

Table of Contents

Working with Trees in ROOT.....	1
Exercise 1: Writing and Reading Histogram from a file.....	1
Exercise 2: Writing and Reading a ROOT n-tuple.....	2
Exercise 3: Creating a ROOT Tree.....	3
Exercise 4: Reading a ROOT Tree.....	3
Exercise 5: Chaining ROOT Trees.....	5
Exercise 6: Using Tree Friends.....	5
Exercise 7: Using the TSelector class for analysing a TTree.....	6

Working with Trees in ROOT

Welcome to hands-on session dedicated on working with the Trees in ROOT. First will start with an exercise on the I/O of ROOT by storing and reading an histogram from a file

Exercise 1: Writing and Reading Histogram from a file

Exercise 1: Writing and Reading histogram from a file

Open a file then create a simple histogram, for example an histogram generated with exponential distribution. Fit it and write it in a file. Why the ROOT Canvas does not show the histogram ?

Hint Hide

Use `TFile::Open` to open the file or just create a `TFile` object. Call `TH1::Write` to write the histogram in the file after having filled it.

Solution Hide

```
#include "TFile.h"
#include "TH1.h"

void histogramWrite() {

    TFile f("histogram.root", "RECREATE");

    TH1D * h1 = new TH1D("h1", "h1", 100, 0, 10);
    for (int i = 0; i < 10000; ++i)
        h1->Fill(gRandom->Exp(5));

    h1->Fit("expo");
    h1->Draw();

    f.Write("h1");
    f.Close();
}
```

Now read the histogram from the file and plot it.

Hint Hide

Create a file object (or call `TFile::Open`) and then `TFile::Get`


Solution Hide


```
void histogramRead() {

    TFile * file = new TFile("histogram.root");

    TH1 * h1 = 0;
    file->GetObject("h1", h1);
    // you can also use nut you need to cast if you compile the code
    //TH1 * h1 = (TH1*) file->Get("h1");

    h1->Draw();
}
```

 You can also use the `TBrowser` to open the file and display the histogram.

 What is going to happen if you delete the file after having retrieved the histogram from the file ?

Exercise 2: Writing and Reading a ROOT n-tuple

Create a ROOT ntuple, a class `TNtuple` containing 4 variables (for example x, y, z, t). Fill the ntuple with data (for example 10000 events) where x is generated according to a uniform distribution, y a gaussian and z an exponential and t a Landau distribution. Write also the tuple in the file.

Hint Hide

Create and fill the tuple as explained in slide 21.

Solution Hide

```
#include "TRandom.h"
#include "TFile.h"
#include "TNtuple.h"

void exampleNtuple() {

    TNtuple data("ntuple", "Example N-tuple", "x:y:z:t");

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

        data.Fill(x,y,z,t);
    }
    // write in a file
    TFile f("ntuple_data.root", "RECREATE");
    data.Write();
    f.Close();
}
```

Afterwards having saved the file, re-open the file and get the ntuple. Plot each single variable of the tuple and also one variable versus another one (for example x versus y).

Hint Hide

Open the file, get the tuple object and call the Draw method (see for example slides 22 and 23).

Solution Hide

```
void exampleNtupleDraw() {

    TFile f("ntuple_data.root");

    TNtuple *ntuple=0;
    f.GetObject("ntuple", ntuple);

    ntuple->Draw("t");
    //ntuple->Draw("y:z");
    // to add a selection cut and a graphic option
    ntuple->Draw("y:z", "x>0", "colz");
}
```

Then loop on the tuples, and print every 100 entries on the screen

Hint Hide

Open the file, get the tuple object and loop on the entries as in slide 24

Solution Hide

```

void exampleNtupleRead() {

    TFile f("ntuple_data.root");

    TNtuple *ntuple=0;
    f.GetObject("ntuple", ntuple);

    // loop on the ntuple entries
    for (int i = 0; i < ntuple->GetEntries(); ++i) {

        ntuple->GetEntry(i);
        float * raw_content = ntuple->GetArgs();
        float x = raw_content[0];
        float y = raw_content[1];
        float z = raw_content[2];
        float t = raw_content[3];

        if (i%100) std::cout << x << " " << y << " " << z << " " << t << std::endl;
    }
    // write in a file


    f.Close();

}

```

Exercise 3: Creating a ROOT Tree

Create a ROOT tree which contains an EventData object. The Event data object is defined in the EventData.h attached file. The macro creating the tree and filling with them with the events is CreateTree.C. Look at the macro and try to understand. Run the macro to create and write the tree in the file.

 Re-make exercise 2 (the writing part), but instead of using the TNtuple class, use the TTree class. The way to do is to describe the variables when calling TTree::Branch. For example for defining a floating variable x in the TTree you do: `tree.Branch("x", &x, "x/F")`; . See the example at page 38 of the booklet ROOT Guide for Beginners.

Exercise 4: Reading a ROOT Tree

Read the tree from the file and by using the TBrowse or TTree::Draw plot for example the momentum of all the particle or the event size.

Now read the Tree with a macro and calculate the sum of all event sizes.

Hint Hide

- Open the file using its file name in `TFile::Open()` and get the Tree. Remember to check if the file pointer is not null. If it is null means the file is not existing.
- Get then a pointer to the tree.
- Connect a Tree Branch with the Data Member. We have to somehow connect the branch we want to read with the variables used to actually store the data by calling `TTree::SetBranchAddresses()`.
- Load the TTree data. For the analysis example we need to access the events' size, which is stored in the variable `eventSize`. But the TTree first needs to load the data for each event it contains. For that call `TBranch::GetEntry(entry)` in a loop, passing the TTree entry number from the loop index to `GetEntry()`. Again TBranch is the class name, but you obviously need to call it on an object. To know how many entries the tree contains, simply call `TTree::GetEntries()`. The branch is stored in `eventSizeBranch`.

- In the same loop, compute the total size of all events (simply add the current event size to the total size)
- Without the call to `GetEntry()`, the variables will not contain data. `GetEntry()` loads the data into the variables connected with the tree by the call to `SetBranchAddresses()`.
- Access the Analysis Result. At the end of the loop, print the sum of all event sizes. This sum shows you the real power of a TTree: even though you can analyze large amounts of data (our example tree with 22MB is tiny!) ROOT needs just a few MB of your RAM, no matter how many events you analyze. Imagine what it would be like if you had to load all data into memory, e.g. using a simple `vector<EventData>`

Solution Hide

```
#include "TFile.h"
#include "TTree.h"
#include "TBranch.h"

void AnalyzeTree()
{
    // Variables used to store the data
    Int_t      totalSize = 0;          // Sum of data size (in bytes) of all events
    Int_t      eventSize = 0;         // Size of the current event
    TBranch    *eventSizeBranch = 0;  // Pointer to the event.fEventSize branch

    // open the file
    TFile *f = TFile::Open("eventdata.root");
    if (f == 0) {
        // if we cannot open the file, print an error message and return immediatly
        printf("Error: cannot open eventdata.root!\n");
        return;
    }
    // get a pointer to the tree
    TTree *tree = (TTree *)f->Get("EventTree");

    // To use SetBranchAddress() with simple types (e.g. double, int)
    // instead of objects (e.g. std::vector<Particle>).
    tree->SetMakeClass(1);

    // Connect the branch "fEventSize" with the variable
    // eventSize that we want to contain the data.
    // While we are at it, ask the tree to save the branch
    // in eventSizeBranch
    tree->SetBranchAddresses("fEventSize", &eventSize, &eventSizeBranch);

    // First, get the total number of entries
    Long64_t nentries = tree->GetEntries();
    // then loop over all of them
    for (Long64_t i=0; i<nentries; i++) {
        // Load the data for TTree entry number "i" from branch
        // fEventSize into the connected variable (eventSize):
        eventSizeBranch->GetEntry(i);
        // compute the total size of all events
        totalSize += eventSize;
    }

    Int_t sizeInMB = totalSize/1024/1024;
    printf("Total size of all events: %d MB\n", sizeInMB);
}
```

You can also download the macro AnalyzeTree.C.

Exercise 5: Chaining ROOT Trees.

Use the macro you have used to create a TNtuple (or a TTree) in exercise 2 and run few times, but changing always the name of the file where the tree is stored. Use then the TChain class to merge the tree and draw some of the variables of the tree.

Solution [Hide](#)

Here is for example the macro to create several trees (We use the Tree class for showing how a TTree is built with simple variables)

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

void exampleTTree(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();
}
```

%

This are the few lines to create the TChain, that you can run directly from the prompt. You can also use wildcard's to chain many files

```
TChain chain("tree");
chain.Add("tree*.root")
chain.Draw("t")
```

Exercise 6: Using Tree Friends

Make a Tree with three variables (x,u) and you fill with some random variables that you prefer. Read from the file the Tree used in exercise 5 and add as a friend to this tree. Plot the x variable of the first tree versus the x variable of the second one.

Solution [Hide](#)

Here is the macro to create a second tree, containing x, u:

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

void exampleTTree2() {

    TTree data("tree_2", "Example TTree");
    double x, u;
    data.Branch("x", &x, "x/D");
    data.Branch("u", &u, "u/D");

    // fill it with random data
    for (int i = 0; i < 10000; ++i) {
        x = gRandom->Exp(100);
        u = gRandom->Uniform(0, 10);

        data.Fill();
    }
    // write in a file
    TFile f("tree_2.root", "RECREATE");
    data.Write();
    f.Close();
}
```

Here are the lines of codes to Draw the x of the first tree versus the x of the second tree with a selection depending on u and t. You can run these lines from the ROOT prompt.

```
TFile f("tree.root"); // to get the first tree
tree->AddFriend("tree_2", "tree_2.root");
tree->Draw("x:tree_2.x", "t < 100 && tree_2.u < 6", "COLZ");
```

Exercise 7: Using the TSelector class for analysing a TTree

Create your own Selector for the simple (x,y,z,t) TTree made in Exercise 5. Use TTree::MakeSelector to create your own Selector class. Inside the code of your Selector do the following:

- book an histogram in the initialisation routine, for one of the variable of the tree (e.g. the variable t)
- fill the histogram in the Process function
- draw the histogram in the Terminate function

Hint Hide

Here is what you need to do, after having opened the file with the tree

```
tree->MakeSelector("MySelector.C");
```

The file MySelector.h and MySelector.C will be created. Add in MySelector.h, inside the class MySelector, a new data member, the histogram you want to create,

```
TH1D * h_t;
```

Edit then the file MySelector.C and add in MySelector::SlaveBegin the booking of the histogram.

```
h_t = new TH1D("h_t", "t", 100, 0, 100);
```

In MySelector::Process the filling of the histogram after calling TSelector::GetEntry()

```
GetEntry(entry);
h_t->Fill(t);
```

In `MySelector::Terminate` the drawing of the histogram.

After having saved the file run the selection by doing (for example from the ROOT prompt):

```
TFile f("tree.root");  
tree->Process("MySelector.C+");
```

Solution [Hide](#)

See the attached file `MySelector.h` and `MySelector.C`.

This topic: Main > RootIRMMTutorial2013IOandTreesExercises

Topic revision: r8 - 2013-03-04 - LorenzoMoneta



Copyright &© 2008-2024 by the contributing authors. All material on this collaboration platform is the property of the contributing authors.
or Ideas, requests, problems regarding TWiki? use [Discourse](#) or [Send feedback](#)