Using ROOT

Gregory Maust

Summer 2008

This document is intended to take someone from knowing nothing about ROOT to a level which understands the programs used by Dr. Whisnant’s research group. In this application ROOT is used to analyze data collected by CODA (Jefferson Lab Common Data Acquisition System) from a NaI detector. More information about CODA and the hardware involved is available in Sean O’Brien’s paper on the subject [?]. You should have ROOT running while reading beyond the first section of this document, so that you can try things for yourself.

1 Starting

Unix Type root at a command line.

Apple While these are unix based, you may need to open a shell such as X11, then type root into its terminal window.

Windows When starting ROOT in Windows, you need to make sure you are running it from the installation directory. Right click the icon you’re using to run it and select Properties. Make sure the location listed beside Start in: is the installation directory, for example C:\root.

Once you’re in ROOT, you’ll see the prompt, which looks like this:

\texttt{root \[0\]}

The number is just the line number, telling you how many times you’ve given it a command. At the prompt you can use two types of input - commands to ROOT and C/C++ code. Commands going to ROOT itself start with a “.” For instance, the command to quit ROOT is \texttt{.q} (try it, and re-open ROOT). ROOT has a C/C++ interpreter, so we can also give it commands in that language. For instance, type \texttt{3 + sqrt(16)} and hit enter.

\begin{verbatim}
root [0] 3 + sqrt(16)
(const double) 7.00000000000000000e+00
root [1]
\end{verbatim}

\texttt{sqrt()} is part of the Standard C Library, so ROOT knows how to take the square root of the 16 you put into it. Note that every line not preceeded by a “.” is interpreted as a C/C++ statement.
2 Creating Objects

2.1 Histograms

If you’re using ROOT, you’re probably going to histogram something. Many, many somethings. You should read the TH1 Reference Page [?] (aside from the Methods list). We’ll use a histogram to show you the syntax of creating an object in ROOT. Note that ROOT is case sensitive.

root [0] TH1F *name = new TH1F('name','title', numbins, xlow, xhigh)

You start with the class of the item1, then its name and an equal sign relating it to what you want it to be. Many classes in ROOT start with T. The H indicates that it is a histogram, 1 specifies that it is one-dimensional, and F says that it will be storing data of the float form. Next you define the object’s name - the * is present because you’re really creating a pointer to the object. Then you say what it is - it’s a new TH1F, with specified properties. The first is the object’s name. Always match this to the name you made earlier (the pointer)2. The next is the title of the histogram - make it whatever you want, but make it meaningful. The final three should be entered as integers - they are the number of bins, what number to start at (included), and what number to end at (excluded). With this method, the bin size is automatically defined by root to fit the range and number of bins given. When you hit enter, you’ll see something like this:

(class TH1F*)0x1d97c70

You get the same result if you enter the name of the histogram and hit enter. It is ROOT telling you basic information about histo. As it told us before that $3+\sqrt{4}$ was a constant double with a value of 7.0, it is now telling us we’ve created something of the class TH1F. It then tells us where it is in terms of a hexadecimal memory address.

That’s a lot to remember to create one histogram. Fortunately ROOT has tab completion for most commands. Creating a new object is no exception. Type TH1F *histo = new TH1F( and then hit tab. ROOT lists the possibilities for continuing what you’ve typed, then neatly returns you to the command line with what you’d typed still there.

root [0] TH1F *histo = new TH1F( TH1F TH1F() TH1F TH1F(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup) TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)

1The two TH1F’s look redundant, and in fact when not using pointers the first can be dropped. However, with a pointer the first is necessary because it is the declaration of the variable.

2Why two names? One refers to the object, and one refers to its address, or something like that. If they don’t match, you might use the wrong name for a given situation, and it’s tricky to know which is right. The best method is to make them the same.
You can see that after the name and title, you can choose to specify the number of bins and the low and high ranges as before, or you can just tell it how many bins to make. Within the latter choice, you can give it a float or a double.

What if you just need a quick histogram for testing something? Creation commands can be significantly abbreviated. If you enter

```c
TH1F *histo = new TH1F()
```

ROOT will set defaults for what you don’t specify, and create an empty histogram. It will automatically expand for what you fill it with. You may not, however, specify a few things and then leave off the rest. The () should either be empty, or have a complete set of arguments. Look at the list from tab completion above. You could not create a histogram by typing

```c
TH1F *histo = new TH1F('histo','title')
```

### 2.2 TCanvas

Histograms don’t do much good if you can’t see them. A canvas is what ROOT draws things on. The creation is simple. Try the tab completion trick you learned to look at the options for drawing a canvas by typing what you see below and hitting tab.

```c
TCanvas *c1 = new TCanvas()
```

You can see that the abbreviated `c1 = TCanvas()` would work, as would specifying the name, title, and size. You can specify the width and height, or also tell it where to put the top left corner in turns of pixels. These measure from the left and top. Try

```c
c2 = new TCanvas("c2","canvas2",300,600,100,100)
```

That creates a 100 x 100 pixel canvas 300 pixels from the left edge of the screen and 600 pixels from the top. If you instead just enter the width and height...

```c
c3 = new TCanvas("c3","canvas3",100,100)
```

...it places it in the upper left corner by default. Leaving the location unspecified as in the latter example is a good idea, because the next person to use your program may have a lower screen resolution. If you create a canvas at 700 x 700 and they are running at 800 x 600, the canvas will be off the bottom, out of sight!
3 Using Objects

So now we have a histogram and something to draw things on. What can we do with them? As in C++, objects in ROOT have function members (methods) which can be called. A good example is the `Draw()` method of the TH1 class. While you will find out later that many classes have a `Draw()` member, `Draw()` isn’t a part of ROOT that draws things - it is a part of the histogram (or other class), which knows how to draw itself. There are many advantages to objects knowing how to draw themselves, instead of a big program that knows how to draw everything. The most obvious is that if you are only using histograms, you don’t need to load the information that’s used to draw everything else.

To use a method, you first call the object, then put a “-” or “.” between it and the method you wish to use. The “-” is used when the object is a pointer, and the “.” otherwise. Methods can be used successively, as in `object->method->method->method...`

A good example is resizing a histogram’s X-axis, which is outlined below.

3.1 Using Histograms

To draw a histogram named `histo`, type

```cpp
histo->Draw()
```

When you do this, ROOT finds histo, then asks it to draw itself. Try it yourself (you created histo earlier, right?), and note that ROOT will create a canvas to draw on if you haven’t already created one. Your histogram will be very boring, since you haven’t filled it with anything. Try using the Fill member to put a few entries in.

```cpp
root [0] histo->Fill(2,5)  
root [1] histo->Fill(6,3)  
root [2] histo->Draw()
```

Once you’ve done that, you can try using some of the options for `Draw()`. A fun one to try is `histo->Draw("LEGO")`. Try `LEGO2` next. Cool, isn’t it? It may look better if it had more than two non-zero bins. The fastest way to fill a histogram for playing is with the `FillRandom` member.

```cpp
histo->FillRandom("gaus",10000);
```

As the TH1 page phrases it, the following will “fill an histogram 10000 times with a default gaussian distribution of mean 0 and sigma 1.” `gaus` is a function that root knows, but you can also use functions you have created.

As mentioned earlier, methods can be used successively. There is no `SetXAxisRange` method for histograms. Instead you must ask the histogram to get its X axis, then ask it to set its range. Setting the range of the histogram itself would be too vague (vertical? horizontal?)

```cpp
histo->GetXaxis()->SetRangeUser(100,4095)
```
3.1.1 Some Useful Methods of TH1

**Draw**  This draws the histogram. In the case of the histogram, **Draw** is actually calling the **THistPainter** class. Accordingly, to find the available options for **Draw** you need to read the reference page for **THistPainter**.

**Fill**  This fills a bin in a histogram. `->Fill(n)` fills the n\textsuperscript{th} bin once. `->Fill(n,m)` fills the n\textsuperscript{th} bin m times. `->FillRandom("function",n)` will fill the histogram randomly based on the function, n times.

**Fit**  This fits a function to the histogram. `->Fit("gaus","R")` fits a gaussian to the histogram. The `R` option tells it to use the range of the function (\texttt{gaus} in this case) for the fit, instead of the range of the histogram. \texttt{gaus} is a function ROOT knows, but you can also create and use your own functions.

**GetXaxis()**  How to resize a histogram via command line isn’t obvious. As shown above, you must get the axis member then use its resize member.

**SetLineColor**  Sets the line color for a histogram. Very useful when overlaying multiple graphs or histograms. Numbers are used for colors - there is a convenient chart at the end of the document.

3.2 Using Canvases

You aren’t just stuck with making 5 canvases if you want to draw 5 things. Canvases can be divided into pads, whose class is TPad. If you give it one number, it divides the canvas evenly into that many vertical sections. If you give it two numbers, the first is the number of vertical sections and the second is the horizontal sections. Create a canvas named \texttt{c1} and try

\begin{verbatim}
c1->Divide(2,3)
\end{verbatim}

By hitting tab after the opening parantheses you may notice that you can adjust the margins and color of the pads created. You may be wondering what else ROOT can divide with this member. But you should recall that \texttt{Divide} is a member of TCanvas, \textit{not} something ROOT does to the canvas. The canvas knows how to divide itself.

Of course you aren’t forced to follow neat divisions of the canvas. You can arbitrarily create pads across any rectangular portion of the canvas using

\begin{verbatim}
TPad *p = new TPad("p","pad",0,0.1,0.5,0.7)
\end{verbatim}

Four numbers follow the usual name and title. The pattern is \texttt{xlo,ylo,xhi,yhi} (most sets of 2 ranges in ROOT follow this pattern). Do you notice what’s different about the numbers from the ones used in canvas creation? They are fractions. A pad’s size and position are described in terms of the canvas, not the screen, and measure from the bottom left. So the numbers above create a pad that reaches from the left side to halfway across, and from 0.1 to 0.7 above the bottom.
To switch between pads on a canvas, either center-click (push down on the mousewheel) or use the method \texttt{cd()}. The options are numbers, so to select the 4th pad on canvas \texttt{c1} you would enter

\texttt{c1->cd(4)}

If a canvas is divided like a grid, the pads are numbered in the direction you read. Try dividing a canvas into 9 pads laid out 3 x 3 and then navigate between them.

### 3.3 Using Other Objects

If you want to do something with an object, but aren’t sure what the command is, there are a couple things to try first. Tab completion is the easiest. After that is looking at the Reference Guide on CERN’s website [?]. Once you get used to reading the entries they are very helpful. On the TH1 page you can see the general info about the TH1 class, followed by a list of methods, followed by descriptions of each method.

### 4 Scripts

When you start using more than a few commands at once, retyping something you’re tweaking gets old, fast. A more efficient method is to enter a set of commands into a script with your favorite text editor, then execute the script.

#### 4.1 Execution

The location of your script is important - it should be in the folder you are running root from. On a unix machine, simply start root from the folder the script is located in. In Windows, save the scripts in the installation directory and run from there.\textsuperscript{3} Be wary of hidden extensions in Windows - you can name something \texttt{file.C} and waste a lot of time realizing it’s actually \texttt{file.C.txt} because the .txt was hidden. To avoid this, go to \textit{Tools} → \textit{Folder Options} in a folder window. Then go to the \textit{View} tab and make sure “\textit{Hide extensions for known file types}” is not checked.

To execute a script you have written, type \texttt{x filename}. To try this, save the following block of code as \texttt{canvas.C}

\begin{verbatim}
{
TCanvas *c1 = new TCanvas("c1","First",100,100);
TCanvas *c2 = new TCanvas("c2","Second",300,300);
c2->Divide(3,2);
}
\end{verbatim}

\textsuperscript{3}Go to \textit{Start→Run} and then enter \texttt{cmd}. In the terminal, enter \texttt{cd C:\root} or whatever directory you installed to, press enter, then type \texttt{root} and press enter.
Then open root and enter `.x canvas.C`. As you can see it executes all the commands inside. One thing it will trip up on is a space in a line. The following line will not work inside a script:

```
.x filename.C
```

This is because the whitespace gets stripped out, so ROOT sees `.x filename.C` and doesn’t know what to do with it. To get around this, use the command

```
gROOT->ProcessLine(
  "\".x filename.C\\n\""
)
```

The quotes protect the whitespace and the command will work.

### 4.2 Loading

If you want to use a function inside a program you’ve written, you’ll need to load the program first with `.L filename`. Once it is loaded you can call any functions inside the program as you would within code. Save this block as `program.C`

```c
int double(Int_t input) {
  Int_t output = input * 2;
  return output;
}
```

Now open root and type `.L program.C`. You can now use the function `double` inside. To do so, give it a number to double, for instance `double(2)`. ROOT will return 4. You can now use the function just like a program might. Try

```
3 * double(2).
```

Tada, 12! Here you loaded a program, then fed the function an input value via `functionName(inputParameter)`. The method when executing a script is not very different. You simply put the input value in brackets at the end, like this:

```
.x program.C(inputParameter)
```

**Quiz:** Use what you’ve learned about scripts to write one that accepts an input, uses your doubling function (via loading) on it, then triples the result. (This is not an exercise in efficient programming - it is practice calling functions.). My solution is at the end of the document.

### 5 Trees, Branches, & Leaves

You can read about this on the root website. Specific to our application, you should understand how an event relates to this structure. Each “event” is like
a period of growth. Leaves grow on various branches. Each PMT has its own branch. So one event may result in a leaf on shield 3, shield 4, shield 7, and the core. The leaf has information like the energy that was detected.

6 Further Learning

I don’t get it! Where else can I learn?

Practice, practice, practice! The best way to learn how to program, code, or work with a computer is to do it. The Doctors Niculescu have created an excellent set of exercises for learning ROOT [?]. You should definitely work through them.

A rough understanding of C or C++ isn’t strictly necessary to use ROOT, but is very helpful; find a book and get through at least the first several chapters. The book I had good luck with is in the references [?]. Finally, the answer to most any question about ROOT is somewhere on root.cern.ch [?], but actually finding said answer can be a time consuming process. The User’s Guide there is a good place to look for a less technical description of a given item than the reference pages.
7 Extras

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
</tr>
</tbody>
</table>

7.1 Quiz Solution

Used in the style of `.x quiz.C(input)`

```c
int quiz(Int_t inputQ)
{
    gROOT->ProcessLine(".L program.C");
    Int_t outputQ = 3*double(inputQ);
    return outputQ;
}
```
References


