Using Root for Evaluating LAT Performance

S. Ritz

1 October 2001

NOTE: examples given here are just examples. Don't use these as documentation of any selections!

First, some history and observations

- I decided to use ROOT to do the analysis for the AO response (someone needed to try it out). This was a significant time investment. Operate on tuples.
- ROOT is enormously powerful w/ good documentation and a large and growing user base.
- ROOT is a *toolkit*, **NOT** an analysis platform. We must still make a user platform!
- ROOT sometimes handles errors badly (or sometimes doesn't even tell you!)

My Modes of Using of ROOT and Related Utilities

- interactive quick look at data, tryout ideas
- captured analysis
- full analysis

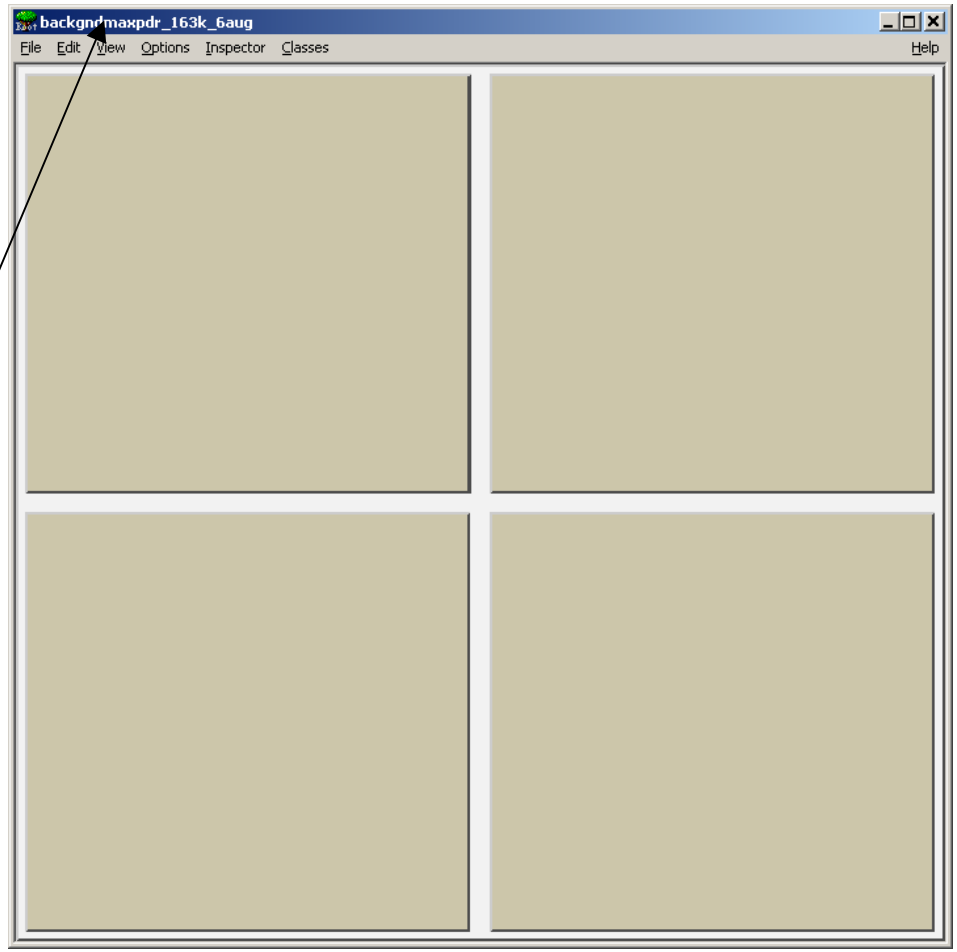
Interactive Mode

- **Macro for setup** `root[0] .x setupb.c`

```
{
//setup the canvas and read in the file
TCanvas *c1 = new TCanvas("c1","backgndmaxpdr_163k_6aug",680,0,720,720);
c1->Divide(2,2,.01,.01,21); // create 4 pads
// grab the file
f1 = new TFile("backgndmaxpdr_163k_6aug.root");
TTree *t1=(TTree*)f1->Get("PDR/t1");
//
// Change default style for the statistics box
gStyle->SetStatW(0.30);
gStyle->SetStatH(0.20);
gStyle->SetStatColor(42);
gStyle->SetOptStat(10);
t1->SetLineWidth(4);
t1->SetMarkerStyle(4);
t1->SetMarkerSize(0.5);
t1->SetMarkerColor(4);
t1->SetFillColor(5);
//
//
c1_1->cd();
c1_1->SetGrid();
c1_2->SetGrid();
c1_3->SetGrid();
c1_4->SetGrid();
c1_1->SetLogy();
c1_2->SetLogy();
c1_3->SetLogy();
c1_4->SetLogy();
//c1_1->SetLogx();
//c1_2->SetLogx();
//c1_3->SetLogx();
//c1_4->SetLogx();
c1->Update();
//
}
```

(Is there a ready-made way to title the canvas with the filename??)

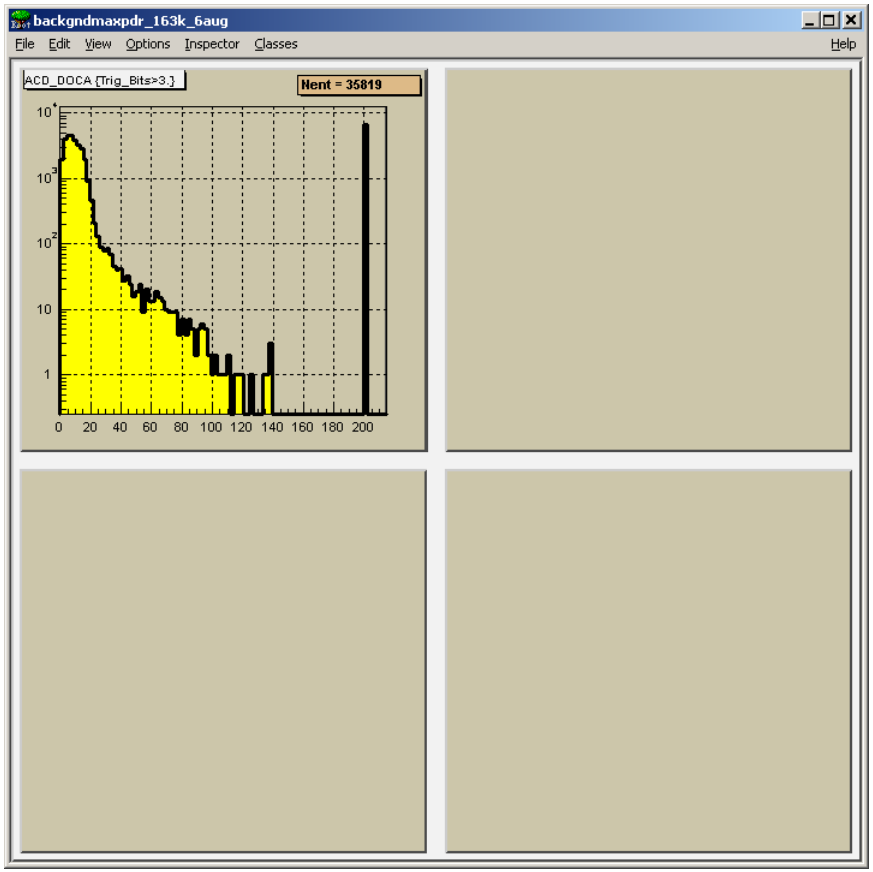
Note the subdirectory!!



Interactive Mode (2)

- Plots using TTree->Draw() with cuts, eg.,

```
t1->Draw("ACD_DOCA","Trig_Bits>3.");
```



Hint: move around pads in two ways:

1) center-click with mouse
2) command line, e.g.,

```
c1_2->cd();
```

can drag out pads to make larger/smaller/reposition

Interactive Mode (3)

- This gets cumbersome quickly, as the number of cuts grows. Also, BEWARE, there is some undefined limit to the length of the command line – it never fails outright, it just gets flaky!
- Two approaches: (1) another setup file to define cuts, or (2) eventlists

Exmaple: Define Cuts File (.x defcuts.c)

```
{
TCut L1V = "((vetoword==0||vetoword>127.)&&(ntothit-nhitsiderow3-
   nhitsiderow2)<3.)||(Trig_Bits&16)";
TCut L1T = "Trig_Bits>3.";
TCut L2T = "(TKR_No_Tracks>0.&&ACD_DOCA>25.)||(Trig_Bits&16)";
TCut tracks = "TKR_No_Tracks>0.";
}
```

Then, give the command

```
t1->Draw("ACD_DOCA",L1T&&L1V&&L2T);
```

Note that you can mix selections, such as

```
t1->Draw("ACD_DOCA","(Trig_Bits&8)&&!(Trig_Bits&4)"&&L1T&&L2T&&L1V);
```

BUT the opposite order doesn't work for some reason!

Also, the command character limit seems to apply to TCuts as well, so beware.

Using Event Lists

- Tell ROOT only to use a subset of the events. Speeds processing, and reduces typing.

```
{
t1->Draw(">>glist2","Trig_Bits>3&&(ACD_DOCA>25.&&TKR_No_Tracks>0&&(ntothit-
    nhitsiderow3-nhitsiderow2)<3)||(Trig_Bits&16)");
TEventList *listL22 = (TEventList*)gDirectory->Get("glist2");
t1->SetEventList(listL22);
}
```

All additional Draw commands will display only the subset of events that are in the eventlist sastifying these cuts.

Captured Analysis

- It soon becomes important to capture the selections and plot sequences into a macro. Place all the interactive commands into macros to make well-defined sets of plots that can be saved in a logbook with the macro. **WOULD BE NICE TO HAVE A MATHCAD- or HIPPO-STYLE "LIVING DOCUMENT" to capture the analysis and the rationale. This document would integrate the macros and the resulting plots.**

Full Analysis

- The final stage is a full analysis, using TTree->MakeClass(); This produces a .h and .c file with useful methods such as Loop. More sophisticated analyses can be done here, including branches and detailed numerical calculations that are not appropriate for command-line type formats.
- **BEWARE:** you have to edit the .h file produced by MakeClass to point into our PDR/t1 subdirectory. Also, **beware** that eventlists defined on the command line are NOT operative in these classes because the .h reads in the ROOT tuple file fresh. And **REALLY BEWARE:** **unlike with macros, you can't just reuse these classes on other ROOT tuple files. The .h file explicitly opens the ROOT file used to make the class. You can start a new analysis session with a different ROOT file on the command line, load the class you made, and when you run the loop you'll be operating on the first ROOT file! No warning, no hint, no nothing! Once understood, it makes sense, but…**

Example: Event Selections

```
#define genlist2_cxx
#include "genlist2.h"
#include "TH2.h"
#include "TStyle.h"
#include "TCanvas.h"
void genlist2::Loop()
{
//   In a Root session, you can do:
//      Root > .L genlist2.C
//      Root > genlist2 t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(i);  // read all branches
//by  b_branchname->GetEntry(i); //read only this branch
   if (fChain == 0) return;

   Int_t nentries = Int_t(fChain->GetEntries());
   Int_t imax=1000;
   Int_t nbytes = 0, nb = 0, iout = 0;
   printf("Run_Number  Event_ID\n");
   for (Int_t jentry=0; (jentry<nentries&&iout<imax);jentry++) {
      Int_t ientry = LoadTree(jentry); //in case of a TChain, ientry is the entry number in the current file
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      if (iout<imax&&genlist2::Trig_Bits>3.&&((genlist2::ACD_DOCA>25.&&genlist2::TKR_No_Tracks>0.&&((genlist2::ntothit-
      genlist2::nhitsiderow3-genlist2::nhitsiderow2)<3.))||(genlist2::Trig_Bits&16))){
        printf("   %i          %i   \n",genlist2::Run_Number,genlist2::Event_ID);
        iout++;
        }
   }
   printf("Total number of events: %i\n",iout);
}
```

This produces a list of run and event numbers for events that satisfy the selections. I send this list to Karl, who has a script that automatically extracts these events from the irfs. Critical for analysis of the 50M event files – can examine these in the single-event displays.

Note how to access tuple variables