Universität Zürich[UZH]
Physik-Institut

Lecture 8: Scripting using ROOT 5 and 6

1 Introduction

If you have not yet installed ROOT (https://ph-root-2.cern.ch/releases/release-62200/), you will have to use the installed binaries existing already within our physik.uzh.ch clusters.
Let's first secure shell (ssh) into our respective computers.
Once inside, run the following command:

```
source /app/cern/root_v6.18.04/bin/thisroot.sh
```

There will be no confirmation, so let's try to start root.

```
~> root -l
root [0]
```

Now you are in root command line. From here on, you can type in simple sections of your code to test it out. For example:

```
root [0] for (int i=0; i<10;i++){
root (cont'ed, cancel with .@) [1]cout<<"blah "<<i<<endl;
root (cont'ed, cancel with .@) [2]}
blah 0
...
blah 9
root [3]
```

To exit your root command line, you need to type in:

```
root [4] .q
```

There is a very large documentation on ROOT available in the following link https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html. However, in this class wee will only use the following:

```
.x //Executes the script that follows .x
.q //Quits the ROOT command line
```

You can also write a short script as the following:

```
1 //short.cxx
2 {
3         for (int i=0;i<10;i++){
4                 cout<<"script "<<i<<endl;
5         }
6 }
```

From the terminal, you can execute the above as the following.

```
> root -l short.cxx
root [0]
Processing short.cxx...
script 0
...
script 9
root [1]
```

Many physicists rely on ROOT to perform data analysis to the extent where compilation of a code is never done. You must note that this is a great disadvantage. There is a performance penalty of using ROOT and the greatest disadvantage of writing ROOT scripst is simply the fact it is scripting and not programming. Results will not be as fast and not as stable as a compiled program. Furthermore, it eliminates the possibility of using some of the parallel processing features. In the next class we will go over how to compile using ROOT packages, but for today, we will stay with scripting.
You should not that it is also possible to script in form of a proper function or a C++ code but some modification need to be made. For example

```
1  //main.cxx
2  #include<iostream>
3  #include<stdio.h>
4  #include<vector>
5  #include<fstream>
6  #include<sstream>
7  #include <ctime>
8
9  void check(int i=0){
10         cout<<"check "<<i<<endl;
11 }
12
13 int main(int argc,char *argv[]){//Main begins
14     return 0;
15 }//Main Ends
```

Will not work and yield the following error message:

```
root -l main.cxx
root [0]
Processing main.cxx...
input_line_12:2:2: error: no matching function for call to 'main'
 main() /* '.x' tries to invoke a function with the same name as the macro
  ↪  */
 ^~~~
/mnt/e/2018_DAMIC_UZH/2020_PHY224/Lecture_8_PHY_224_SJLPHI/examples/main.cxx:15:5:
 ↪  note: candidate function not viable: requires 2 arguments, but 0 were
 ↪  provided
int main(int argc,char *argv[]){//Main beg
```

The problem is with the possibility of the input argument in main. Also you need to know that ROOT command line includes many packages. This means, all of the loaded package headers are not needed. They are already loaded by the framework.
The following will work however.

```
//main.cxx
void check(int i=0){
        cout<<"check "<<i<<endl;
}
int main(){//Main begins
        vector<int> test;

    return 0;
}//Main Ends
```

```
root [1] .x main.cxx
(int) 0
```

As you can see, a real C++ code may or may not work with ROOT depending on which features are used. A number of modifications need to be made in the source C++ code in order to be executed by ROOT. Since these modified C++ codes do not need compilations, they are really scripts. As a result, you are more problem to memory leaks, and undefined behaviours.
You need to also know that the ROOT 6 syntax works perfectly on ROOT 5, but many of the supported ROOT 5 syntaxes do not work on ROOT 6. Furthermore, almost any C++ code that is compiled with either of the ROOT versions work regardless of which version is used. This not the case if you are scripting.
Before we go on talking more about the ROOT features themselves, we will start practising scripting what we already know.

= The practical programming part of this course will now begin for 60 minutes. =

2 Command line and scripting in C++ using ROOT

– Using the ROOT command line:

1. Count 6 to 15 using for loop.
2. Declare all simple variables in stack.
3. Declare all simple variables in heap then free them and delete the pointers.
4. Perform simple calculations and try to use ROOT as a calculator.

– Using the script execution feature:

1. Write a script enclosed by { } and count 2 to 11 using for loop.
2. Write a main script with simple calculation functions (addition, multiplication and etc.).
3. Write a main script with 2 x 2 matrix addition and multiplication function.
4. Write a main script with a custom object with 5 inner variables
5. Write a main script with a function that allows you to calculate mean, median, skew, sample standard deviation, maximum and minimum given an array in stack.
6. Same as above but for vector in heap and stack and array in heap.

= The theoretical lecture part of this course will now continue for 15 minutes. =

3 Basic ROOT features

Now that we are a little bit familiar with the ROOT command line and scripting features, let's go over some of its nice features. Arguably, the best feature in ROOT is its file browser. You can open a file browser called TBrowser from command line so that you can preview any and all codes, objects and images.

```
root [0] new TBrowser
(TBrowser *) 0x2f8f9d0
root [1]
```

Your TBrowser will open in your XQuartz, XMing or natively by Xserver as seen in **Figure 1** . If at this point if you get a display error, make sure that either Xming or XQuartz is open since by default they are not supposed to auto-start when the computer starts. Also keep in mind of the delays due to forwarding through secure shell.

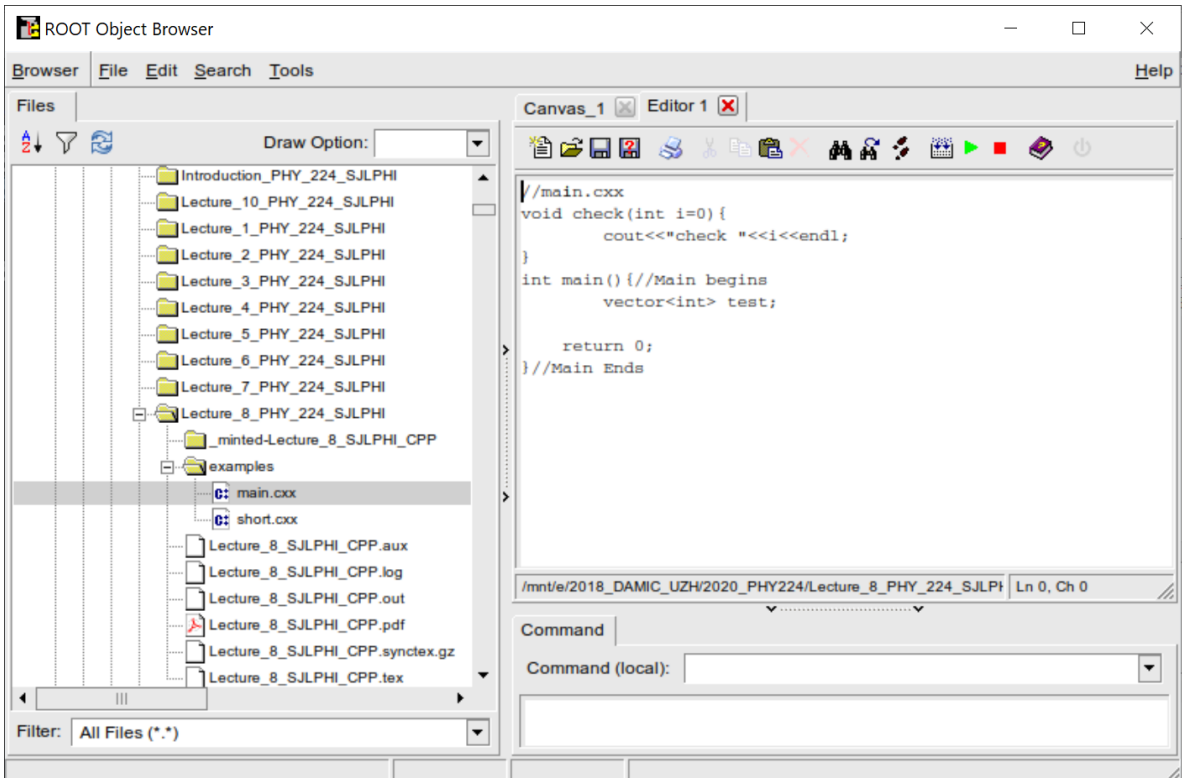

Figure 1: Your TBrowser may look different than this.

The TBrowser can be also useful for browsing data organized using ROOT called .root file. For the purpose of this course, we will only show the basic features of root.
Let's look at the following example:

```
1  //rootexample.cxx
2  //Main Begins
3  int rootexample(){
4          TRandom3* randomer=new TRandom3();
5          TFile * file= new TFile("myrootfile.root", "RECREATE");
6          TH1I * integerhist=new TH1I("integerhist","integerhist",10,0,100);
7          for (int i=0;i<1000;i++){
8                  integerhist->Fill( randomer->Gaus(50,10) );
9          }
10         file->cd();
11         integerhist->Write();
```

```
12      file->Close();
13          delete file;
14
15      return 0;
16  }//Main Ends
```

Note that the "main" has to match the filename in order to execute.
In the above, TRandom3 is used to create 1000 Gaussian distributed random numbers with the mean value 50 and standard deviation of 10. The Trandom3 is logged into a 1 dimensional histogram TH1I. The TH1I is titled and its pointer is named "integerhist", it has 10 bins from 0 to 100. Once the integerhist has been filled it is then written into the root file and saved.
We can run the above script and quit root after execution using the command:

```
> root -l rootexample.cxx -q
Processing rootexample.cxx...
(int) 0
```

"myrootfile.root" should now be created. Using TBrowser we can have a quick look at this file and the histogram within it seen below in **Figure 2** . Obviously you can do more things

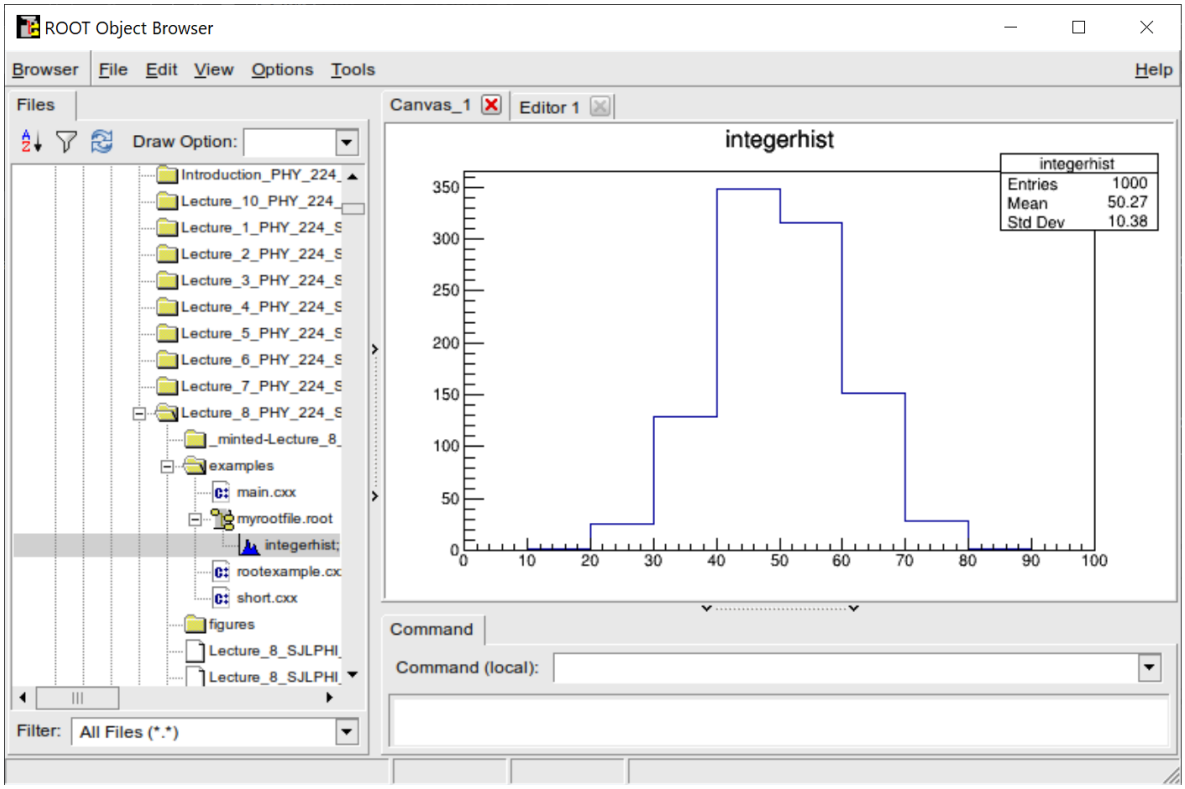

Figure 2: Newly generated TH1I in myrootfile.root.

to the histogram, and there are 2D and 3D histogram features in ROOT. We will use some of them during the practice session.
You should not that there are a lot of resources available online on ROOT and it is an important skill to be able to look them up and apply them into your script or code. The following exercise will use what has been demonstrated and can be found in the ROOT guide https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html.
You will be expected to look up the appropriate commands and implement them into your script.

= The practical programming part of this course will now begin for 60 minutes. =

4 Root Scripting: Root features

1. Create a script that allows you to save an empty .root file.
2. Create a script that allows you to read the above empty .root file.
3. Create a script that can read the above and write new information, and write a TH1I histogram with 1000 random variables generated by TRandom3::Gaus().
4. Same as above but using 10000 random variables generated by TRandom3::Uniform().
5. Create a script that creates and fills a 2 dimensional TH2I with 10000 TRandom3::Gaus() numbers. Use SetBinContent to set the content of histogram and use "colz" mode to draw and write into the root file.
6. Perform a fit on the above histograms using TBrowser.
7. Perform a fit on the above TH1Is using a built-in ->Fit("gaus") function from script.
8. Create a custom TF1 function and perform a fit on the above TH1Is.

5 Conclusion

From this class, we learned how to use the command line features and scripting features of ROOT and tried out some of the basic features available in ROOT. In the next class, we will do similar exercises but this time we will properly compile our C++ code but import the features available from ROOT.