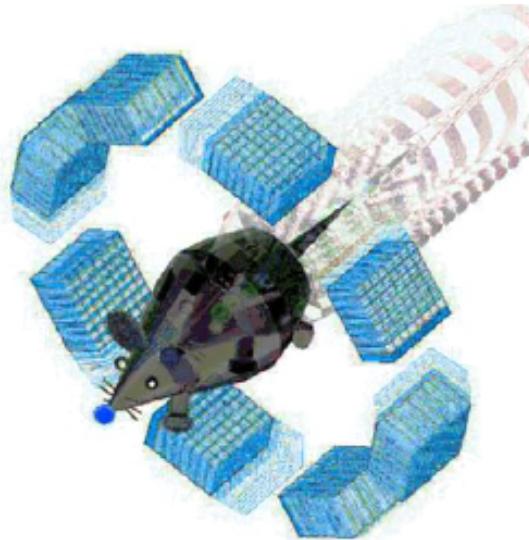




# ROOT manual for GATE users

Olga KOCHEBINA

[kochebina@gmail.com](mailto:kochebina@gmail.com)



Version 1.0

August 6, 2018



# Contents

<b>1</b>	<b>Obtain your ROOT file</b>	<b>4</b>
<b>2</b>	<b>Painless ROOT file opening</b>	<b>4</b>
<b>3</b>	<b>Smart plotting of your histograms (without TBrowser and python scripts)</b>	<b>6</b>
3.1	Colors, markers, lines . . . . .	8
3.1.1	Color . . . . .	8
3.1.2	Size . . . . .	9
3.1.3	Style . . . . .	9
3.2	How to apply a simple selection . . . . .	12
3.3	How to rebin a histogram . . . . .	12
3.4	Superposition of two distributions . . . . .	14
3.5	Superposition of two distributions from different ROOT files . . . . .	16
3.6	Simple fitting . . . . .	16
<b>4</b>	<b>Scan option</b>	<b>17</b>
<b>5</b>	<b>Simplest script to draw with ROOT</b>	<b>19</b>
<b>6</b>	<b>Example of script making beautiful plots</b>	<b>19</b>
<b>7</b>	<b>Adding a new variable to your TTree*</b>	<b>22</b>

## **Abstract**

The manual for GATE users who wants to become a friend with ROOT output files. This documents contains the simplest actions with a ROOT files. They are so simple that you cannot find it easily. In some sense it is the manual of "From what to begin" of ROOT using. Please, don't hesitate to contact [kochebina@gmail.com](mailto:kochebina@gmail.com) if you have any questions and comments. And of course, I strongly invite you to visit and use the ROOT web site, <https://root.cern.ch/>. This manual is very brief introduction and may just help you only in your first steps. Anyway, I hope that this document will save you some time.

Have fun!

*I'd like to thank Benoit VIAUD who thought me almost everything that you will find in this manual.*

## 1 Obtain your ROOT file

The standard way to get ROOT output is to add in your macro:

```
/gate/output/root/enable  
/gate/output/root/setFileName      YourOutputFile
```

This command creates a file `YourOutputFile.root`, which contains usually several ROOT TTree objects, like Hits, Singles, Coincidences. TTree is ....

One can choose the standard TTree with commands:

```
/gate/output/root/setRootHitFlag    1  
/gate/output/root/setRootSinglesFlag 1
```

It is possible to have some additional TTrees, for example, with an energy selection. To do so one should add in digitizer something like:

```
/gate/digitizer/name peak171  
/gate/digitizer/insert singleChain  
/gate/digitizer/peak171 setInputName Singles  
/gate/digitizer/peak171/insert thresholder  
/gate/digitizer/peak171/thresholder setThreshold 153.9 keV  
/gate/digitizer/peak171/insert upholder  
/gate/digitizer/peak171/upholder setUphold 188.1 keV
```

This will add to the .root file a TTree with a name peak171 which will contain Singles events with energies [153.9,188.1] keV.

## 2 Painless ROOT file opening

The simplest way to open your ROOT file is to type in terminal:

```
> root -l YourOutputFile.root
```

option -l will remove the ROOT start image.

There are two ways to check the content of your file:

1. use a `TBrowser`, which is not very powerful and gives an access to visualized structure of your ROOT file but very limited if for selection application or two histograms superposition
2. I strongly recommend use command:

```
root[0] .ls
```

This command will show you the list of all ROOT objects that you have in your file or that you created in the current ROOT session (see Figure 1). The names of TTree's are often repeated twice, this should not bother you.

```
root [1] .ls
TFile**      YourOutputFile.root      ROOT file with histograms
TFile*       YourOutputFile.root      ROOT file with histograms
KEY: TTree    Hits;236      The root tree for hits
KEY: TTree    Hits;235      The root tree for hits
KEY: TTree    Singles;160     The root tree for singles
KEY: TTree    Singles;159     The root tree for singles
KEY: TTree    spectrum;149   The root tree for singles
KEY: TTree    spectrum;148   The root tree for singles
KEY: TTree    peak171;73     The root tree for singles
KEY: TTree    peak171;72     The root tree for singles
KEY: TTree    peak245;43    The root tree for singles
KEY: TTree    peak245;42    The root tree for singles
KEY: TTree    scatter;16     The root tree for singles
KEY: TTree    scatter;15     The root tree for singles
KEY: TH1D    latest_event_ID;1   latest_event_ID(#)
KEY: TH1D    total_nb_primaries;1 total_nb_primaries(#)
KEY: TTree    OpticalData;1   OpticalData
root [2] □
```

Figure 1: Output of .ls command

In order to get the content of each TTree, i.e. list of saved histograms or "branches", for example for Hits TTree, one should do:

```
root[1] Hits->Show()
```

The results of this command is shown in Figure 2

```
root [2] Hits->Show()
=====> EVENT:-1
PDGEncoding      = 0
trackID          = 0
parentID         = 0
trackLocalTime   = 0
time              = 0
edep              = 0
stepLength        = 0
trackLength       = 0
posX              = 0
posY              = 0
posZ              = 0
localPosX         = 0
localPosY         = 0
localPosZ         = 0
momDirX           = 0
momDirY           = 0
momDirZ           = 0
headID            = 0
crystalID         = 0
pixelID           = 0
unused3ID          = 0
unused4ID          = 0
unused5ID          = 0
photonID          = 0
nPhantomCompton   = 0
nCrystalCompton   = 0
nPhantomRayleigh  = 0
nCrystalRayleigh  = 0
primaryID          = 0
sourcePosX         = 0
sourcePosY         = 0
sourcePosZ         = 0
sourceID           = 0
eventID            = 0
runID              = 0
| axialPos          = 0
```

Figure 2: Output of Show() command

### 3 Smart plotting of your histograms (without TBrowser and python scripts)

To plot a histogram one should use option `TTree -> Draw("name_of_the_branch")`. For example, if you want to plot 1D, 2D or 3D histograms of energy you could do

```
root[1] Hits->Draw("edep") for 1D (Figure 3, top)
```

root[2] Hits->Draw("posX:posY") for 2D, where the first argument, posX is ordinate axis and posY is abscissa axis (Figure 3, middle)

root[3] Hits->Draw("posX:posY:posZ") for 3D (Figure 3, bottom)

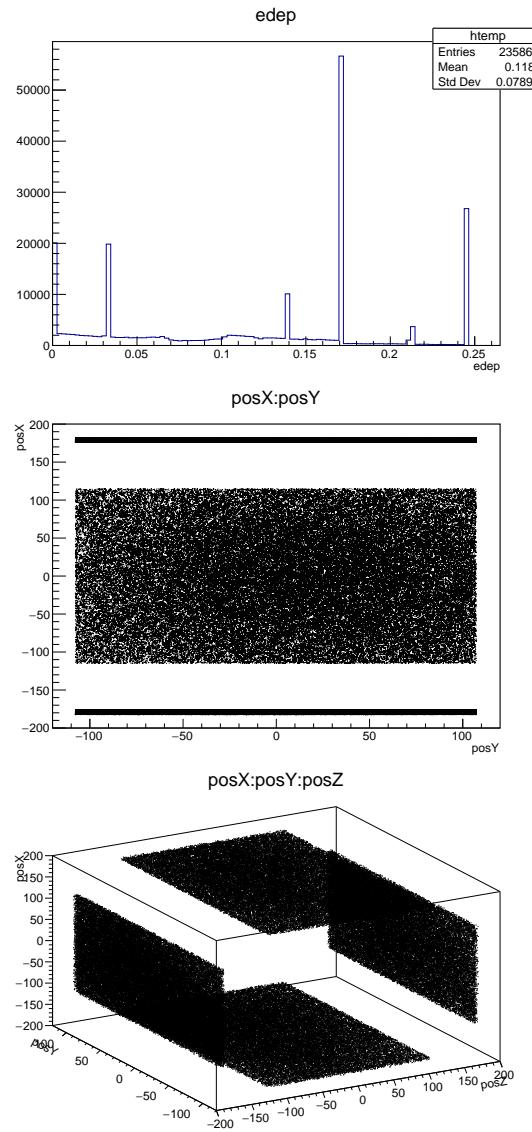


Figure 3: Output of Draw command

## 3.1 Colors, markers, lines

### 3.1.1 Color

It is possible to change the colors, line and marker styles and size. To change the color one can use `TTree ->SetMarkerColor(n_of_color)` or `TTree ->SetLineColor(n_of_color)`. The `n_of_color` can be chosen from table in Figure 4, for more information check <https://root.cern.ch/doc/v606/classTColor.html>



Figure 4: Basic ROOT colors used as a `Draw()` option. For more information check: <https://root.cern.ch/doc/v606/classTColor.html>

The example of usage is in the following command and in Figure 6:

```
root[4] Hits->SetLineColor(2)  
root[5] Hits->Draw("edep")
```

or to change marker color one can do:

```
root[4] Hits->SetMarkerColor(2)  
root[5] Hits->Draw("edep")
```

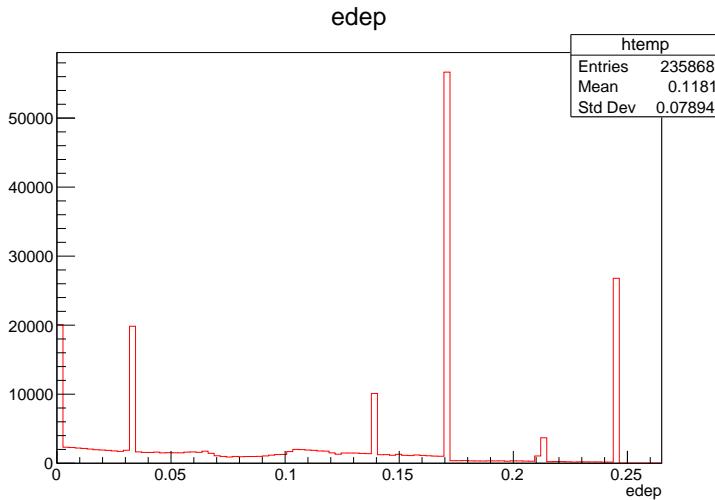


Figure 5: Drawing of the 1D histogram with a color option

### 3.1.2 Size

It is also possible to change size of lines with `TTree ->SetLineWidth(size)`:

```
root[4] Hits->SetLineWidth(3)
root[5] Hits->Draw("edep")
```

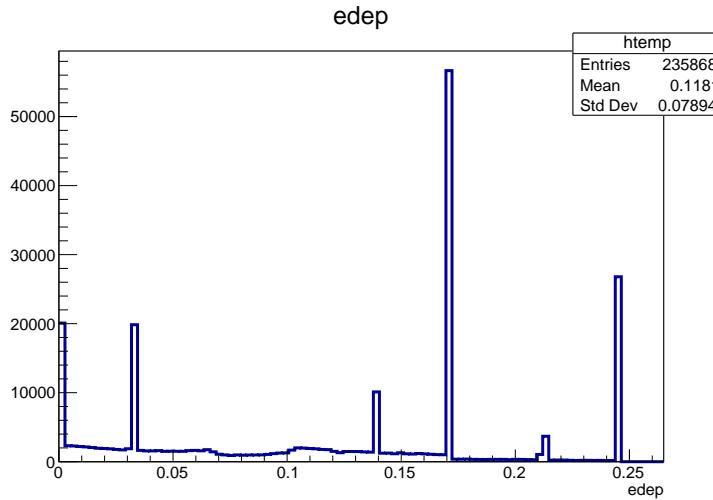


Figure 6: Drawing of the 1D histogram with a LineWidth option

### 3.1.3 Style

It is also possible to change style of lines and markers with `TTree ->SetLineStyle(n_of_style)` and `TTree ->SetLineStyle(n_of_style)`. The `n_of_style` can be chosen from table in Figure 7 and in Figure 8, for more information check  
<https://root.cern.ch/doc/v606/classTAttLine.html> and  
<https://root.cern.ch/root/html534/TAttMarker.html>

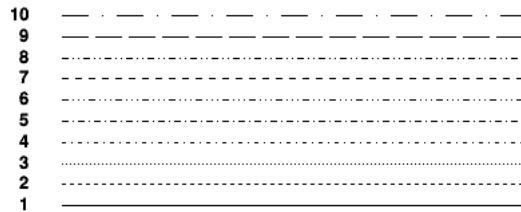


Figure 7: Basic ROOT line styles options. For more information check:  
<https://root.cern.ch/doc/v606/classTAttLine.html>

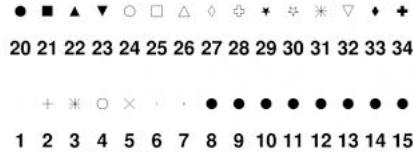


Figure 8: Basic ROOT marker styles options. For more information check: <https://root.cern.ch/root/html534/TAttMarker.html>

An example how to use these options is in the following and the results are illustrated in Figure 9 and in Figure 10:

```
root[4] Hits->SetLineStyle(4)
root[5] Hits->Draw("edep")
```

and

```
root[4] Hits->SetMarkerStyle(4)
root[5] Hits->Draw("time:trackLength")
```

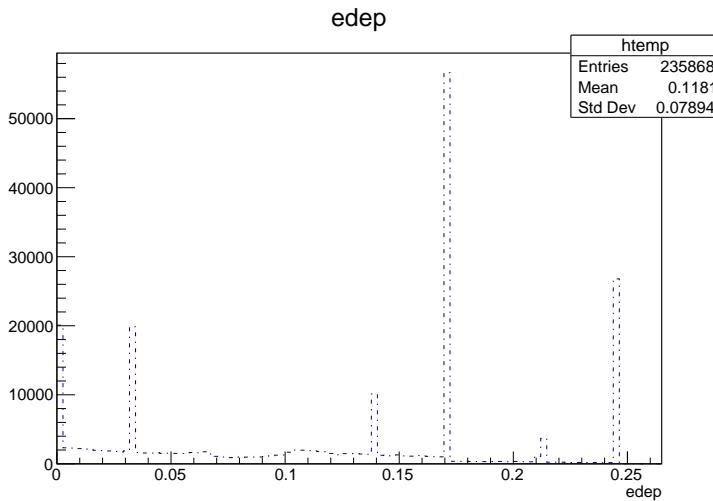


Figure 9: Drawing of the 1D histogram with a LineStyle option

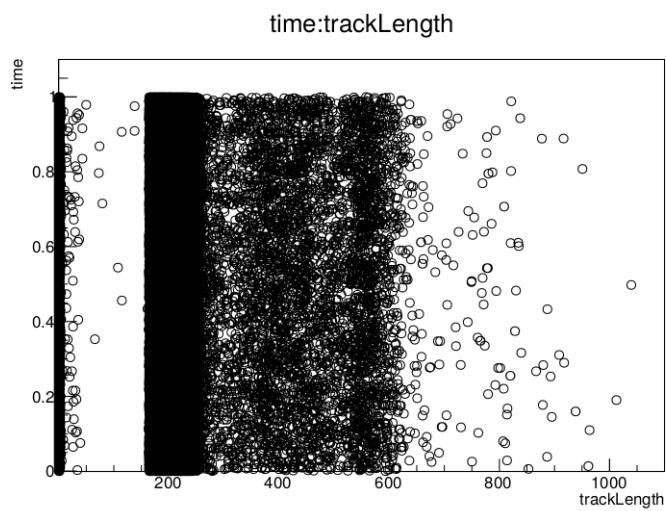


Figure 10: Drawing of the 2D histogram with a MarkerStyle option

## 3.2 How to apply a simple selection

It is also possible to apply a selection and after plot your variable. To do so one should use second option of the Draw() function: `TTree ->Draw("name_of_the_branch","selection")`. The several examples of cuts are presented here and plot the effect of them on `edep` variable:

```
root[4] Hits->Draw("edep","trackID==1")
root[5] Hits->Draw("edep","posX>0.5")
root[6] Hits->Draw("edep","localPosX>=0.5")
root[7] Hits->Draw("edep","processName=="phot") (variable is a string)
```

Or even more complicated selections like:

```
root[8] Hits->Draw("edep","trackID==1 && posX>0.5")
root[9] Hits->Draw("edep","trackID==1 || posX>0.5")
```

## 3.3 How to rebin a histogram

The default binning of 1D histogram in ROOT is 100 bins per X axis. It is possible to change and the simplest way is the following:

One has to defined a histogram with needed parameters:

```
root[1] TH1F* h = new TH1F("h","h",50,0,0.3)
```

Here we have a histogram with 50 bins in a range from 0 to 0.3.

Next step is to plot your histogram using the defined histogram "h" (Figure 11):

```
root[2] Hits->Draw("edep>>h","","")
```

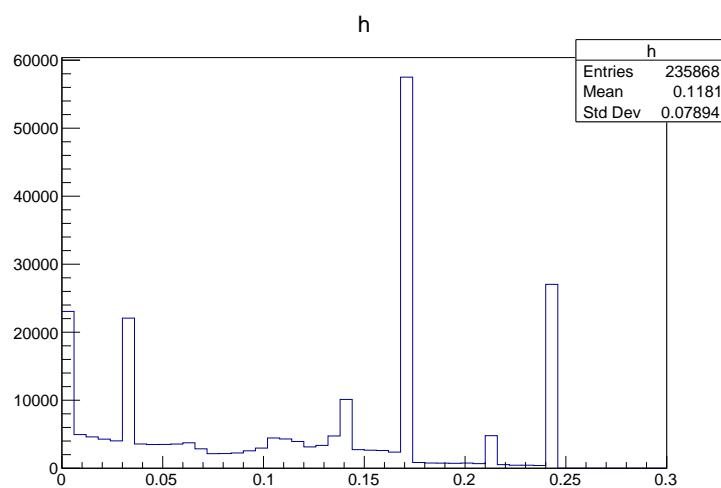


Figure 11: Drawing of rebinned 1D histogram

### 3.4 Superposition of two distributions

To superimpose two or more histograms one should write word "same" in the third option of the Draw() function: `TTree ->Draw("name_of_the_branch","selection","same")`. For example, to obtain a histogram like in Figure 12 one can do:

```
root[4] Hits->Draw("edep", "")
root[5] Hits->SetLineColor(2)
root[6] Hits->Draw("edep","processName==\"phot\"", "same")
```

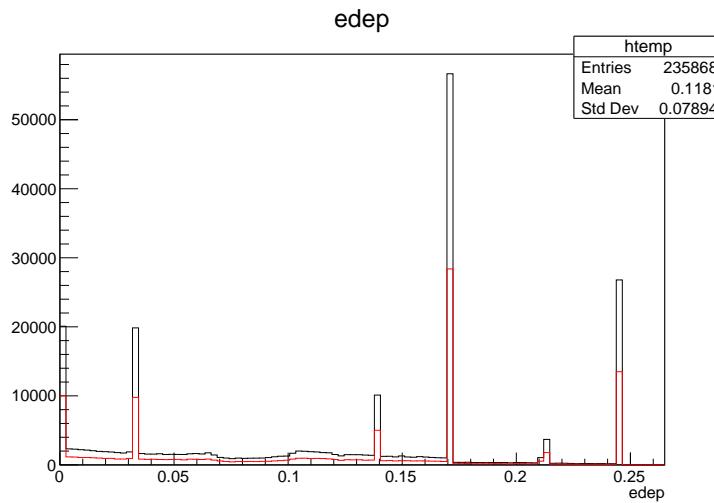


Figure 12: Drawing of two superposed the 1D histogram: one without any selection (black) and with a cut (red)

There is also a possibility to use "`esame`" option for the fast comparison of two histograms. Actually, this option plot error bars ("`e`" stands for that) but I would not recommend to rely on it and **calculate and plot your own error bars**. However, for a fast and easy superposition of two histograms without changing of line color it works perfectly (Figure 13):

```
root[4] Singles->Draw("energy")
root[5] Singles->Draw("energy","time<0.3", "esame")
```

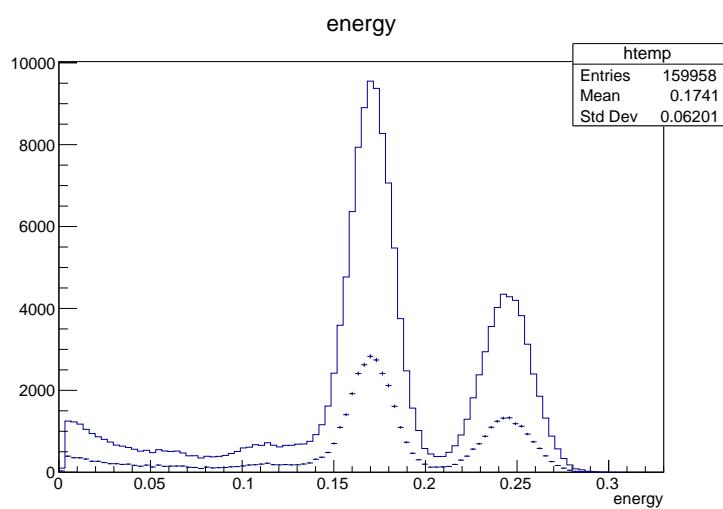


Figure 13: Drawing of two superposed the 1D histogram with "esame" option

### 3.5 Superposition of two distributions from different ROOT files

Sometimes it is useful to plot a superposition of two histograms from different ROOT files. In this case on start by opening the ROOT file 1:

```
> root -l YourOutputFile.root
```

or by an equivalent command directly in ROOT:

```
root[0] TFile *_file0 = TFile::Open("YourOutputFile.root");
```

and drawing a histogram:

```
root[2] Singles->Draw("energy")
```

The next step is to open the second file by:

```
root[3] TFile *_file1 = TFile::Open("YourOutputFile1.root");
```

and draw the new histogram with "same" or "esame" option:

```
root[4] Singles->Draw("energy","","esame")
```

### 3.6 Simple fitting

It is possible to do a simple fit of your distribution. For this after drawing your histogram you need to go to the canvas window and select **Tools --> Fit Panel**. You will have new window with a Fit Panel as presented in Figure 14. There you can choose your fitting function, fitting method, fitting range etc. In the following example, the fit is done by Gaussian function in the range [0.13;0.21] MeV. The parameters of the fit is printed in the terminal window where you started ROOT and looks like following:

FCN=1259.74 FROM MIGRAD STATUS=CONVERGED			75 CALLS	76 TOTAL
EDM=1.82902e-12 STRATEGY= 1			ERROR MATRIX ACCURATE	
EXT PARAMETER			STEP	FIRST
NO.	NAME	VALUE	SIZE	DERIVATIVE
1	Constant	9.05059e+03	4.05523e+01	5.29487e-01
2	Mean	1.70049e-01	4.42797e-05	7.60710e-07
3	Sigma	1.28126e-02	3.88904e-05	1.30065e-05

More about the fitting options can be found on ROOT web page.

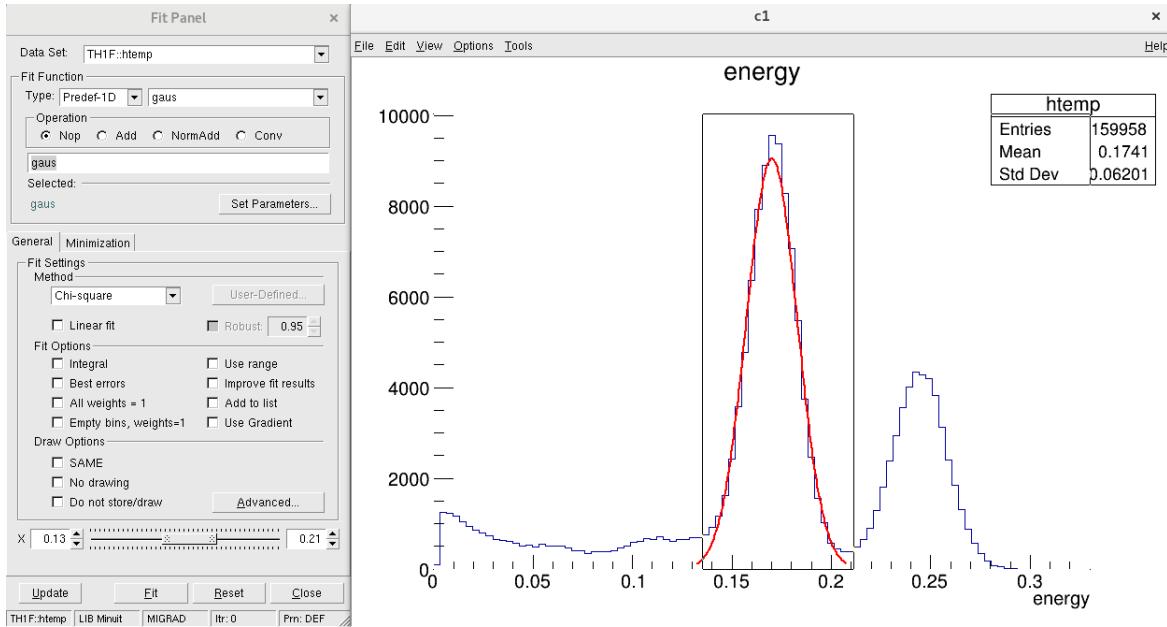


Figure 14: Fit panel and an example of fitted histogram

## 4 Scan option

Sometimes it is useful to printout the content of a branch, for example, in case when you want to study some anomalies. For this purpose one can use `Scan()` function. It is similar to `Draw()`, i.e. you also can apply cuts and plot several variables. For example, if one does:

```
root[4] Hits->Scan("edep:posX","time<0.1&&posY>0")
```

the output will be something like this:

```
*****
*   Row    *      edep *      posX *
*****
*       4 *  0.0198734 * 25.182537 *
*       5 *          0 * 25.227350 *
*      11 *          0 * -180.9978 *
*      14 *  0.2450000 * 104.99372 *
*      15 *  0.1710000 * -91.10610 *
*      18 *  0.2199750 * 58.401462 *
*      21 *  0.2450000 * -181.6459 *
*      24 *  0.0803248 * 176.01010 *
*      27 *  0.1520019 * -38.77223 *
*      29 *  0.2450000 * 34.775840 *
*      30 *  0.0811032 * -176.0955 *
```

```
*      33 * 0.1103092 * -69.64908 *
*      34 * 0.0957561 * -68.88422 *
*      38 * 0.2450000 * 21.076614 *
*      42 * 0.1710000 * -178.0622 *
*      43 * 0.1710000 * -1.295397 *
*      44 * 0.1387058 * 178.04272 *
*      45 * 0.0322941 * 178.14222 *
*      48 * 0.2450000 * -177.6772 *
*      54 * 0.2450000 * -105.5232 *
*      55 * 0.0675205 * -176.8821 *
*      56 * 0.0562481 * 7.8245472 *
*      57 * 0.1557220 * 7.5984726 *
*      58 * 0.0330297 * 7.5733656 *
*      60 * 0.1710000 * -47.08741 *
```

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

There are only first 25 lines in the output, one can press either "Enter" to continue or "q" to exit.

## 5 Simplest script to draw with ROOT

Sometimes it is useful to group the previous commands in a small script. Let's call it `my_script.C`, in order to run it one can do:

```
> root -l my_script.C
```

The `my_script.C` should contain something like this:

```
{  
    TFile *_file0 = TFile::Open("YourOutputFile.root");  
    Singles->Draw("energy");  
    Singles->Draw("energy","time<0.3","esame");  
},
```

where you open file at the beginning and after you can copy you plotting commands.

## 6 Example of script making beautiful plots

I hear all the time a lot of complains that the ROOT plots are not beautiful. Thus, in this section I propose you a template to start for your plots. To run the script one should do:

```
> root -l my_script.C
```

The name of the main function in `my_script.C` must be the same as the name of your script file containing the following:

```
void my_script() {  
    //remove the stat from upper right corner  
    gStyle->SetOptStat(0);  
    //remove the title  
    gStyle->SetOptTitle(0);  
    //define fonts sizes  
    gStyle->SetTextSize(0.06);  
    gStyle->SetLabelSize(0.06,"x");  
    gStyle->SetLabelSize(0.06,"y");  
    gStyle->SetLabelSize(0.06,"z");  
    gStyle->SetTitleSize(0.06,"x");  
    gStyle->SetTitleSize(0.05,"y");  
    gStyle->SetTitleSize(0.06,"z");  
    //define number of divisions on any axis, here is done only for "Y"  
    gStyle->SetNdivisions(505,"y");  
    gStyle->SetLineWidth(3);  
  
    //define your ROOT file name  
    TString filename = "YourOutputFile.root";
```

```

// open the file
TFile *f = TFile::Open(filename);
// get the TTree
TTree *Tree = (TTree*)f->Get("Singles");

// define the variable(s) of interest, type of variable must be respected
Float_t energy;

// define the branch(s) of interest
 TBranch *benergy;

// Set branch address
Tree->SetBranchAddress( "energy", &energy, &benergy);

// Get number of events in the TTree
const int n=(const int)Tree->GetEntries();

// Define a histogram with 100 bins, on x from 0 to 0.3
TH1F* h = new TH1F("h","h",100,0,0.3);

// Loop over events
for(int i=0;i<n;i++)
{
    //get event i
    Tree->GetEntry(i);
    benergy->GetEntry(i);
    //print out the values
    cout<<i<<" "<<energy<<endl;
    //fill histogram
    h->Fill(energy);
}

//Define canvas
TCanvas *can = new TCanvas("can","can",600,800);

//Define paramters of the canvas
can->SetFillColor(0);
can->SetBorderMode(0);
can->SetBorderSize(3);
can->SetBottomMargin(0.14);
can->SetLeftMargin(0.16);
can->SetFrameBorderMode(0);
can->SetFrameLineWidth(3);

```

```

can->SetFrameBorderMode(0);

//Define parameters of the histogram
h->SetLineWidth(2);
h->SetLineColor(2);
h->GetXaxis()->SetTitle("Energy, MeV");
h->GetYaxis()->SetTitleOffset(1.6);
h->GetYaxis()->SetTitle("Arbitrary units");
//Draw histogram
h->Draw();
//Save cancas as .pdf
can->SaveAs("energy.pdf");
}

```

This script could also be found here.

The output result is shown in Figure 15.

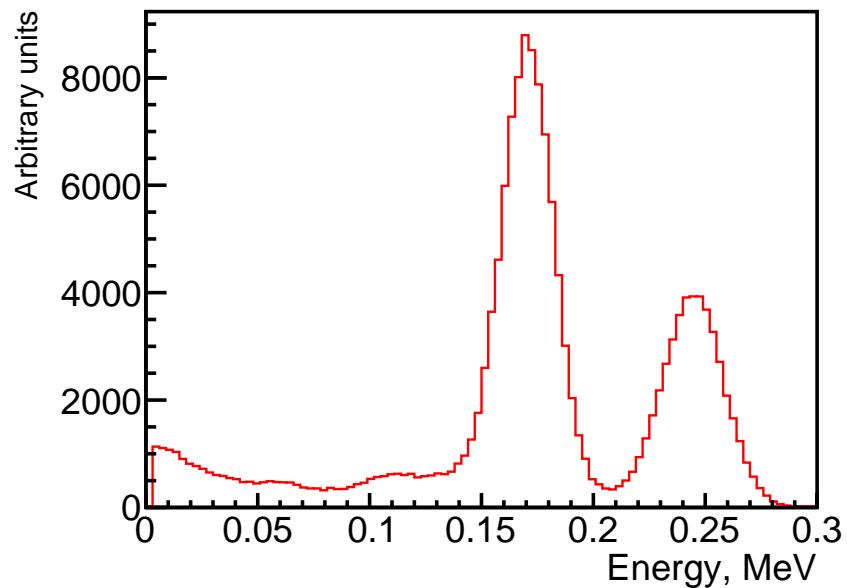


Figure 15: The output of the example of script making beautiful plots.

## **7 Adding a new variable to your TTree\***

In this section I will just add a link to the script that indicates how to add a new variable. It is a script that adds a shift on energy which is different for each of four detection heads.

One can find it here: AddShiftedEnergy.C.