The Basics of Root

Deborah Weber and Rory Kelly

March 16, 1999

Contents

List of Tables

List of Figures

1 Introduction

In the Supersymmetry group at CU-Boulder, we are going to change from using PAW to using ROOT. In order to make this transition smoothly, we need a guide to using ROOT for those of us who are unfamiliar with the ROOT system. There is a tutorial section on the web at the following address:

http://root.cern.ch/

The tutorials sometimes seem rather disjointed and do not always seem to apply to the work some of us may have done in the past. Therefore, it is necessary to create some sort of guide for ROOT that explains how to do much of what our group has done in the past. This paper will attempt to do just that.

Whenever a command is shown in this guide, it is assumed that the user will enter that command at the ROOT prompt, unless otherwise stated. If portions of a command are shown italics, it denotes that the user's input is required in some way. For example, if the portion of the command is shown as follows:

> *filename***.root**,

it means that the user should enter the filename he/she has designated for that file.

Additionally, in certain ROOT commands, the italicized portion may be in quotation marks. For example, one may see

> "*Browser Title*" .

The above portion of the command is the user's chance to customize his/her browser by giving it a title that has some meaning for him/her. It does not mean that the user needs to enter the exact string of words, although the quotation marks are definitely required.

Furthermore, ROOT's command system is sensitive to case and certain spacings. Therefore, it is in the user's best interests to enter the command *exactly* as it is shown in this guide.

2 Definitions

In ROOT, one must get used to new definitions. The following is a list of terms and their definitions:

Canvas	The window that is opened to display graphs
Class	A piece of C++ code that defines variables and functions related to a specific purpose; the names of all of ROOT's classes begin with a capital "T"
Object	A combination of code and data that acts as a self-contained "black box"
Pad	Sub-division of the canvas; the canvas can be sub-divided so that more than one graph can be put on one canvas
Pointer	A label that "points" to an actual address in memory where something is stored (i.e., an object)

3 Setting Up One's ROOT Access

One should ensure that the following is in the path statement which resides in the **.cshrc** file:

```
/cern/root/bin
```

Additionally, one must have the following in his/her **.cshrc** file[1] (I have it directly after the **setenv MAIL /usr/spool/mail/$USER:** statement in my **.cshrc** file):

```
setenv ROOTSYS /cern/root
setenv LD_LIBRARY_PATH /cern/root/lib
```

After this step is performed, the session needs to be re-initialized. This can be done by closing the dtterm or xterm that is in use and then opening a new dtterm or xterm. A faster, simpler way than closing the terminal and opening it again is to type at the UNIX prompt:

```
source .cshrc
```

[1] Or in his/her **.login** file

or

source .login

depending upon which file has been modified. Once that is done, the user can access the ROOT program.

4 Logging On and Logging Off

4.1 Logging On

Once the **.cshrc** (or **.login**) file has been modified for ROOT access, the user can log on to the ROOT program. To run ROOT, one must type the following at the UNIX command prompt:

root

After entering the above command, an orange box with a lightning bolt and the word "ROOT" in it will appear in the middle of the user's screen, followed by the following:

```
*****************************************
*                                       *
*        W E L C O M E  to  R O O T     *
*                                       *
*   Version   2.20/06   18 December 1998 *
*                                       *
*   You are welcome to visit our Web site *
*           http://root.cern.ch          *
*                                       *
*****************************************
CINT/ROOT C/C++ Interpreter version 5.13.83, Dec 2 1998
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

Note that the command prompt is now characterized by **root [0]**. The number in square brackets indicates the number of commands that have been entered. For example, if the command prompt looks like

> **root [5]**,

it means that 5 commands have been entered. As in UNIX, one can use the up arrow to scroll through the previously entered commands in order to repeat any typed commands without having to tediously re-type them. Also, unlike PAW, ROOT has a history file that saves many of the commands that were typed in previous ROOT sessions. Therefore, if one uses the up arrow key to scroll through the previous commands, he/she should be able to see commands that were entered during a previous session.

4.2 Logging Off

To complete the ROOT session, one types

> **.q**

at the ROOT command prompt. This will terminate the session and return the user to the UNIX prompt.

5 Getting Started

One of the attractive attributes of ROOT is that its **.root** files take up much less space than the **.hbook** files from NLCPROC.

Once data has been created using ISAJET, there is a (rather large) **.dat** data file or a **.xdr** data file on the directory. The first question is whether the data file can be made directly into a **.root** file or if it must first be converted to **.hbook** files using NLCPROC or to a .out file using GISMO.

5.1 Creating .root files from an ISAJET .dat or .xdr file

Unfortunately, this cannot be done. It is virtually impossible to create a **.root** file from an ISAJET **.dat** file or an ISAJET **.xdr** file. This is due to the fact that ISAJET is written in FORTRAN and its data file is therefore in a FORTRAN format. The ROOT system runs with the C++ format. The only way to make an ISAJET **.dat** file or **.xdr** file into a **.root** file is to convert the data file to **.hbook** files with NLCPROC or to a **.out** file with GISMO and then use the processes described in the following subsections to convert the **.hbook** file or **.out** file into a **.root** file.

5.2 Converting an NLCPROC .hbook file to a ROOT .root file

There is a utility called **h2root** included in ROOT which converts PAW files into ROOT files. In order to convert an **.hbook** file to a **.root** file, one must type the following at the UNIX command prompt:

h2root *filename*.**hbook** *filename*.**root**

It is not strictly necessary to include the second file name. If the second file name is omitted, **h2root** will use the same file name as before, substituting **.root** for **.hbook**.

Now, if the user has already done some analysis in PAW and has made a selection of histograms into an **.hbook** file, this process will convert those histograms to histograms that ROOT can manipulate. The histograms will have the same titles as before. For example, suppose the newly converted file had histogram 1000. The conversion process modifies it so that ROOT can understand that it is a histogram. What was histogram 1000 in PAW is now histogram h1000 in ROOT. This conversion is useful if one just wants to access the histograms one has already made and use the outstanding graphing capabilities available to the ROOT user. After this step is performed, the **.root** file can be manipulated and analyzed.

Possible errors

It may happen that the above procedure, rather than converting the file, instead produces an error message. This could be due to the fact that the user has forgotten to set the correct path. If this error occurs, go back to the *Setting Up One's ROOT Access* section of this guide and follow the steps taken there. That should correct the error.

5.3 Creating .root files from a GISMO .out file

If one wants to analyze GISMO data with ROOT, it is possible to convert the GISMO **.out** file to a **.root** file. There is a conversion process using the **asc2root** utility in ROOT that will accomplish this step. To convert a **.out** file to a **.root** file, one must type the following at the UNIX command prompt:

asc2root *filename*.**out** *filename*.**root**

5.4 Opening a .root file

Once the **.hbook** file (or the **.out** file) has been converted to a **.root** file, one can enter ROOT (as described above) and start to manipulate the file. To open the file, the following command is used:

TFile *f = new TFile("*filename*.root");

Description of the command:

TFile	The class of the ROOT object
f	The pointer being defined; the user can choose any name he/she wants for the pointer
=	Operation symbol indicating that an assignment is taking place
new	Command that tells ROOT that the pointer **f** is a new pointer
TFile	Command that tells ROOT that the pointer **f** points to a **TFile** object
"*filename*.root"	The name of the file the user wants to retrieve

A shortcut to the above command is the following:

TFile f("*filename*.root");

The above command is the equivalent of PAW's **h/file 1 *filename*.hbook**.

5.5 Combining multiple .root files

Since an ISAJET **.dat** data file is usually quite massive, the NLCPROC program creates many **.hbook** files from a **.dat** file. Each **.hbook** file must separately converted to a **.root** file and then combined so that all of the data can be analyzed.

As in PAW, it is not necessary to combine multiple **.root** files into one big **.root** file. All that one needs to do is to chain the ntuples within the **.root** files. The steps to chain the ntuples together are as follows:

TChain *chain* = new TChain("*ntuple*");

Description of the command:

TChain	The class of the ROOT object
chain	The pointer being defined; the user can choose any name he/she wants for the pointer
=	Operation symbol indicating that an assignment is taking place
new	Command that tells ROOT that the pointer **chain** is a new pointer
TChain	Command that tells ROOT that the pointer **chain** points to a **TChain** object
"*ntuple*"	The name of the ntuple which the user wants to chain

This command is followed by

chain ->Add("*filename1*.root");
chain ->Add("*filename2*.root");
chain ->Add("*filename3*.root");

Description of the commands:

chain	The pointer describing the name of the chain
->	The redirection operator; when referring to a member function (i.e., one that is defined by ROOT and not the user), the redirection operator -> can be replaced by **.**

Add	Command that tells ROOT to add the ntuple in the file to the chain
***filename1*.root**	Name of the file in which the first ntuple is stored
***filename2*.root**	Name of the file in which the second ntuple is stored
***filename3*.root**	Name of the file in which the third ntuple is stored

This process of adding ntuples to the chain can carry on until the user has listed all the files he/she wants to combine.

When the chaining has been completed, the combined ntuples can be plotted. The plotting is accomplished by the following command:

chain –>Draw();

The above command is similar to the command used to draw histograms. To see a description, see the subsection on drawing histograms.

To see the contents of the chain, the following command should be entered:

chain –>GetListOfFiles()–>Print();

or

chain –>GetListOfFiles()–>ls();

Description of the commands:

chain	The pointer describing the name of the chain
–>	The redirection operator
GetListOfFiles()	Command that tells ROOT to get the list of files inside the chain
–>	The redirection operator
Print()	Command that tells ROOT to output the list to the screen
ls()	Command that tells ROOT to list the contents on the screen

Shortcuts for the above commands are:

> *chain* −>Print();

or

> *chain* −>ls();

Either of these commands works equally well.

6 Ntuples

6.1 Making Ntuples from a .root file

When the **.hbook** file is converted to the **.root** file, it automatically creates an ntuple. This ntuple will contain all of the information that was stored in the PAW ntuple.

6.2 Viewing the contents of an ntuple

To see what is inside an ntuple, one can browse it. To browse an ntuple, enter the following command:

> TBrowser **browse* = new TBrowser("browse", "*Browser Title*");

Description of the command:

TBrowser	The class of the ROOT object
browse	The pointer being defined; the user can choose any name he/she wants for the pointer
=	Operation symbol indicating that an assignment is taking place
new	Command that tells ROOT that the pointer **browse** is a new pointer
TBrowser	Command that tells ROOT that the pointer **browse** points to a **TBrowser** object

"browse"	The name of the pointer; this name must be the same as the pointer that is being defined (see above)
"Browser Title"	The title given to the browser by the user; this title can be anything the user chooses

The above command will open a browser that will not only enable one to browse that ntuple, but will also give a window which can be used to view the contents of the disk.

6.3 Contents of Ntuples

Upon viewing the ntuple with the browser, the user will note that there are several different items inside the ntuple. The following is a list of variables and their definitions[2]:

Ievt	event number (from the program that generated the data)
Njset	number of partons from ISAJET
Costh1	cosθ of the first particle in the event
Costhn	cos θ of the primary positively-charged particle when two charged primaries are produced, otherwise it contains 1.1

Quantities that are defined for every particle in the event:

Npart	number of particles created
Px	x-component of momentum
Py	y-component of momentum
Pz	z-component of momentum
Ptot	total momentum
E	total energy
Mass	mass
Q	charge
Id	particle ID
Idk	child particles (or 0 if the particle is stable). If the particle decays to particles number $m \ldots n$, inclusive, then $Idk = 10000 \cdot m + n$

[2]The list is taken from David Wagner's paper *NLC SUSY Generation and Analysis*

Lvis	Flag indicating that the particle is visible (i.e., it is stable, it is neither a neutrino nor a $\widetilde{\chi}_1^0$, and the smearing routine says that it wasn't inefficient and that it didn't go down the beam pipe)
Lnu	the particle is a neutrino
Lbottom	flag indicating that the particle contains a b quark
Ltop	flag indicating that the particle contains a t quark
Lself	the particle is the "parent of itself". ISAJET will sometimes have a particle decay into itself
Ipar	particle ID of the parent of this particle
Ieff	flag returned from the smearing routine: **0** — particle was lost due to inefficiency **1** — particle was found **2** — particle was lost down the beam pipe hole
Jetpart	the jet with which the particle is associated

Visible particles (e.g., stable particles in the detector):

Nvis	number of visible particles
Ncharge	number of charged particles
Nneutral	number of neutral particles
Ngamma	number of photons
Nelec	number of electrons and positrons
Nelect	number of electrons
Nposit	number of positrons
Nmuon	number of muons and antimuons
Npion	number of charged pions
Npi0	number of π^0s
Nkaon	number of charged kaons
Nks	number of K_S^0s (those that don't decay)
Nkl	number of K_L^0s
Nproton	number of protons
Nneutron	number of neutrons
Nhbaryon	number of heavy baryons

Unstable particles:

Nhiggs	Number of Higgs bosons of all types: $h^0, H^0, H^{\pm}, A^0$
Nw	number of $W^{\pm}$
Nz	number of Z^0s
Nquark	number of quarks

Invisible particles:

Nz1ss	number of $\widetilde{\chi}_1^0$s
Nneutrino	number of neutrinos

Quantities associated with energy:

Evis	total visible energy
Echarge	total charged energy (i.e., energy determined by the momentum of charged particles in the tracking system, if the tracking system really did go down to $\cos\theta = 0.99$)
Eem	total electromagnetic energy (i.e., energy deposited by electrons and photons into the electromagnetic calorimeter)
Ehad	total hadronic energy (i.e., energy deposited by hadrons in the hadronic calorimeter
Emuon	total energy of muons
Ebeamp	energy escaping down the beam pipe
Eneutrino	Total energy of the neutrinos (i.e., sum total of energy of all neutrinos)
Eineff	energy lost due to detector inefficiencies

Quantities associated with thrust:

Thrust	the thrust of the event, defined as $T = \frac{\sum_i \lvert \vec{p_i} \cdot \widehat{T} \rvert}{\sum_i \lvert \vec{p_i} \rvert}$
Tvec	the components of $\widehat{T}$ This vector is normalized to 1, so *Tvec(3)* $= \cos\theta_{thrust}$

Tmajor value of T_{major}
Tmvec the components of $\widehat{T}_{major}$
Tminor the value of T_{minor}
Tmnvec the components of $\widehat{T}_{minor}$
Efor sum of the energy of all visible particles in the forward hemisphere
Eback sum of the energy of all visible particles in the backward hemisphere
Pfor momentum sum of all visible particles in the forward hemisphere
Pback momentum sum of all visible particles in the backward hemisphere

Quantities associated with jets:

Njet number of jets found
Pxjet x-component of the jet (i.e., the sum of the x-components of momentum of all the particles in the jet)
Pyjet y-component of the jet
Pzjet z-component of the jet
Ejet energy of the jet (i.e., the sum of the energies of all the particles in the jet)
Npjet number of particles in the jet
Qjet sum of the particle charges in the jet

Quantities associated with sigma:

Sigma cross section summed over types
Sigevt partial cross section for selected channel
Sigf average cross section (in millibarns). The equivalent running time (in seconds) will be $t = \frac{N}{\mathcal{L}\sigma} = \frac{Nevt}{(50 \cdot 10^{-7} fb^{-1} s^{-1}) \cdot (Sigf \cdot 10^{12})}$

6.4 Chaining ntuples

This topic is covered under the subsection *Combining multiple .root files.*

7 Histograms

There are many different types of histograms. Histograms can be 1-, 2-, or 3-dimensional. Additionally, the values stored in histograms can be character, short, floats, or doubles. Histogram class names tell exactly which kind of histogram one is seeing.

All histograms classes start with **TH**. The **T** begins all ROOT classes. The **H** denotes that this particular class is some type of histogram. Following the **TH** is the dimension: **1** for 1-dimensional, **2** for 2-dimensional, and **3** for 3-dimensional. After the dimension, the data type is listed: **C** for character, **S** for short, **F** for floats, and **D** for doubles. Hence, the histogram that has a class **TH1F** is a 1-dimensional floating point histogram (the type that is commonly used).

7.1 Making Histograms

Once an ntuple has been made or retrieved from memory[3], the user can make any number of histograms.

7.1.1 1-Dimensional Histograms

To create a 1-dimensional histogram, the following set of commands is used:

TH1F **h1D* = new TH1F("*h1D*","*Title of Histogram*",# of bins,low,high);

Description of the command:

TH1F	The class of the ROOT object
h1D	The pointer being defined; the user can choose any name he/she wants to define the pointer
=	Operation symbol indicating that an assignment is taking place
new	Command that tells ROOT that the pointer **h1D** is a new pointer
TH1F	Command that tells ROOT that the pointer **h1D** points to a **TH1F** object

[3]See the subsection entitled *Opening a .root File*.

"h1D"	The name of the pointer; this name must be the same as the pointer that is being defined (see above)
"Title of Histogram"	The title given to the histogram by the user; this title can be anything the user chooses
# of bins	As in PAW, the number of bins to have in this particular histogram
low	Again, as in PAW, the starting point for the histogram; this number is a real number, so it must have a decimal
high	Similarly, the stopping point for the histogram; it is a real number as well, so it must have a decimal, too

The above command tells ROOT that a new histogram is to be created. An example of the creation of a new histogram is shown below:

TH1F *myh1D = new TH1F("myh1D","Evis Distribution",150,0.,550.);

The above histogram, called **myh1D**, will have 150 bins. The title of the histogram, which will be displayed when the histogram is drawn, will be **Evis Distribution**. This histogram will display its contents from 0.0 GeV (assuming that it is filled with E_{vis} data) to 550.0 GeV.

The next step is to project the data onto the histogram. To project the data, the following command is used:

ntuple –>Draw("*Quantity* >> *h1D*");

Description of the command:

ntuple	The name of the ntuple with which the user is working
–>	The redirection operator
Draw	The command that tells ROOT to draw the histogram
Quantity	One of the variables that is listed in the ntuple; see the earlier descriptions of ntuple quantities shown in the subsection *Contents of Ntuples*
>>	The symbol used to tell ROOT that this particular drawing is going to be assigned to the histogram whose name follows the symbol

h1D The histogram name (pointer) defined in the previous step

An example of projecting data onto a histogram (the one that was previously created) is as follows:

myntuple –>Draw("Evis>>myh1D");

Histogram **myh1D** will be filled with visible energy data from the ntuple called **myntuple**. Figure 1 shows the output of the canvas following this command.

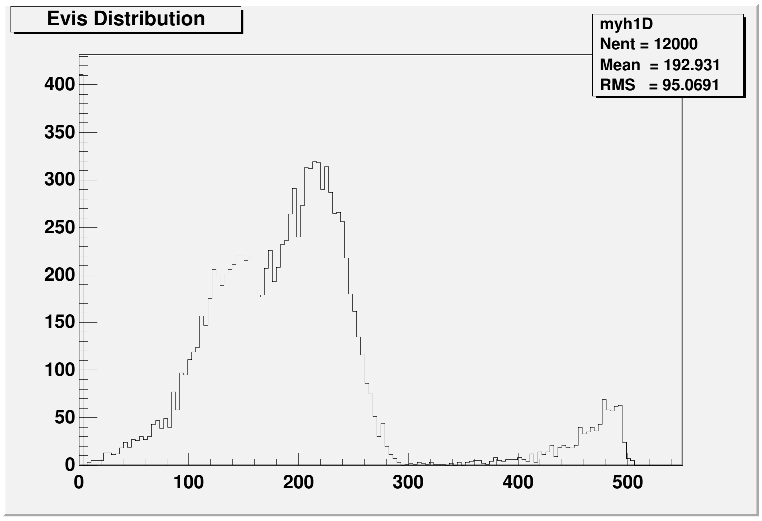

Figure 1: Canvas appearance after creating a histogram displaying visible energy.

If one wanted to make a histogram of the visible energy that was greater than 50 GeV, he/she could do so by typing the following command:

myntuple –>Draw("Evis>>myh1D","Evis>50");

Figure 2 shows the canvas output after the above command.

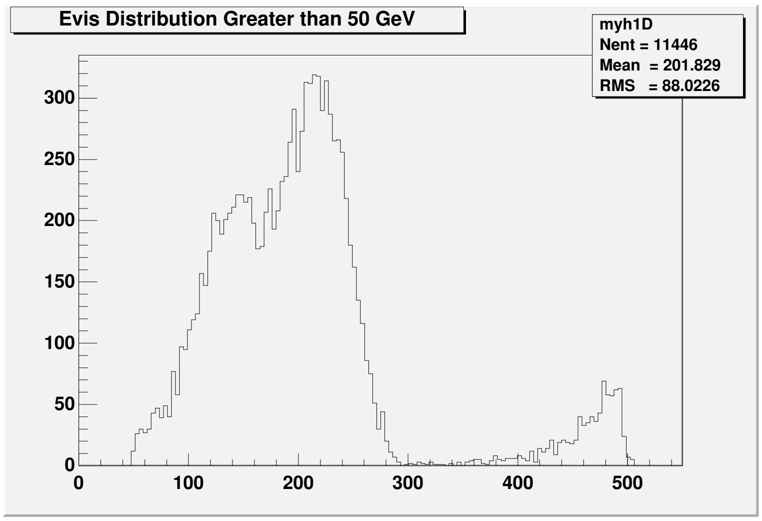

Figure 2: Canvas appearance after creating a histogram displaying visible energy greater than 50 GeV.

Additionally, to make a histogram of the visible energy greater than 50 GeV and less than 200 GeV, one would type

myntuple –>Draw("Evis>>myh1D","Evis>50 && Evis<200");

The canvas output of the above command is shown in Figure 3.

For the remainder of the ROOT session, the user can draw histogram **myh1D** and the computer will recognize it to be the histogram which was just created. However, if the user decides to leave ROOT and log on later, the histogram will have to be re-created, unless histogram is saved.

7.1.2 C++ logical statements

A number of logical statements can be inserted into the command lines of histograms and other commands. Table 1 shows the FORTRAN to C++ equivalencies.

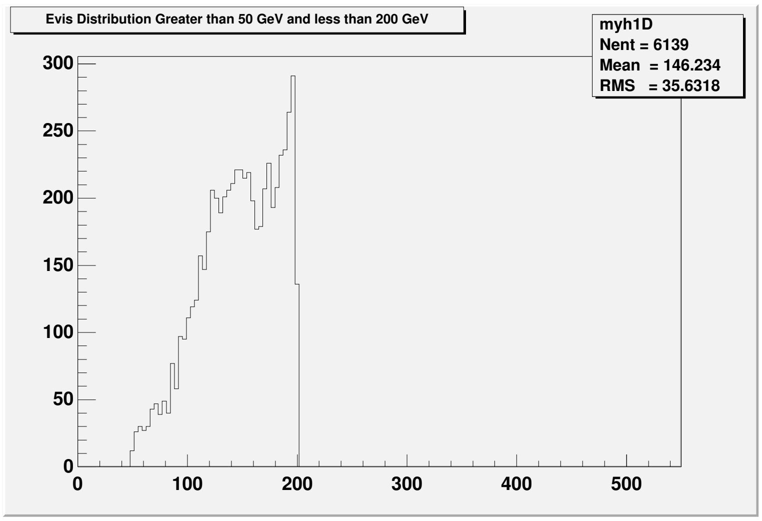

Figure 3: Canvas appearance after creating a histogram displaying visible energy greater than 50 GeV and less than 200 GeV.

Table 1: FORTRAN and C++ equivalent logical statements.

FORTRAN	C++	Function
.and.	&&	Logical AND
.or.	||	Logical OR
.ne.	!=	Not equal to
.eq.	==	Equal to
.gt.	>	Greater than
.lt.	<	Less than
.ge.	>=	Greater than or equal to
.le.	<=	Less than or equal to

7.1.3 2-Dimensional Histograms

To create a 2-dimensional histogram, the following set of commands is used:

TH2F **h2D* = new TH2F("*h2D*","*Title*",# xbins,xlow,xhigh,# ybins,ylow,yhigh);

Description of the command:

TH2F	The class of the ROOT object
h2D	The pointer being defined; the user can choose any name he/she wants to define the pointer
=	Operation symbol indicating that an assignment is taking place
new	Command that tells ROOT that the pointer **h2D** is a new pointer
TH2F	Command that tells ROOT that the pointer **h2D** points to a **TH2F** object
"h2D"	The name of the pointer; this name must be the same as the pointer that is being defined (see above)
"Title"	The title given to the histogram by the user; this title can be anything the user chooses
# xbins	As in PAW, the number of bins to have in this particular histogram for the 1st dimension
xlow	Again, as in PAW, the starting point for the 1st dimension of the histogram; this number is a real number, so it must have a decimal
xhigh	Similarly, the stopping point for the 1st dimension of the histogram; this number is a real number as well, so it must have a decimal, too
# ybins	The number of bins to have in this particular histogram for the 2nd dimension
ylow	The starting point for the 2nd dimension of the histogram; as with **xlow**, this number must have a decimal
yhigh	The stopping point for the 2nd dimension of the histogram; as with **xhigh**, this number must have a decimal, too

The above command tells ROOT to create a new 2-dimensional histogram. An example of the creation of a new 2-dimensional histogram is shown below:

TH2F *myh2D = new TH2F("myh2D","Eback vs. Efor",150,0.,550.,100,0.,200.);

The above histogram, called **myh2D**, will have 150 bins in the x-direction and 100 bins in the y-direction (x corresponding to E_{for} and y corresponding to E_{back}). The title of the histogram, which will be displayed when the histogram is drawn, will be **Eback vs. Efor**. This histogram will display its E_{for} contents from 0.0 GeV (assuming that it is filled with E_{for} data) to 550.0 GeV and its E_{back} contents from 0.0 GeV to 200.0 GeV.

As in the case of the 1-dimensional histogram, the next step is to project the data onto the histogram. To project the data, the following command is used:

ntuple –>Draw("*yQuantity*:*xQuantity* >> *h2D*");

Description of the command:

ntuple	The name of the ntuple with which the user is working
–>	The redirection operator
Draw	The command that tells ROOT to draw the histogram
yQuantity	This variable is the y-part of the 2-dimensional histogram
xQuantity	The x-part of the 2-dimensional histogram
>>	The symbol used to tell ROOT that this particular drawing is going to be assigned to the histogram whose name follows the symbol
h2D	The histogram name (pointer) defined in the previous step

An example of projecting data onto a histogram (the one that was previously created) is as follows:

myntuple –>Draw("Eback:Efor>>myh2D");

Histogram **myh2D** will be filled with data from the ntuple called **myntuple**. The data will be projected as E_{back} vs. E_{for}. The canvas output of the above command is shown in Figure 4.

If one wanted to make a 2-dimensional histogram of E_{for} vs. E_{back} with the visible energy greater than 50 GeV, he/she could do so by typing the following command:

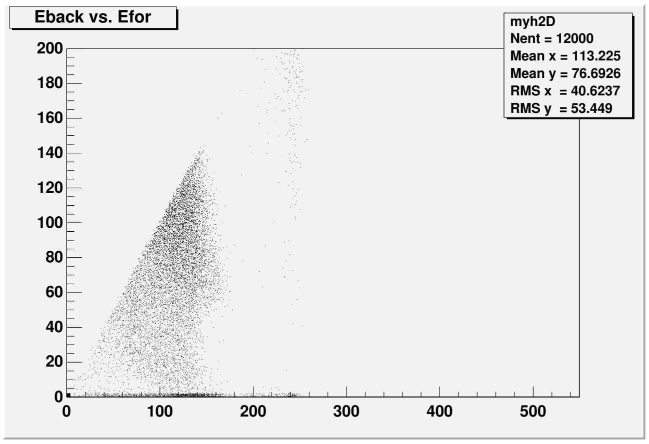

Figure 4: Canvas appearance after creating a 2-dimensional histogram displaying E_{back} vs. E_{for}.

myntuple –>Draw("Eback:Efor>>myh2D","Evis>50");

Figure 5 shows the canvas appearance following the above command.

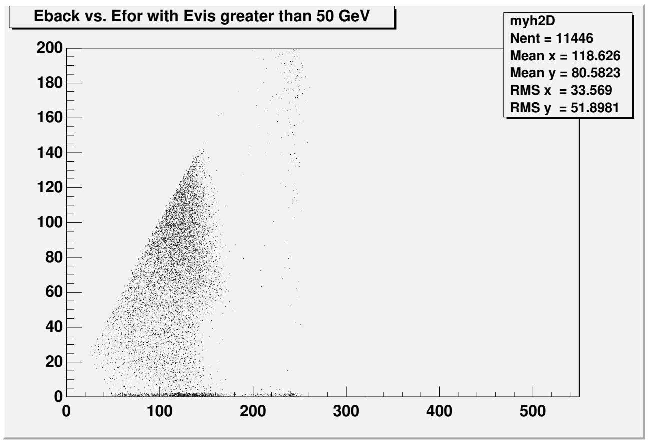

Figure 5: Canvas appearance after creating a 2-dimensional histogram displaying E_{back} vs. E_{for}, with E_{vis} greater than 50 GeV.

Additionally, to make a 2-dimensional histogram of E_{back} vs. E_{for} with the visible energy greater than 50 GeV and less than 200 GeV, one would type

myntuple –>Draw("Eback:Efor>>myh2D","Evis>50 && Evis<200");

Figure 6 shows the output to the canvas following the above command.

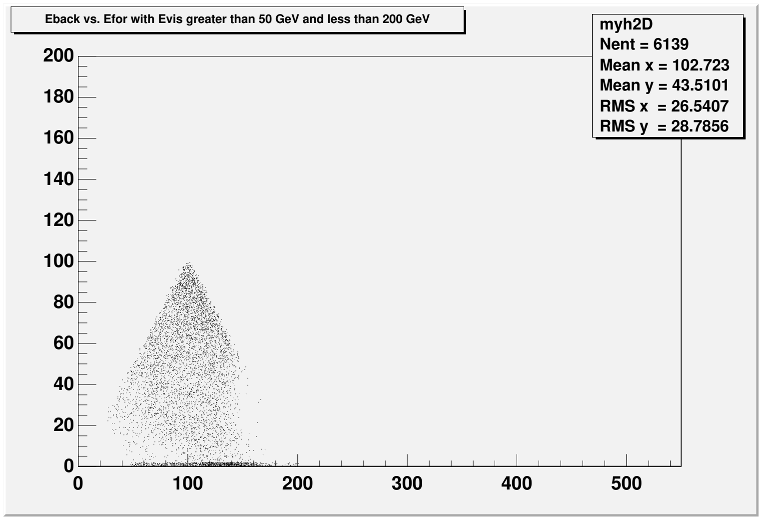

Figure 6: Canvas appearance after creating a 2-dimensional histogram displaying E_{back} vs. E_{for}, with E_{vis} greater than 50 GeV and less than 200 GeV.

As in the case of the 1-dimensional histogram, the user can draw histogram **myh2D** and the computer will recognize it to be the 2-dimensional histogram which was just created. However, if the user decides to leave ROOT and log on later, the histogram will have to be re-created, unless histogram is saved.

7.1.4 3-Dimensional Histograms

In principle, it is possible to draw 3-dimensional object. However, creating 3-dimensional histograms in the same way as the 1- and 2-dimensional cases above is not possible. To draw a 3-dimensional object, use the following command:

ntuple –>Draw("Px:Py:Pz");

This command will take the Px, Py, and Pz data from the ntuple and plot it on the canvas. The output to the canvas is shown in Figure 7. Note that there are no axes, title, or statistics box.

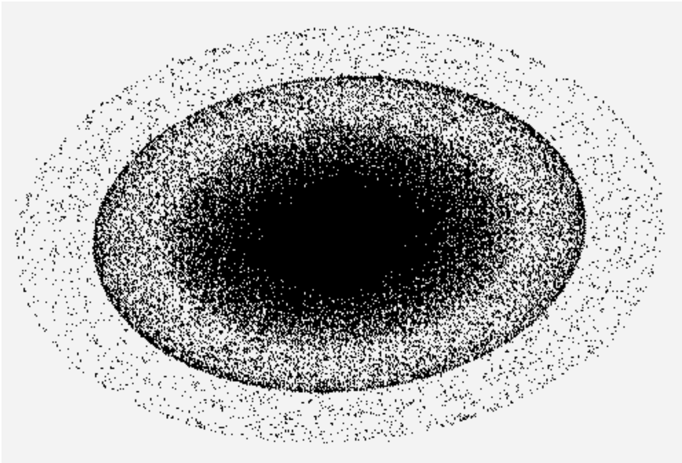

Figure 7: Canvas appearance after creating a 3-dimensional plot displaying Px vs. Py vs. Pz.

If one wanted to make a histogram of the momentum data as above, but take only those events whose visible energy was greater than 50 GeV, he/she could do so in the same manner as before:

ntuple –>Draw("Px:Py:Pz","Evis>50");

Figure 8 shows the output to the canvas after the above command is entered. Again, note that the axes, title, and statistics box are all absent.

Additionally, to make a histogram of the momentum data and only take those events whose visible energy greater than 50 GeV and less than 200 GeV, the command is

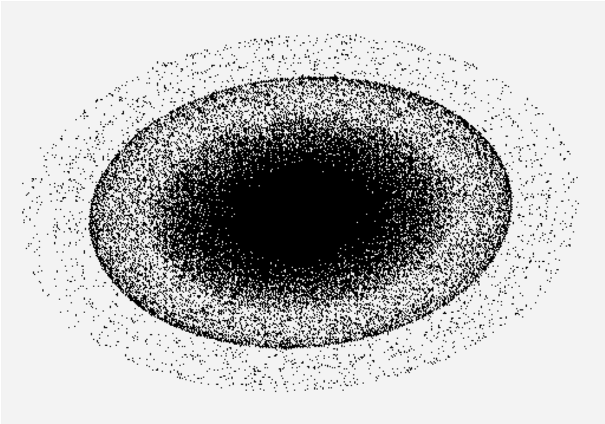

Figure 8: Canvas appearance after creating a 3-dimensional plot displaying Px vs. Py vs. Pz, taking only those events whose visible energy is greater than 50 GeV.

ntuple –>Draw("Px:Py:Pz","Evis>50 && Evis<200");

Figure 9 shows the output to the canvas after the above command is entered. Once more, the axes, title, and statistics box are missing.

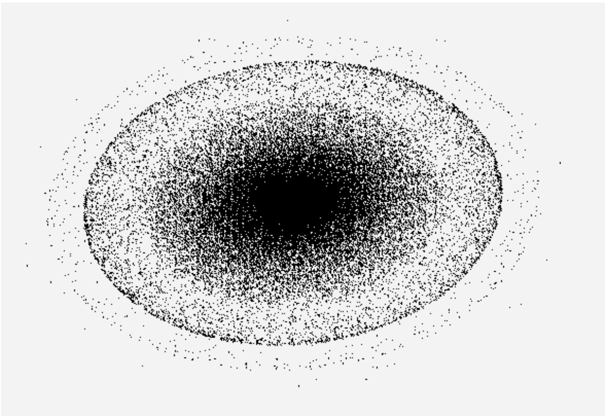

Figure 9: Canvas appearance after creating a 3-dimensional plot displaying Px vs. Py vs. Pz, taking only those events whose visible energy is greater than 50 GeV and less than 200 GeV.

7.2 Saving Histograms

Once a histogram is created, it is a simple matter to save it to a file. Only three commands are required to save a histogram to a file. The first command is

TFile **myfile* = new TFile("*filename*.root","RECREATE");

Description of the command:

TFile	The class of the ROOT object
myfile	The pointer being defined; the user can choose any name he/she wants for the pointer
=	Operation symbol indicating that an assignment is taking place
new	Command that tells ROOT that the pointer **myfile** is a new pointer
TFile	Command that tells ROOT that the pointer **myfile** points to a **TFile** object
"*filename*.root"	The name of the file to which the user wants to write
"*RECREATE*"	ROOT command that updates the file ***filename*.root**

An example of the above command is

TFile *myfile = new TFile("histfile.root","RECREATE");

The example command uses the pointer **myfile** to open a new file called **histfile.root**.

Following the opening of a new file, the histogram is saved by typing the following command:

hist –>Write();

Description of the command:

hist The name of the histogram that the user wants to save

–>	Redirection symbol (see earlier descriptions)
Write	ROOT command that tells the computer to save the the histogram to a file

An example of the command is below:

myh1D –>Write();

The example command will save histogram **myh1D** to the file **histfile.root**.

At this point, if there is more than one histogram to be saved to the file, the other histograms can be written. The user can wait, though, and create more as he/she wants and write the new histograms as he/she works with ROOT. The important thing to remember is that the file must be closed before the user logs off ROOT. To close the file, the user types the following command:

myfile –>Close();

Description of the command:

myfile	The pointer defined when the file was created
Close	ROOT command which closes the file

7.3 Working with Histograms

7.3.1 Basic histogram drawing

The first step after making a histogram is to plot the histogram. This can be done by typing

hist –>Draw();

Description of the command:

hist	Histogram title which the user wants to plot
–>	Redirection operator
Draw	ROOT command that tells the computer to draw histogram ***hist***

The above command will draw the histogram on the canvas. If there is no canvas, ROOT will automatically create one and label it “c1”.

7.3.2 Additional drawing commands

Superimposing If the user would like to superimpose one graph upon another, this can be done, just as in PAW. To superimpose one histogram on another (say, superimpose 1010 onto 1000), one would type the following:

h1000 –>Draw();
h1010 –>Draw(“SAME”);

This will superimpose histogram 1010 onto histogram 1000. See Figure 10 for the canvas output of the above command.

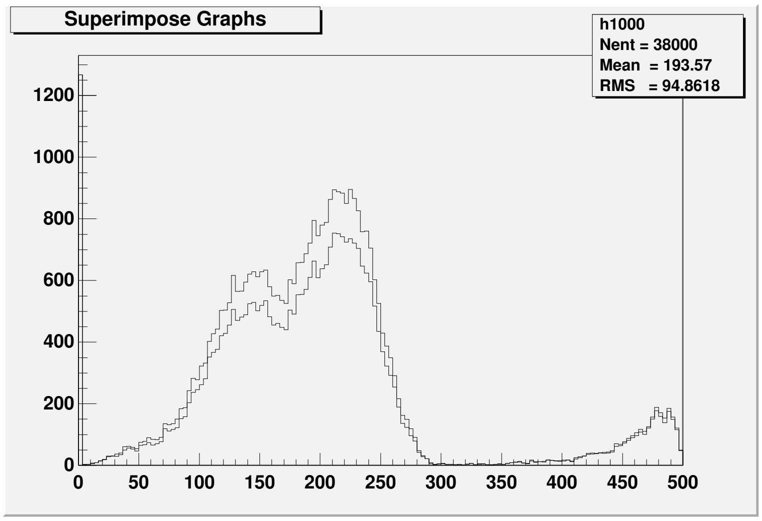

Figure 10: Canvas appearance after superimposing histogram h1010 onto h1000.

Coordinate system changes The default coordinate system for ROOT plots is Cartesian. If, however, the user wanted to change to a different coordinate system, ROOT can accomodate the user. To use cylindrical coordinates, the option **“CYL”** is inserted into the command line. **“POL”** will use polar coordinates, **"SPH"** will use spherical coordinates, and **"PSR"** will use pseudo rapidity/phi coordinates. An example of one of the above commands is

hist –>Draw("CYL");

The above command will draw the histogram in cylindrical coordinates.

Lego plots ROOT has three different lego options for its users. The lego options can be used with 1-dimensional and 2-dimensional histograms.

Using option **"LEGO"** will draw a lego plot with hidden line removal. Using option **"LEGO1"** will draw a lego plot with hidden surface removal. Using option **"LEGO2"** will draw a lego plot using colors to show the cell contents. An example of a command using a lego option is

hist –>Draw("LEGO2");

The above command uses the option of a lego plot that uses colors to show the cell contents to draw the histogram. Figure 11 shows an example of this type of lego plot on a 1-dimensional histogram and Figure 12 shows an example of a this type of lego plot on a 2-dimensional histogram.

Surface plot options There are five surface plot options supported by ROOT. These plot options are used with 2-dimensional histograms.

The first surface plot option is **"SURF"**, which draws a surface plot with hidden line removal. The second is **"SURF1"**, which draws a surface plot with hidden surface removal. The third is **"SURF2"**, which draws a surface plot using colors to show the cell contents. The fourth is **"SURF3"**, which shows a surface plot with hidden line removal and a contour view drawn on the top. The fifth and final option is **"SURF4"**, which draws a surface using Gouraud shading. To use these options, just as with the previous options, the user inserts the option between the parentheses following the word **Draw**.

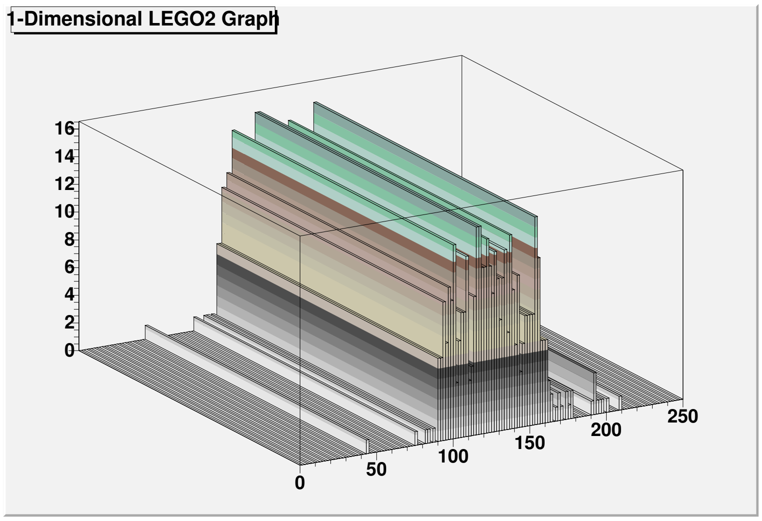

Figure 11: Canvas appearance after using the "LEGO2" command on a 1-dimensional histogram.

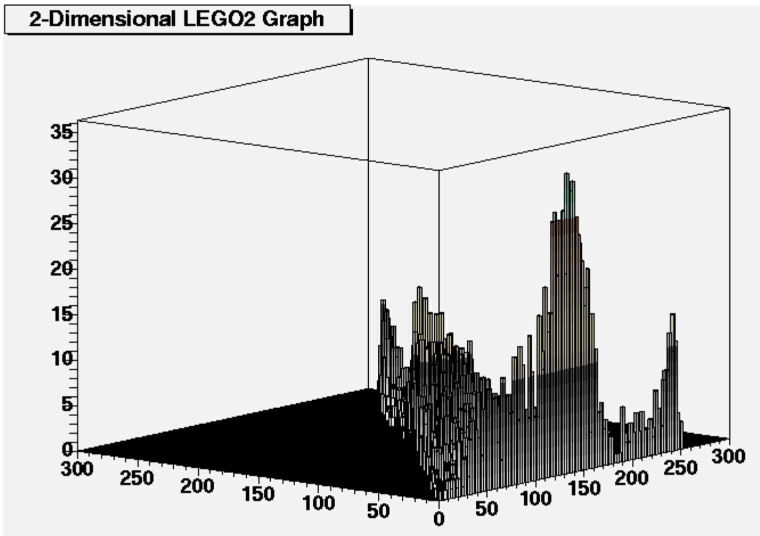

Figure 12: Canvas appearance after using the "LEGO2" command on a 2-dimensional histogram.

7.3.3 Changing colors

Line color Unfortunately, the default color for the lines of the histograms is black. Hence, it could be difficult to distinguish between histogram 1000 and histogram 1010, which has been superimposed upon it. If one would like to see the superimposed graph in a different color, the line colors of the histograms must differ.

ROOT far surpasses PAW in this instance. Once the line color of the histogram has been changed, it will remain that color until it is changed again. Not only that, but also—unlike PAW—none of the other histograms will be affected by this change. The ability to set the line color is a great time-saving device in ROOT.

To change the line color of the histogram, the following command is used:

```
h1010 –>SetLineColor(#);
```

where # is the number of the line color desired. To find the number of the line color is to use the pull-down menu on the canvas called **View**. In the **View** menu, there is an option called **Colors**. Selecting this option will cause a color palette to appear on the screen. This color palette shows the colors as well as their numerical values.

After changing the line color of the histogram, the histogram can once again be plotted. The new color will be shown in the plot. Figure 13 shows the superimposed plots of before with the color of h1010 changed.

Fill color Similar to line color, histograms can be filled with a color so that they are "colored in" as opposed to being drawn in with an outline. To change the color of the "filling" of the histogram, use the command

```
h1010 –>SetFillColor(#);
```

As with the **SetLineColor** command, the # refers to the color with which the user wants to fill the histogram. Refer to the **View:Colors** menu for a list of colors. Once again, after changing the fill color, re-type the **Draw** command, and the histogram will appear "colored in". Figure 14 shows the superimposition of h1010 (which is filled) onto h1000 (which is not filled).

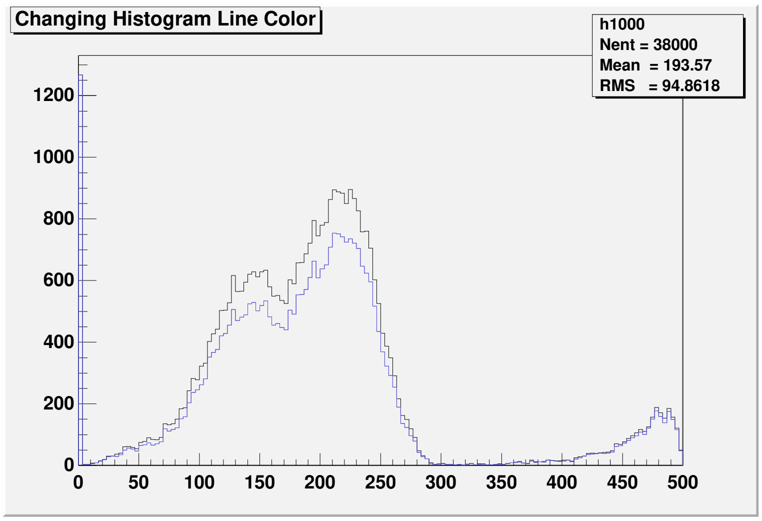

Figure 13: Canvas appearance after superimposing the newly color-enhanced histogram h1010 onto h1000.

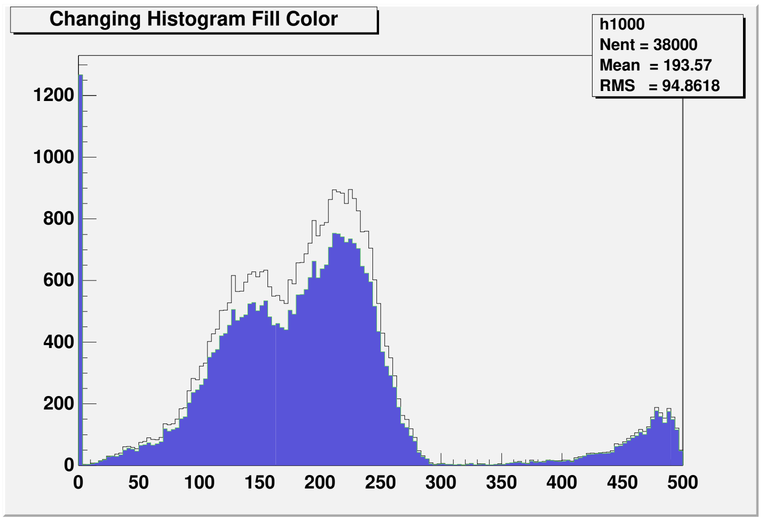

Figure 14: Canvas appearance after superimposing the newly color-filled histogram h1010 onto h1000.

7.3.4 Modifying histograms

A problem that can arise when working with histograms is that the pertinent bins are not always displayed. The binning can be changed in ROOT, but the histogram needs to be reset and filled again. To reset the bins, the following commands are used:

> *hist* –>Reset();
> *hist* –>SetBins(# of bins,low,high);

Description of the commands:

hist	Histogram name
–>	Redirection operator
Reset	Command that tells ROOT to reset the histogram
SetBins	Command that tells ROOT that the number of bins for the histogram will change
# of bins	The number of bins to have in the histogram
low	The starting point of the histogram
high	The stopping point of the histogram

Following the above two commands, the histogram must be filled again. Recall that the command for filling histograms is

> *ntuple* –>Draw("*Quantity >> h1D*");

for 1-dimensional histograms and

> *ntuple* –>Draw("*yQuantity:xQuantity >> h2D*");

for 2-dimensional histograms. In the 2-dimensional case, the **SetBins** command will require an additional description of the bins in the second dimension.

8 Graphical Output

When one plots an object from an ntuple, the default values for the canvas will show the title of the object, the numerical values for the x- and y-axes, and a statistics box. The statistics box has a default of showing the histogram name, the number of entries for the histogram, the mean, and the RMS value.

8.1 Histogram title and statistics box

8.1.1 Title

By default, the title of the histogram is written in the upper left corner of the canvas. It is placed in a shadow box. By clicking on the box with the right mouse button, the user can change various attributes of the title's box. A menu will appear while the right mouse button is depressed and will disappear if the button is released. The title of the menu is **TPaveText::title**. The options included in the **TPaveText::title** menu are as shown below:

AddBox	Adds a new graphics box to the title
AddLine	Adds a new graphics line to the title
Clear	Clears all the lines in the title
DeleteText	Deletes the text at the mouse pointer
EditText	Edits the text at the mouse pointer
InsertText	Adds a new text line at the mouse pointer
ReadFile	Reads the lines of filename in the title
SetAllWith	Sets attribute options for all lines containing string text
SetMargin	???
SetBorderSize	Determines the depth of the shadow box
SetCornerRadius	???
SetName	Change (or set) name of the object
Delete	Delete the object from CINT symbol table so that it cannot be used anymore
DrawClass	Shows all the draw classes and the options within those classes
DrawClone	Draw a clone of this object in the current pad
Dump	Gives information about the TPaveText; prints it to the screen
Inspect	Dumps contents of the object in a graphics canvas; same action as Dump but in a graphical form
SetDrawOption	Sets drawing option for object; only affects the drawing style

SetLineAttributes	Pulls up a menu that allows the user to modify the attributes of the line surrounding the title
SetFillAttributes	Pulls up a menu that allows the user to modify the attributes of the box in which the title resides
SetTextAttributes	Pulls up a menu that allows the user to modify the attributes of the text in which the title is written

To change the placement of the title, one can use the mouse to drag the box to its new location. By clicking on the left mouse button and dragging the mouse to the new spot, the title will move. Another use of the mouse upon the title box is to increase or decrease its size. The title box can be increased or decreased in the same manner as any computer's window. Position the mouse on the edge of the box, and an arrow will appear. Dragging the mouse will either enlarge or reduce the size of the title box.

The title displayed is usually the title that was specified when the histogram was made. If one is just picking objects out of the ntuple (either from the browser or from the command line), the title will be that object (i.e., if one double-clicked on the quantity **Efor** in the browser, the title will be **Efor**). If the user does not want that particular title, he/she can change it. To change the title of a histogram, the following command is used:

hist –>SetTitle("*Title of Histogram*");

Description of the command:

hist	Name of histogram
–>	Redirection operator
SetTitle	Command that tells ROOT that a title is going to be added to the existing graph
Title of Histogram	Title the user chooses for the histogram

If one does not want a title at all, one can "hide" the title. In the pull-down menu of the canvas entitled **Options**, there is an option of whether to show the histogram title. If there is a check mark next to this option, the histogram title will be displayed. If there is no check mark, then the title of the histogram will not be displayed.

8.1.2 Statistics box

By default, the statistics box is displayed in the upper right corner of the canvas. It is placed in a shadow box. By clicking on the box with the right mouse button, the user can change various attributes of the statistics box. A menu will appear while the right mouse button is depressed and will disappear if the button is released. The title of the menu is **TPaveText::stats**. The options included in the **TPaveText::stats** menu are the same as those listed for the **TPaveText::title** menu, with the modification that any time the word "title" is shown in the description of the option, substitute "statistics box".

If, for some reason, the user does not like the statistics that are displayed and only wants certain options, he/she can modify the statistics box. The command to modify the statistics box is

gStyle –>SetOptStat(######);

Description of the command:

gStyle	Command that tells ROOT the graphing style is going to change
–>	Redirection operator
SetOptStat	Command that tells ROOT which part of the graphing style is going to change
######	Each of these numbers is either a 1 or a 0, corresponding to a choice of statistic to be displayed; a 1 means to display the statistic and a 0 means to suppress the statistic The choices (in order) are as follows: – Number of underflows – Number of overflows – Root mean square deviation – Arithmetic average (mean) – Number of entries – Name of object (i.e., histogram name)

As with the title, the statistics box can be moved and modified by using the mouse. It can also be removed, or "hidden" by using the **Options** pull-down menu on the canvas in a way similar to the way the title is removed.

8.2 The Axes

8.2.1 Axis Titles

By default, when a ROOT histogram is drawn on the canvas, axis titles are not displayed. To display axis titles, the user must enter a separate command. The command is as follows:

For the x-axis:

hist –>SetXTitle("*X-Axis Title*");

For the y-axis:

hist –>SetYTitle("*Y-Axis Title*");

Description of the commands:

hist	Name of the histogram
–>	Redirection operator
SetXTitle	Command that tells ROOT that the x-axis title is about to be defined
SetYTitle	Command that tells ROOT that the y-axis title is about to be defined
"*X-Axis Title*"	User's choice for the x-axis title
"*Y-Axis Title*"	User's choice for the y-axis title

There is another way to change the axis titles. One can use the mouse. To change axis titles using the mouse, position the cursor on any of the numbers along the axis (a pointing hand will note that the cursor is in the correct spot) and depress the right mouse button. A menu will pop up on the screen with the title **TAxis::xaxis** (or **TAxis::yaxis**, if one is clicking on the y-axis). When the right mouse button is released, the menu will disappear. The second option under the menu title is **SetTitle**. This option changes (or sets, if there is no current title) the title of the axis. Choosing this option causes a small dialog box to appear which allows the user to input a title. Clicking on **OK** or pressing the Return key after entering the title will apply the change to the graph.

Sometimes, the font in which the axis title is written is not as pleasing to the eye as the user would like. This is a simple problem to correct. The solution is to use the mouse to change the font. By positioning the mouse pointer on the axis until a little pointing hand appears and clicking with the right mouse button, the **TAxis::xaxis** or **TAxis::yaxis** menu will appear. One of the options under this menu is **SetLabelFont**. Choose this option to change the font of the axis. The font style corresponds to a number, just as the color corresponds to a number. The default font number is 62. Unfortunately, I have been unable to find a list of font numbers. At the present time, one just has to use trial and error. One should be aware that just because the canvas appears to have a certain font, the **.ps** file may not share that font. There is an option under the **View** pull-down menu of the canvas entitled **Fonts**, but I have not been able to get any font information from it.

8.2.2 Axis Options

A complete listing of the options under the **TAxis::xaxis** menu (and the **TAxis::yaxis** menu) is as follows:

SetName	Sets the name of the axis; for instance, one can choose to call the vertical axis the x-axis and the horizontal axis the y-axis
SetTitle	As described earlier, this option allows the user to give the axis a title
Delete	Using this option has a tendency to kill ROOT
DrawClass	Shows all the draw classes and the options within those classes
DrawClone	Draw a clone of this object in the current pad
Dump	Gives information about the graphics options; prints it to the terminal screen

Inspect	Opens a window in which information about the axis is displayed
SetDrawOption	Sets drawing option for object; only affects the drawing style
SetNdivisions	Sets the number of divisions on the axis
SetAxisColor	Sets the color of the actual axis (not the axis label)
SetLabelColor	Sets the color of the axis label
SetLabelFont	As described above, sets the font of the axis label
SetLabelOffset	Sets the distance between the axis and the numbering
SetLabelSize	Sets the font size of the axis numbering
SetTickLength	Sets the length of the tick marks on the axis
SetTitleOffset	Sets the distance between the axis numbering and the axis title
SetTitleSize	Sets the font size of the axis title

8.3 The Canvas

The canvas is where the graphical output goes. It is similar to the HIGZ window that opens when PAW is run. The difference between the HIGZ window and the ROOT canvas is the fact that the canvas does not automatically appear when ROOT is first started. The canvas will automatically be generated when a **Draw** command is given to the computer if there is no canvas already present. The canvas can also be created by the user. In fact, multiple canvases can be created and used.

To create a canvas, the following command is used:

TCanvas **canvas* = new TCanvas("*canvas*","*Canvas Title*");

Description of the command:

TCanvas	The class of the ROOT object
canvas	Pointer that is being defined
=	Command that tells ROOT an assignment is taking place
new	Command that tells ROOT that ***canvas*** is a new pointer
Canvas Title	Title of the canvas; this title will be displayed at the top of the window containing the canvas

If there are multiple canvases on the screen, one can specify the canvas to be used. To change to a specific canvas, the following command is used:

canvas –>cd();

Description of the command:

canvas	Pointer that was defined when canvas was created
–>	Redirection operator
cd	Command that tells ROOT to change to the canvas ***canvas***

One can also change canvases by using the mouse. Simply place the mouse on the desired canvas and click the middle mouse button. The left mouse button will highlight the window *without* activating the canvas. A red line around the canvas denotes that particular canvas is the active one.

Canvases can be modified with the mouse button the same way windows can. The operations one can perform on the canvas with a mouse are dragging, stretching, minimizing, and maximizing. Additionally, objects inside the canvas can be dragged and stretched.