
    }

    // Setup code goes here
    // Loop code goes here
}
```
Analyze.h

Here is the input rootfile
void Analyze::Loop()
{
    // In a ROOT session, you can do:
    // root> .L Analyze.C
    root> Analyze t
    // root> t.GetEntry(12); // Fill t data members with entry number 12
    // root> t.Show();    // Show values of entry 12
    // root> t.Show(16); // Read and show values of entry 16
    // root> t.Loop(); // Loop on all entries
    if (fChain == 0) return;
    Long64_t nentries = fChain->GetEntriesFast();
    cout<<"Number of Entries\t"<<nentries<<endl;
    TH1D *hist_px=new TH1D("hist_px","px distribution",100,0,100);
    Long64_t nbytes = 0, nb = 0;
    for (Long64_t jentry=0; jentry<nentries; jentry++) {
        Long64_t ientry = LoadTree(jentry);
        if (ientry < 0) break;
        nb = fChain->GetEntry(jentry); nbytes += nb;
        if(jentry%1000==0)cout<<"Number of evts analysed\t"<<jentry<<endl;
        //calling variables from .h file(header)
        b_px->GetEntry(jentry);
        hist_px->Fill(pxval);
    }
    TFile *f1=TFile::Open("Output.root","RECREATE");
    hist_px->Write();
    f1->Close();
}
[bawa@t3head 2017]$ root -l -b
root [0] .L Analyze.C
Info in <TUnixSystem::ACLIC>: creating shared library /nfs/t3nfs_common/home/bawa/roottutorial/2017./Analyze_C.so
root [1] Analyze t
Analyze & 0x7f1325b6f028
root [2] t.Loop()
Number of Entries 10000
Number of evts analysed 0
Number of evts analysed 1000
Number of evts analysed 2000
Number of evts analysed 3000
Number of evts analysed 4000
Number of evts analysed 5000
Number of evts analysed 6000
Number of evts analysed 7000
Number of evts analysed 8000
Number of evts analysed 9000
root [3] .q
[bawa@t3head 2017]$ ls -ltr *.root
-rw-r--r-- 1 bawa bawa 159861 Mar 8 16:02 kinematics1.root
-rw-r--r-- 1 bawa bawa 158886 Mar 8 16:05 kinematics2.root
-rw-r--r-- 1 bawa bawa 158454 Mar 8 16:07 kinematics3.root
-rw-r--r-- 1 bawa bawa 3804 Mar 8 17:16 Output.root
[bawa@t3head 2017]$ run
Chains

Scenario:
Perform an analysis using multiple ROOT files. All files are of the same structure and have the same tree.
Chains (cont.)

TChain::Add()

```cpp
root [3] TChain chain("T");
root [5] chain.Draw("fTracks.fPx")
root [6] myCanvas->cd(2);
root [8] chain.Draw("fTracks.fPx")
```
Demo: Changing "MyClass" to use a Chain

1. Just add few lines in Analyze.h

```cpp
#ifndef Analyze_cxx
Analyse::Analyse(TTree *tree) : fChain(0)
{
    // if parameter tree is not specified (or zero), connect the file
    // used to generate this class and read the Tree.

    TChain *chain = new TChain("tree","");  // Initialize TChain
    chain->Add("kinematics1.root");        // Add file to chain
    chain->Add("kinematics2.root");        // Add file to chain
    chain->Add("kinematics3.root");        // Add file to chain
    tree=chain;                             // Assign chain to tree
    if (tree == 0) {
        TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject("kinematics1.root");   // Open file
        if (f || f->IsOpen()) {
            f = new TFile("kinematics1.root"); // Create new TFile
            f->GetObject("tree",tree);         // Get object
        }
    }                                           // End if chain is null
    Init(tree);                                 // Initialize tree
}
```
ROOT has a set of global variables that apply to the session. For example, `gDirectory` always holds the current directory, and `gStyle` holds the current style. All global variables begin with “g” followed by a capital letter.
Global Variables

In addition to gROOT, there are some more global variables that apply to the session:

\textbf{gROOT}: gROOT holds information relative to the current session. By using the gROOT pointer you get the access to basically every object created in a ROOT program.

\textbf{gFile}: gFile is the pointer to the current opened file.

\textbf{gDirectory}: gDirectory is a pointer to the current directory.

\textbf{gPad}: A graphic object is always drawn on the active pad, no matter what it is. For that we have gPad that is always pointing to the active pad.

\textbf{gRandom}: gRandom is a pointer to the current random number generator.

\textbf{gEnv}: gEnv is the global variable with all the environment settings for the current session.
✓ gFile: gFile is the pointer to the current opened file in the ROOT session.

✓ gDirectory: gDirectory is a pointer to the current directory. The concept and role of a directory is explained in the chapter "Input/Output".

✓ gPad: A graphic object is always drawn on the active pad. We have gPad that is always pointing to the active pad. For example, if you want to change the fill color of the active pad to blue, but you do not know its name, you can use gPad.

root> gPad->SetFillColor()

✓ gRandom: gRandom is a pointer to the current random number generator. By default, it points to a TRandom3 object. Root> gRandom=new TRandom2(0)