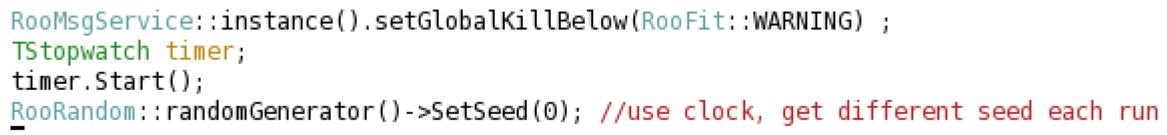
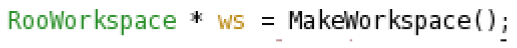
PLR Code

SIRun4.cxx



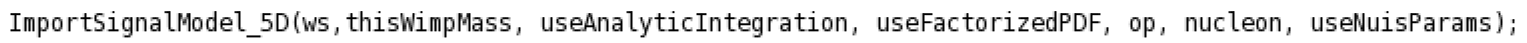
```
RooMsgService::instance().setGlobalKillBelow(RooFit::WARNING) ;
TStopwatch timer;
timer.Start();
RooRandom::randomGenerator()->SetSeed(0); //use clock, get different seed each run
```

- Pretty self explanatory
- Sets up a timer to time the process
- kill warning messages
- Make sure each run uses a different random seed



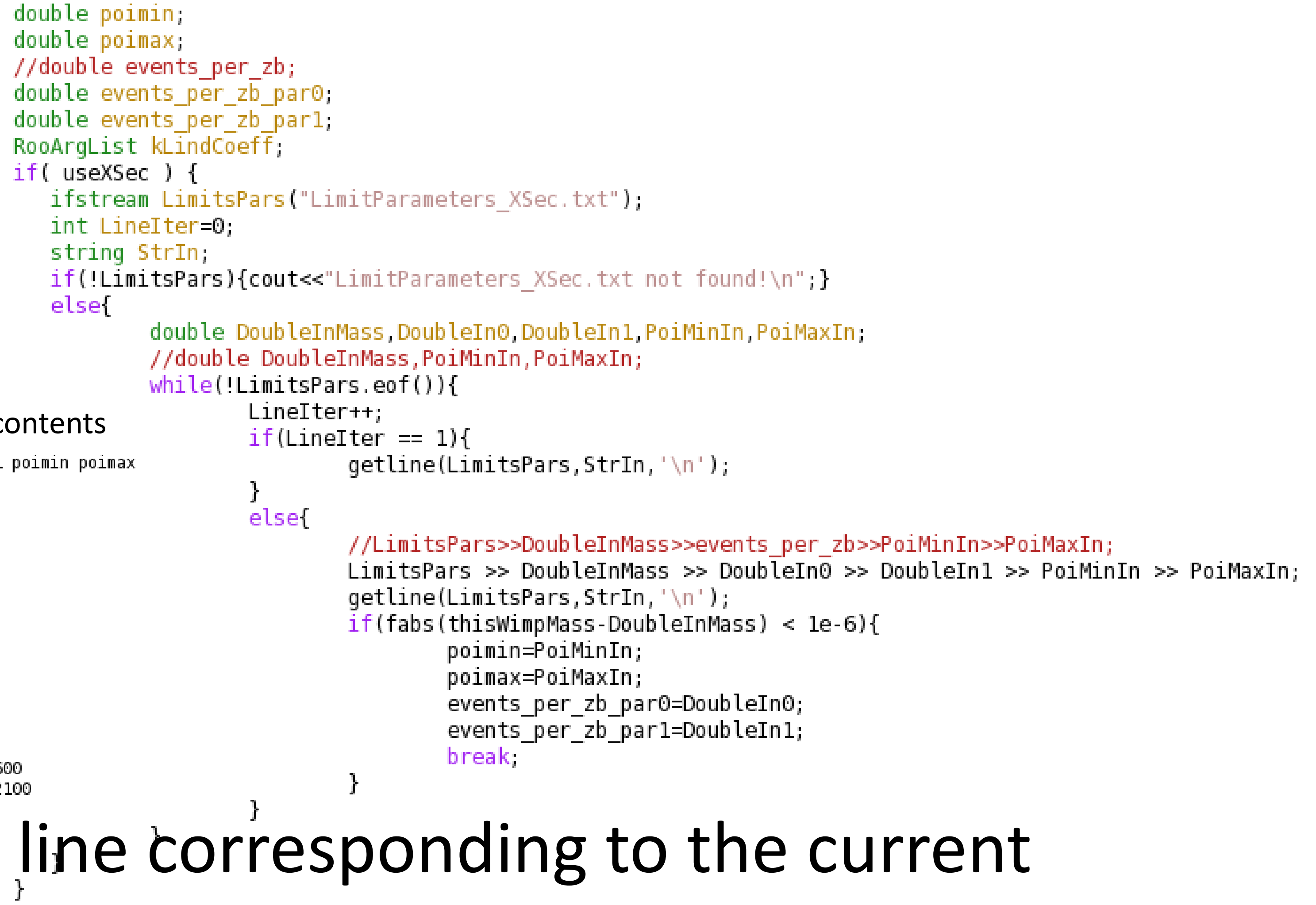
```
RooWorkspace * ws = MakeWorkspace();
```

- Creates the workspace using the MakeWorkspace method from “MakeWorkspace.h”
  - Declares the RooRealVars r, S1, log10S2, phi, drift, mWimp, and xsec and imports into workspace.
  - Creates timeBin categories (whatever that means) and defines exposure for each.
  - Returns the workspace



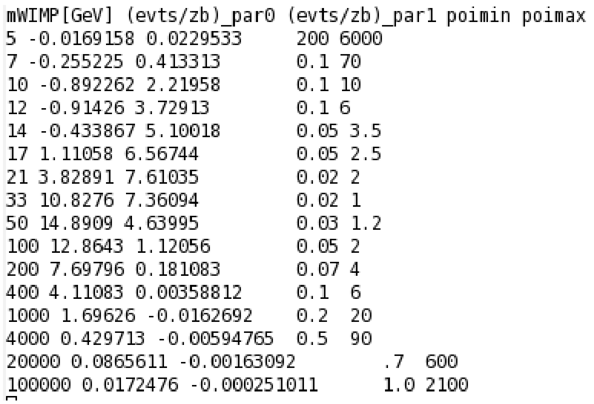
```
ImportSignalModel_5D(ws,thisWimpMass, useAnalyticIntegration, useFactorizedPDF, op, nucleon, useNuisParams);
```

- Uses “ImportSignalModel_5D” from “ImportSignalModel.h” to import the signal PDFs nrPop_z[z-slice]_t[time bin] (RooProdPDF) into the workspace.



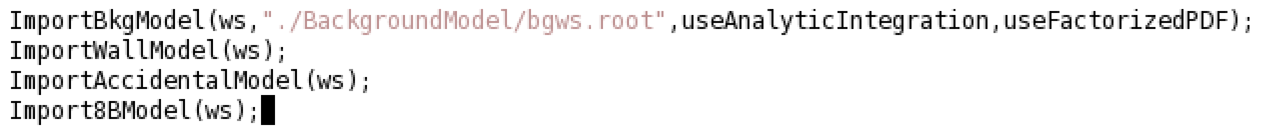
```
double poimin;
double poimax;
//double events_per_zb;
double events_per_zb_par0;
double events_per_zb_par1;
RooArgList kLindCoeff;
if( useXSec ) {
   ifstream LimitsPars("LimitParameters_XSec.txt");
   int LineIter=0;
   string StrIn;
   if(!LimitsPars){cout<<"LimitParameters_XSec.txt not found!\n";}
   else{
           double DoubleInMass,DoubleIn0,DoubleIn1,PoiMinIn,PoiMaxIn;
           //double DoubleInMass,PoiMinIn,PoiMaxIn;
           while(!LimitsPars.eof()){
                   LineIter++;
                   if(LineIter == 1){
                           getline(LimitsPars,StrIn,'\n');
                   }
                   else{
                           //LimitsPars>>DoubleInMass>>events_per_zb>>PoiMinIn>>PoiMaxIn;
                           LimitsPars >> DoubleInMass >> DoubleIn0 >> DoubleIn1 >> PoiMinIn >> PoiMaxIn;
                           getline(LimitsPars,StrIn,'\n');
                           if(fabs(thisWimpMass-DoubleInMass) < 1e-6){
                                   poimin=PoiMinIn;
                                   poimax=PoiMaxIn;
                                   events_per_zb_par0=DoubleIn0;
                                   events_per_zb_par1=DoubleIn1;
                                   break;
                           }
                   }
           }
   }
}
```

LimitParameters_XSec.txt contents



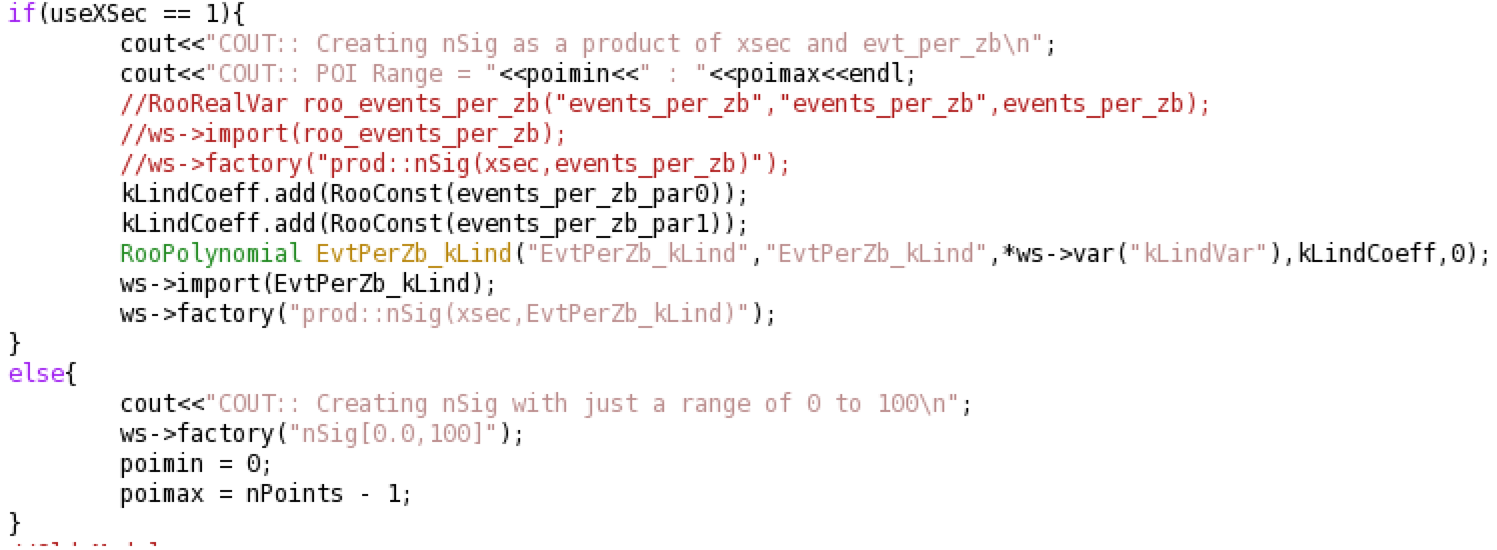
```
mWIMP[GeV] (evts/zb)_par0 (evts/zb)_par1 poimin poimax
5 -0.0169158 0.0229533     200 6000
7 -0.255225 0.413313       0.1 70
10 -0.892262 2.21958       0.1 10
12 -0.91426 3.72913        0.1 6
14 -0.433867 5.10018       0.05 3.5
17 1.11058 6.56744         0.05 2.5
21 3.82891 7.61035         0.02 2
33 10.8276 7.36094         0.02 1
50 14.8909 4.63995         0.03 1.2
100 12.8643 1.12056        0.05 2
200 7.69796 0.181083       0.07 4
400 4.11083 0.00358812     0.1  6
1000 1.69626 -0.0162692    0.2  20
4000 0.429713 -0.00594765  0.5  90
20000 0.0865611 -0.00163092       .7  600
100000 0.0172476 -0.000251011     1.0 2100
```

- Reads the line corresponding to the current WIMP-mass from the “LimitParameters_XSec.txt” text file and sets the poi variables poimin, poimax, etc. accordingly



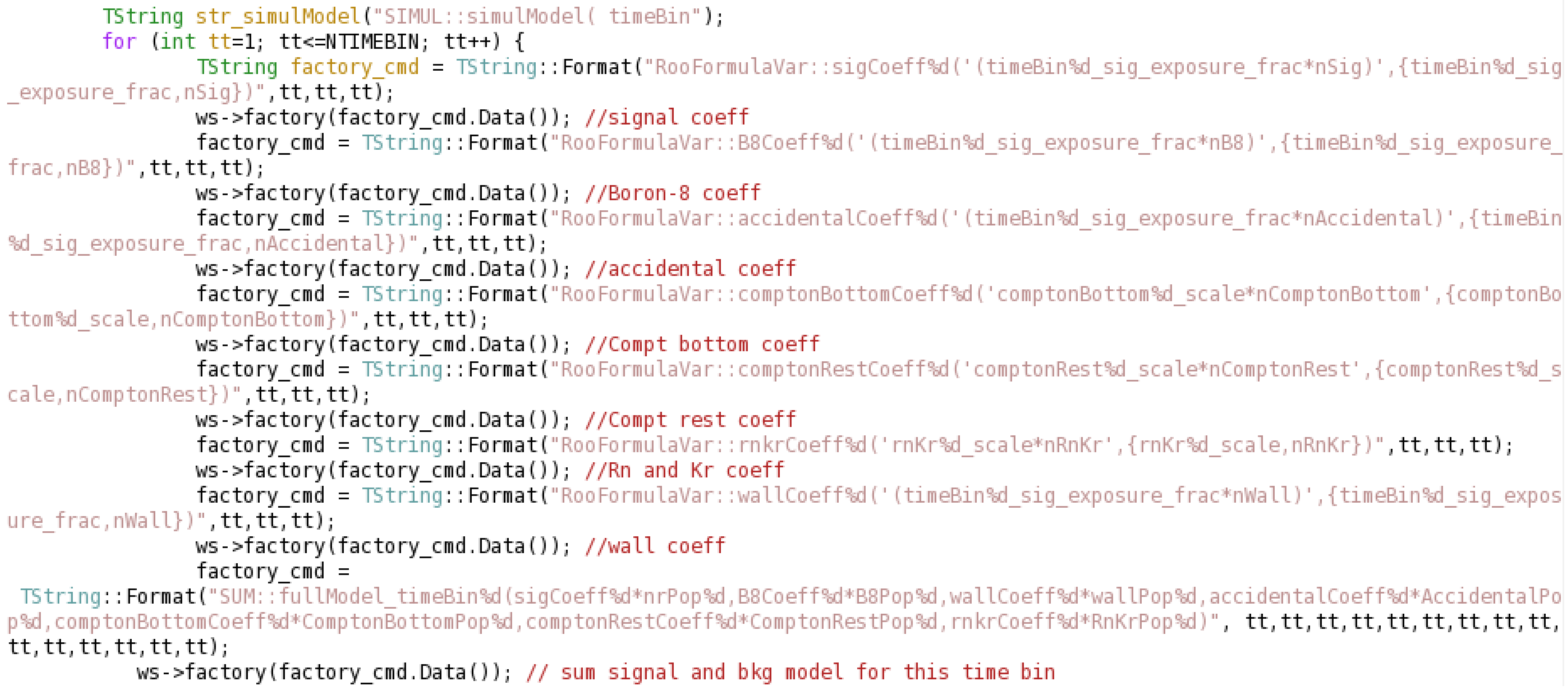
```
ImportBkgModel(ws,"./BackgroundModel/bgws.root",useAnalyticIntegration,useFactorizedPDF);
ImportWallModel(ws);
ImportAccidentalModel(ws);
Import8BModel(ws);
```

- Imports the various background PDFs into the workspace. (I think, haven’t looked too much into this chunk)



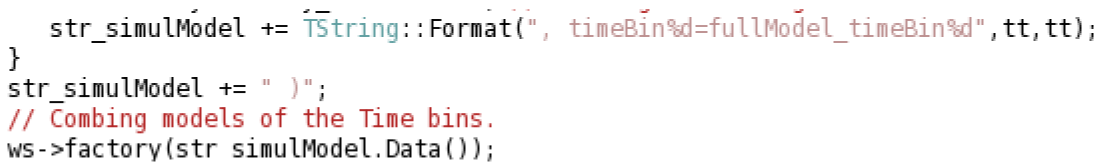
```
if(useXSec == 1){
        cout<<"COUT:: Creating nSig as a product of xsec and evt_per_zb\n";
        cout<<"COUT:: POI Range = "<<poimin<<" : "<<poimax<<endl;
        //RooRealVar roo_events_per_zb("events_per_zb","events_per_zb",events_per_zb);
        //ws->import(roo_events_per_zb);
        //ws->factory("prod::nSig(xsec,events_per_zb)");
        kLindCoeff.add(RooConst(events_per_zb_par0));
        kLindCoeff.add(RooConst(events_per_zb_par1));
        RooPolynomial EvtPerZb_kLind("EvtPerZb_kLind","EvtPerZb_kLind",*ws->var("kLindVar"),kLindCoeff,0);
        ws->import(EvtPerZb_kLind);
        ws->factory("prod::nSig(xsec,EvtPerZb_kLind)");
}
else{
        cout<<"COUT:: Creating nSig with just a range of 0 to 100\n";
        ws->factory("nSig[0.0,100]");
        poimin = 0;
        poimax = nPoints - 1;
}
```

- Creates a new RooRealVar “nSig”
  - If useXSec is true, the poi variables from earlier are used to calculate the value of nSig
  - If false, nSig is just something between 0 and 100



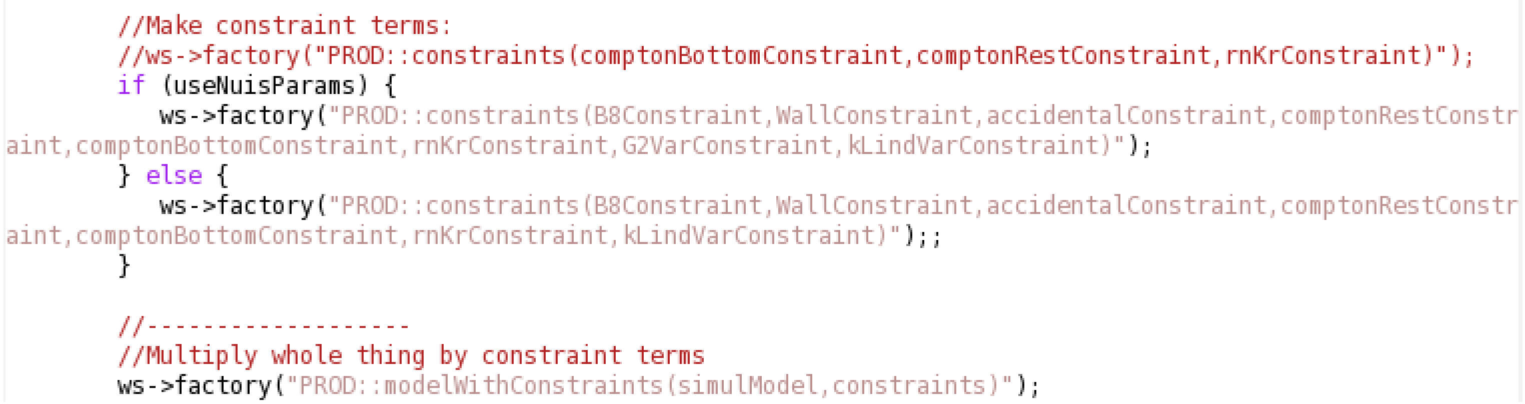
```
        TString str_simulModel("SIMUL::simulModel( timeBin");
        for (int tt=1; tt<=NTIMEBIN; tt++) {
                TString factory_cmd = TString::Format("RooFormulaVar::sigCoeff%d('(timeBin%d_sig_exposure_frac*nSig)',{timeBin%d_sig
_exposure_frac,nSig})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //signal coeff
                factory_cmd = TString::Format("RooFormulaVar::B8Coeff%d('(timeBin%d_sig_exposure_frac*nB8)',{timeBin%d_sig_exposure_
frac,nB8})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //Boron-8 coeff
                factory_cmd = TString::Format("RooFormulaVar::accidentalCoeff%d('(timeBin%d_sig_exposure_frac*nAccidental)',{timeBin
%d_sig_exposure_frac,nAccidental})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //accidental coeff
                factory_cmd = TString::Format("RooFormulaVar::comptonBottomCoeff%d('comptonBottom%d_scale*nComptonBottom',{comptonBo
ttom%d_scale,nComptonBottom})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //Compt bottom coeff
                factory_cmd = TString::Format("RooFormulaVar::comptonRestCoeff%d('comptonRest%d_scale*nComptonRest',{comptonRest%d_s
cale,nComptonRest})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //Compt rest coeff
                factory_cmd = TString::Format("RooFormulaVar::rnkrCoeff%d('rnKr%d_scale*nRnKr',{rnKr%d_scale,nRnKr})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //Rn and Kr coeff
                factory_cmd = TString::Format("RooFormulaVar::wallCoeff%d('(timeBin%d_sig_exposure_frac*nWall)',{timeBin%d_sig_expos
ure_frac,nWall})",tt,tt,tt);
                ws->factory(factory_cmd.Data()); //wall coeff
                factory_cmd =
 TString::Format("SUM::fullModel_timeBin%d(sigCoeff%d*nrPop%d,B8Coeff%d*B8Pop%d,wallCoeff%d*wallPop%d,accidentalCoeff%d*AccidentalPo
p%d,comptonBottomCoeff%d*ComptonBottomPop%d,comptonRestCoeff%d*ComptonRestPop%d,rnkrCoeff%d*RnKrPop%d)", tt,tt,tt,tt,tt,tt,tt,tt,tt,
tt,tt,tt,tt,tt,tt);
           ws->factory(factory_cmd.Data()); // sum signal and bkg model for this time bin
```

- Generating coefficients for each model and combining them together to form a jumbo PDF (I think) called “fullModel_timeBin[time bin]”.



```
    str_simulModel += TString::Format(", timeBin%d=fullModel_timeBin%d",tt,tt);
}
str_simulModel += " )";
// Combing models of the Time bins.
ws->factory(str_simulModel.Data());
```

- Together with the top line from the last slide creates a new SIMUL (whatever that is) from the four time bin models called “simulModel”
- This whole thing, though, would seem to be a one-dimensional PDF that tells you how many total events you should expect.

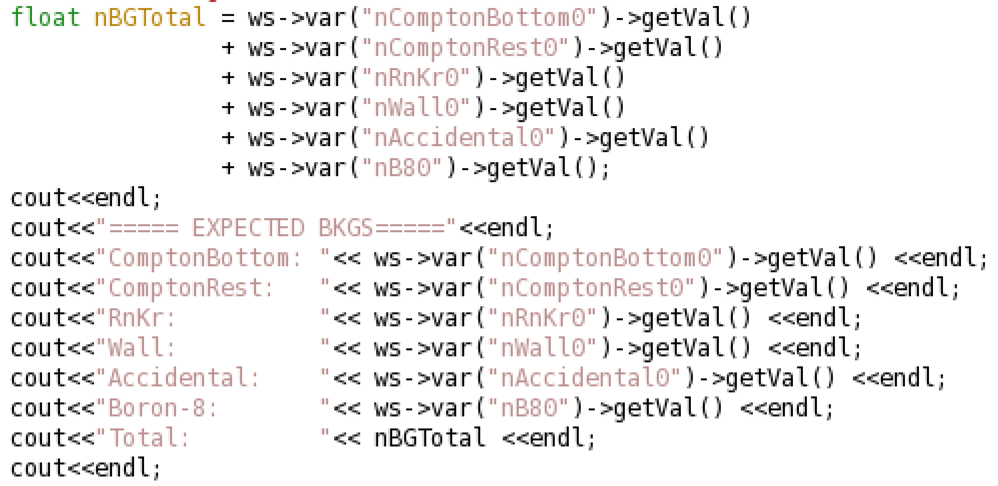


```
        //Make constraint terms:
        //ws->factory("PROD::constraints(comptonBottomConstraint,comptonRestConstraint,rnKrConstraint)");
        if (useNuisParams) {
           ws->factory("PROD::constraints(B8Constraint,WallConstraint,accidentalConstraint,comptonRestConstr
aint,comptonBottomConstraint,rnKrConstraint,G2VarConstraint,kLindVarConstraint)");
        } else {
           ws->factory("PROD::constraints(B8Constraint,WallConstraint,accidentalConstraint,comptonRestConstr
aint,comptonBottomConstraint,rnKrConstraint,kLindVarConstraint)");;
        }

        //-------------------
        //Multiply whole thing by constraint terms
        ws->factory("PROD::modelWithConstraints(simulModel,constraints)");
```

- Create a new PDF “constraints” which is the direct product of all of the nuisance parameter constraint pdfs. (8 or 9 dimensional) Also add it to the workspace.
- Create a new PDF “modelWithConstraints” that is a direct product of the simulModel (tells you how many total events to expect) and the constraints PDF.

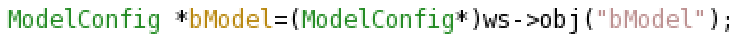
```
ConfigureModel(ws,useNuisParams,useXSec);
```

- Creates a RooStats::ModelConfig object so HypoTestInv demo knows what to do and imports it into the workspace.
- Defines sets of variables into observables, global observables, and nuisance parameters.
- Sets parameter of interest



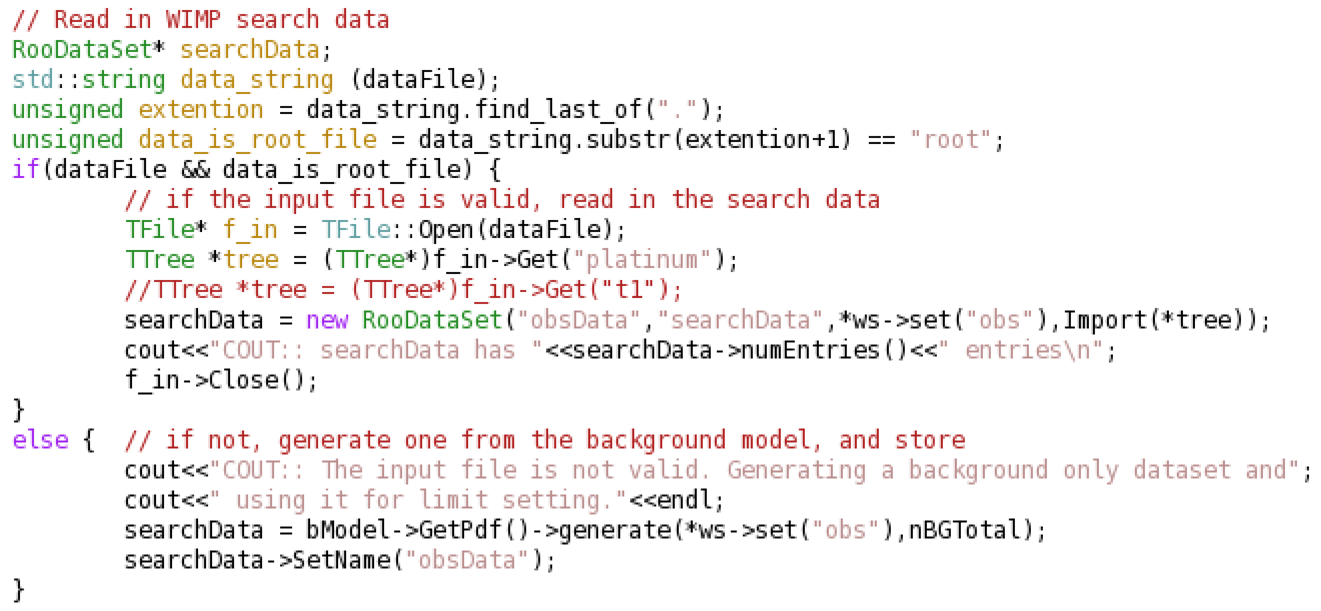
```
float nBGTotal = ws->var("nComptonBottom0")->getVal()
               + ws->var("nComptonRest0")->getVal()
               + ws->var("nRnKr0")->getVal()
               + ws->var("nWall0")->getVal()
               + ws->var("nAccidental0")->getVal()
               + ws->var("nB80")->getVal();
cout<<endl;
cout<<"===== EXPECTED BKGS====="<<endl;
cout<<"ComptonBottom: "<< ws->var("nComptonBottom0")->getVal() <<endl;
cout<<"ComptonRest:   "<< ws->var("nComptonRest0")->getVal() <<endl;
cout<<"RnKr:          "<< ws->var("nRnKr0")->getVal() <<endl;
cout<<"Wall:          "<< ws->var("nWall0")->getVal() <<endl;
cout<<"Accidental:    "<< ws->var("nAccidental0")->getVal() <<endl;
cout<<"Boron-8:       "<< ws->var("nB80")->getVal() <<endl;
cout<<"Total:         "<< nBGTotal <<endl;
cout<<endl;
```

- Calculates the total number of background events and prints the contributions.



```
ModelConfig *bModel=(ModelConfig*)ws->obj("bModel");
```

- Creates a ModelConfig object “bModel” from the one imported earlier into the workspace.

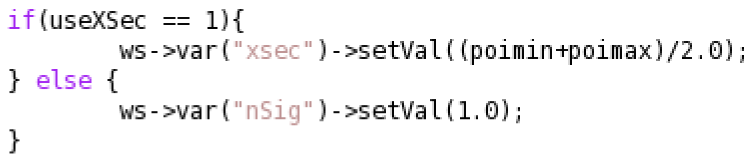


```
// Read in WIMP search data
RooDataSet* searchData;
std::string data_string (dataFile);
unsigned extention = data_string.find_last_of(".");
unsigned data_is_root_file = data_string.substr(extention+1) == "root";
if(dataFile && data_is_root_file) {
        // if the input file is valid, read in the search data
        TFile* f_in = TFile::Open(dataFile);
        TTree *tree = (TTree*)f_in->Get("platinum");
        //TTree *tree = (TTree*)f_in->Get("t1");
        searchData = new RooDataSet("obsData","searchData",*ws->set("obs"),Import(*tree));
        cout<<"COUT:: searchData has "<<searchData->numEntries()<<" entries\n";
        f_in->Close();
}
else {  // if not, generate one from the background model, and store
        cout<<"COUT:: The input file is not valid. Generating a background only dataset and";
        cout<<" using it for limit setting."<<endl;
        searchData = bModel->GetPdf()->generate(*ws->set("obs"),nBGTotal);
        searchData->SetName("obsData");
}
```

- Creates a RooDataSet “searchData” from data in a provided root file.
- If the provided file doesn’t have the .root extension a data set is generated from the background model. (Though I know from testing that this doesn’t work correctly)

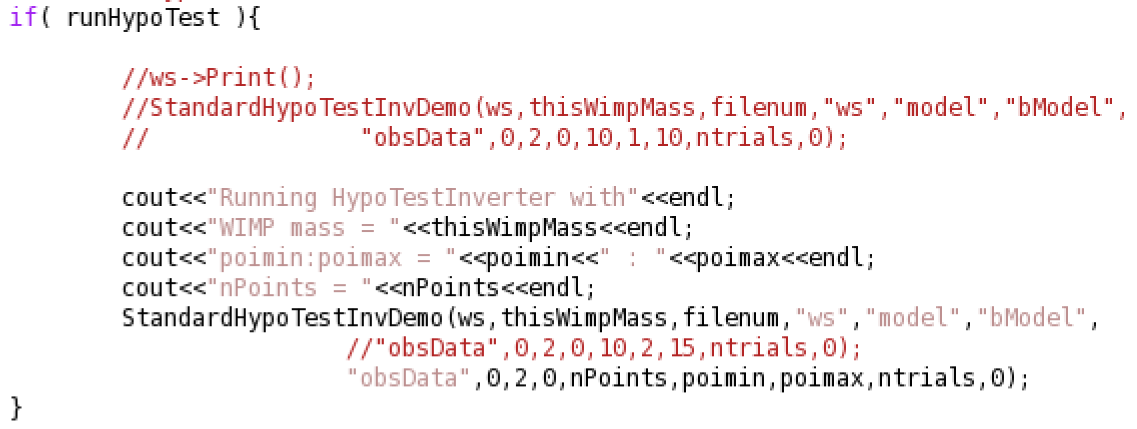
```
ws->import(*searchData);
```

- … Imports the data set that was just created into the workspace.



```
if(useXSec == 1){
        ws->var("xsec")->setVal((poimin+poimax)/2.0);
} else {
        ws->var("nSig")->setVal(1.0);
}
```

- Sets the poi to a value (look for yourself to see what)

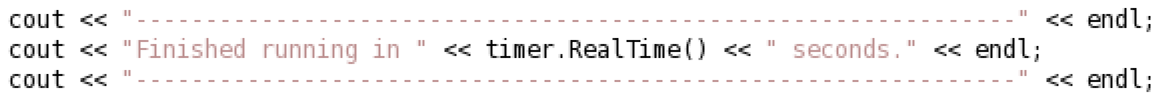


```
if( runHypoTest ){

        //ws->Print();
        //StandardHypoTestInvDemo(ws,thisWimpMass,filenum,"ws","model","bModel",
        //                "obsData",0,2,0,10,1,10,ntrials,0);

        cout<<"Running HypoTestInverter with"<<endl;
        cout<<"WIMP mass = "<<thisWimpMass<<endl;
        cout<<"poimin:poimax = "<<poimin<<" : "<<poimax<<endl;
        cout<<"nPoints = "<<nPoints<<endl;
        StandardHypoTestInvDemo(ws,thisWimpMass,filenum,"ws","model","bModel",
                        //"obsData",0,2,0,10,2,15,ntrials,0);
                        "obsData",0,2,0,nPoints,poimin,poimax,ntrials,0);
}
```

- Runs the HypoTestInvDemo (the actual PLR)

- To me this is a big black box.

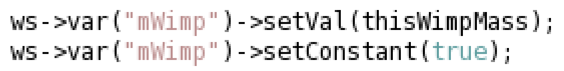
- Note there is no difference between “model” and “bModel” except that the poi in model has been set to 0.



```
cout << "---------------------------------------------------------------" << endl;
cout << "Finished running in " << timer.RealTime() << " seconds." << endl;
cout << "---------------------------------------------------------------" << endl;
```

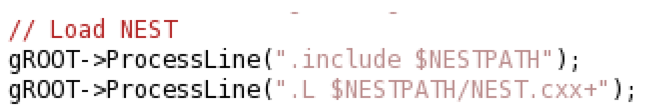
- Tells you how long it took

ImportSignalModel_5D



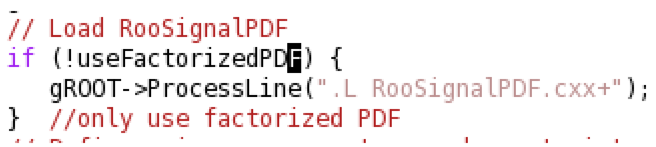
```
ws->var("mWimp")->setVal(thisWimpMass);
ws->var("mWimp")->setConstant(true);
```

- Sets value of the wimp mass mWimp defined in “makeWorkspace”
- Makes the wimp mass constant



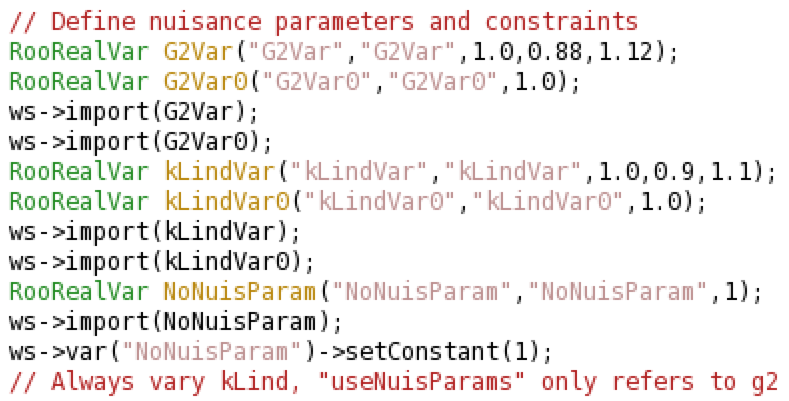
```
// Load NEST
gROOT->ProcessLine(".include $NESTPATH");
gROOT->ProcessLine(".L $NESTPATH/NEST.cxx+");
```

- Compiles NEST.cxx and makes the functions available to this file.



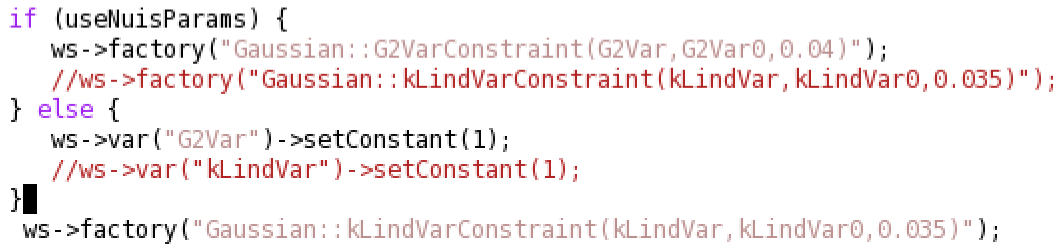
```
// Load RooSignalPDF
if (!useFactorizedPDF) {
   gROOT->ProcessLine(".L RooSignalPDF.cxx+");
}  //only use factorized PDF
```

- Makes RooSignalPDF.cxx functions available to current file



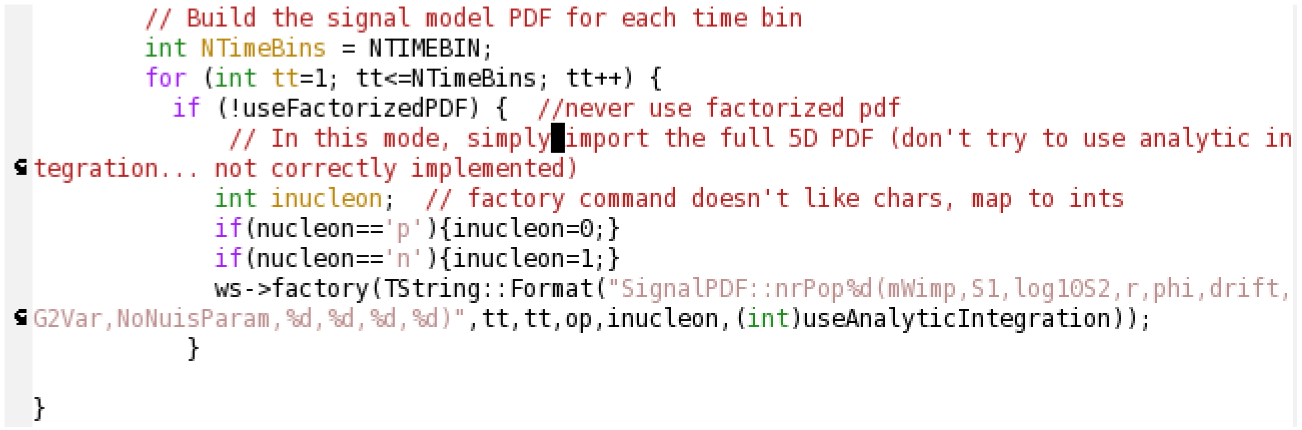
```
// Define nuisance parameters and constraints
RooRealVar G2Var("G2Var","G2Var",1.0,0.88,1.12);
RooRealVar G2Var0("G2Var0","G2Var0",1.0);
ws->import(G2Var);
ws->import(G2Var0);
RooRealVar kLindVar("kLindVar","kLindVar",1.0,0.9,1.1);
RooRealVar kLindVar0("kLindVar0","kLindVar0",1.0);
ws->import(kLindVar);
ws->import(kLindVar0);
RooRealVar NoNuisParam("NoNuisParam","NoNuisParam",1);
ws->import(NoNuisParam);
ws->var("NoNuisParam")->setConstant(1);
// Always vary kLind, "useNuisParams" only refers to g2
```

- Creates several RooRealVars
- Sets the RooRealVar “noNuisParam” to 1 and makes it constant (presumably to be used in turning on and off G2Var)



```
if (useNuisParams) {
   ws->factory("Gaussian::G2VarConstraint(G2Var,G2Var0,0.04)");
   //ws->factory("Gaussian::kLindVarConstraint(kLindVar,kLindVar0,0.035)");
} else {
   ws->var("G2Var")->setConstant(1);
   //ws->var("kLindVar")->setConstant(1);
}
 ws->factory("Gaussian::kLindVarConstraint(kLindVar,kLindVar0,0.035)");
```

- Constructs a gaussian pdf called "G2VarConstraint" if using nuisance parameters, otherwise setting the RooRealVar "G2Var" constant.
- Always constructs a gaussian pdf called "kLindVarConstraint"

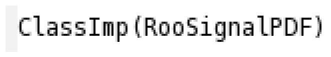


```
        // Build the signal model PDF for each time bin
        int NTimeBins = NTIMEBIN;
        for (int tt=1; tt<=NTimeBins; tt++) {
          if (!useFactorizedPDF) {  //never use factorized pdf
            // In this mode, simply import the full 5D PDF (don't try to use analytic in
tegration... not correctly implemented)
            int inucleon;  // factory command doesn't like chars, map to ints
            if(nucleon=='p'){inucleon=0;}
            if(nucleon=='n'){inucleon=1;}
            ws->factory(TString::Format("SignalPDF::nrPop%d(mWimp,S1,log10S2,r,phi,drift,
G2Var,NoNuisParam,%d,%d,%d,%d)",tt,tt,op,inucleon,(int)useAnalyticIntegration));
          }

}
```

- Loop through each time bin and create a pdf of type “SignalPDF” called “nrPop[time bin]” for each and add them to the workspace.

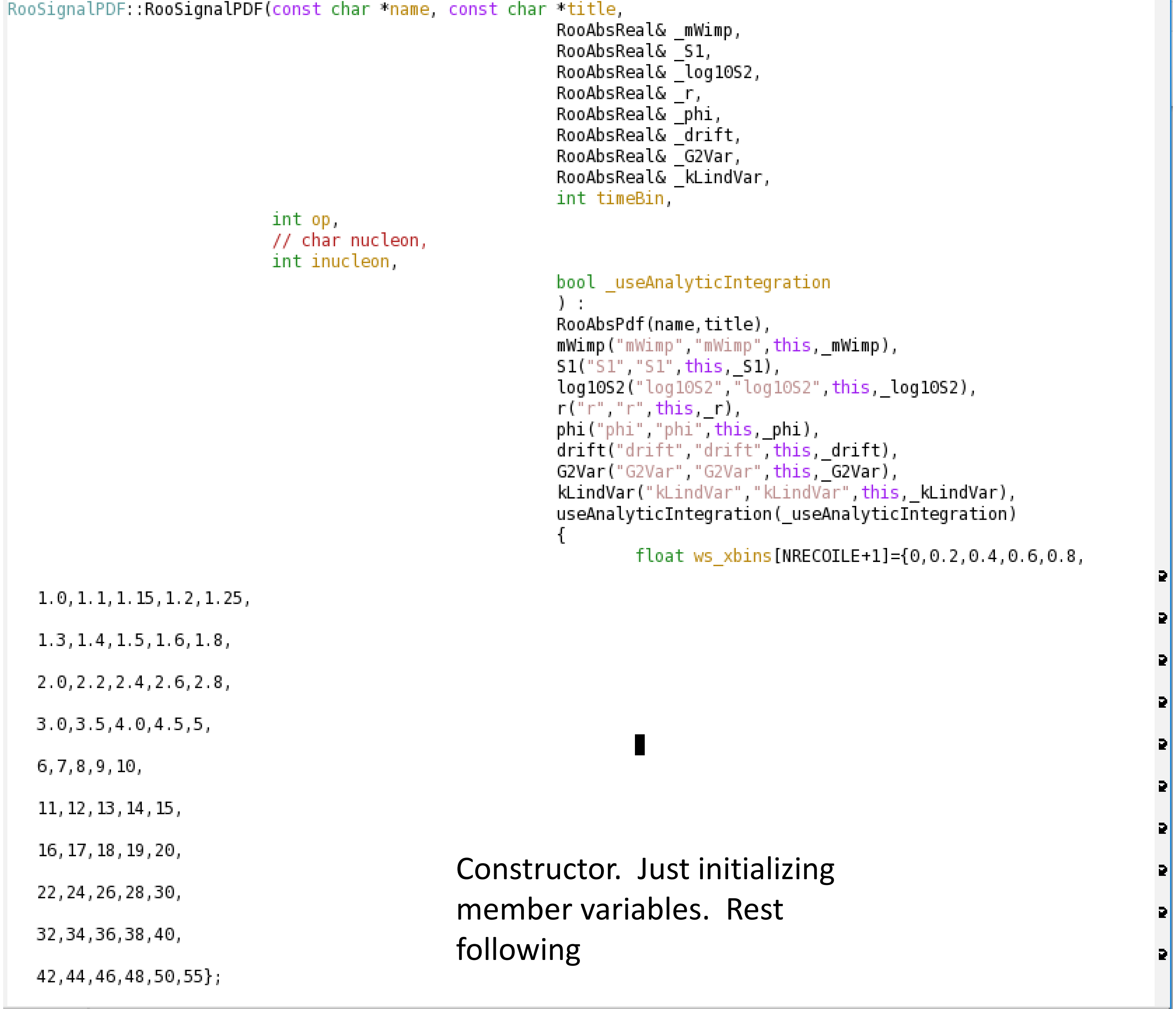
RooSignalPDF.cxx

- RooSignalPDF::RooSignalPDF



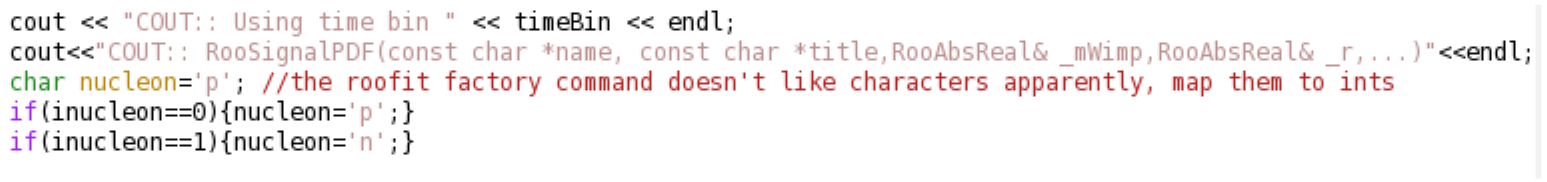
```
ClassImp(RooSignalPDF)
```

- Not sure what this is…



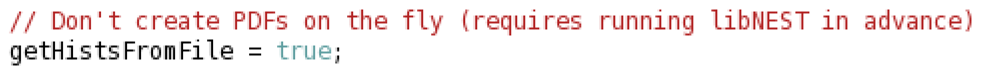
```
RooSignalPDF::RooSignalPDF(const char *name, const char *title,
                                             RooAbsReal& _mWimp,
                                             RooAbsReal& _S1,
                                             RooAbsReal& _log10S2,
                                             RooAbsReal& _r,
                                             RooAbsReal& _phi,
                                             RooAbsReal& _drift,
                                             RooAbsReal& _G2Var,
                                             RooAbsReal& _kLindVar,
                                             int timeBin,
                       int op,
                       // char nucleon,
                       int inucleon,
                                             bool _useAnalyticIntegration
                                             ) :
                                             RooAbsPdf(name,title),
                                             mWimp("mWimp","mWimp",this,_mWimp),
                                             S1("S1","S1",this,_S1),
                                             log10S2("log10S2","log10S2",this,_log10S2),
                                             r("r","r",this,_r),
                                             phi("phi","phi",this,_phi),
                                             drift("drift","drift",this,_drift),
                                             G2Var("G2Var","G2Var",this,_G2Var),
                                             kLindVar("kLindVar","kLindVar",this,_kLindVar),
                                             useAnalyticIntegration(_useAnalyticIntegration)
                                             {
                                                     float ws_xbins[NRECOILE+1]={0,0.2,0.4,0.6,0.8,
  1.0,1.1,1.15,1.2,1.25,
  1.3,1.4,1.5,1.6,1.8,
  2.0,2.2,2.4,2.6,2.8,
  3.0,3.5,4.0,4.5,5,
  6,7,8,9,10,
  11,12,13,14,15,
  16,17,18,19,20,
  22,24,26,28,30,
  32,34,36,38,40,
  42,44,46,48,50,55};
```

Constructor. Just initializing member variables. Rest following



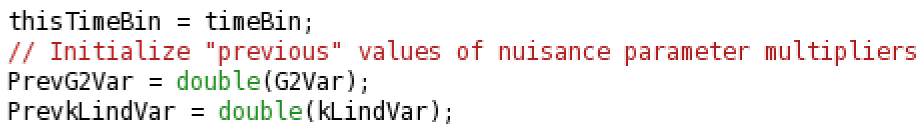
```
cout << "COUT:: Using time bin " << timeBin << endl;
cout<<"COUT:: RooSignalPDF(const char *name, const char *title,RooAbsReal& _mWimp,RooAbsReal& _r,...)"<<endl;
char nucleon='p'; //the roofit factory command doesn't like characters apparently, map them to ints
if(inucleon==0){nucleon='p';}
if(inucleon==1){nucleon='n';}
```

- Just check which nucleon we're using and set it accordingly.



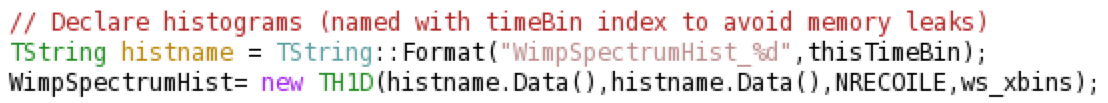
```
// Don't create PDFs on the fly (requires running libNEST in advance)
getHistsFromFile = true;
```

- Set flag to indicate that we’ve already calculated the signal model histograms



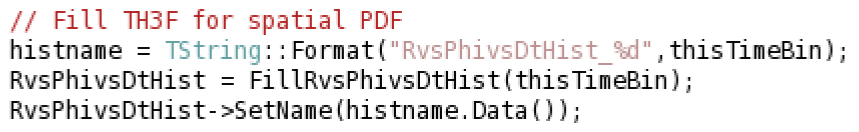
```
thisTimeBin = timeBin;
// Initialize "previous" values of nuisance parameter multipliers
PrevG2Var = double(G2Var);
PrevkLindVar = double(kLindVar);
```

- Set variables that will be iterated to remember previous values.



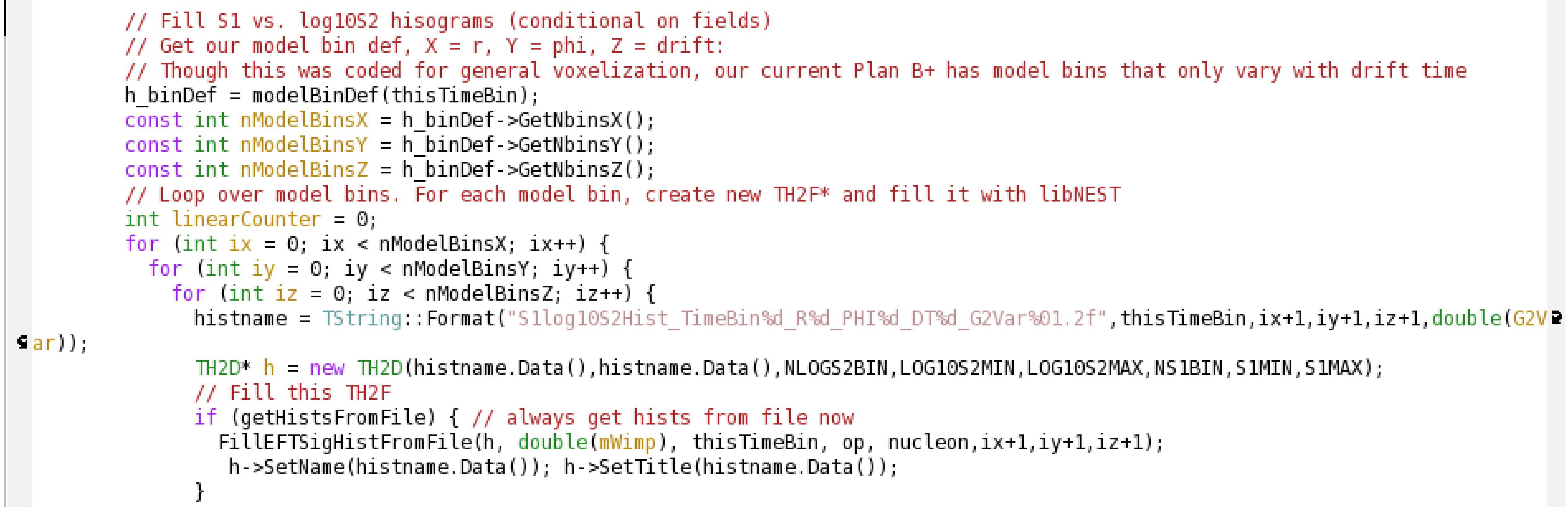
```
// Declare histograms (named with timeBin index to avoid memory leaks)
TString histname = TString::Format("WimpSpectrumHist_%d",thisTimeBin);
WimpSpectrumHist= new TH1D(histname.Data(),histname.Data(),NRECOILE,ws_xbins);
```

- Create a new histogram object to store the wimp recoil spectrum.



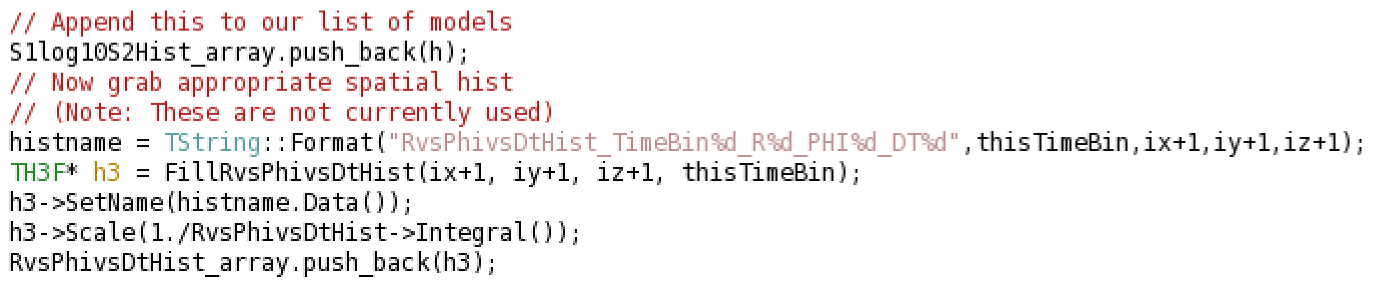
```
// Fill TH3F for spatial PDF
histname = TString::Format("RvsPhivsDtHist_%d",thisTimeBin);
RvsPhivsDtHist = FillRvsPhivsDtHist(thisTimeBin);
RvsPhivsDtHist->SetName(histname.Data());
```

- Creates a histogram to store the spatial portion of the PDF for the signal model



```
      // Fill S1 vs. log10S2 hisograms (conditional on fields)
      // Get our model bin def, X = r, Y = phi, Z = drift:
      // Though this was coded for general voxelization, our current Plan B+ has model bins that only vary with drift time
      h_binDef = modelBinDef(thisTimeBin);
      const int nModelBinsX = h_binDef->GetNbinsX();
      const int nModelBinsY = h_binDef->GetNbinsY();
      const int nModelBinsZ = h_binDef->GetNbinsZ();
      // Loop over model bins. For each model bin, create new TH2F* and fill it with libNEST
      int linearCounter = 0;
      for (int ix = 0; ix < nModelBinsX; ix++) {
        for (int iy = 0; iy < nModelBinsY; iy++) {
          for (int iz = 0; iz < nModelBinsZ; iz++) {
            histname = TString::Format("S1log10S2Hist_TimeBin%d_R%d_PHI%d_DT%d_G2Var%01.2f",thisTimeBin,ix+1,iy+1,iz+1,double(G2V
ar));

            TH2D* h = new TH2D(histname.Data(),histname.Data(),NLOGS2BIN,LOG10S2MIN,LOG10S2MAX,NS1BIN,S1MIN,S1MAX);
            // Fill this TH2F
            if (getHistsFromFile) { // always get hists from file now
              FillEFTSigHistFromFile(h, double(mWimp), thisTimeBin, op, nucleon,ix+1,iy+1,iz+1);
               h->SetName(histname.Data()); h->SetTitle(histname.Data());
            }
```

- Loop through each “model bin” (only voxelized in z in reality) (continues in what follows)
- Create a 2d histogram for each bin to store the s1 vs s2 information from the signal model.
- Use “FillEFTSigHistFromFile” to fill each histogram

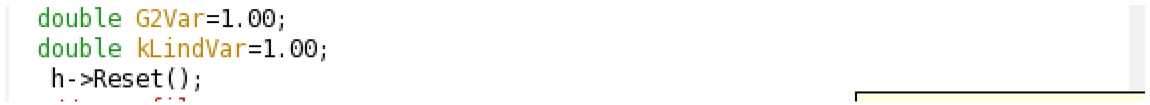


```
// Append this to our list of models
S1log10S2Hist_array.push_back(h);
// Now grab appropriate spatial hist
// (Note: These are not currently used)
histname = TString::Format("RvsPhivsDtHist_TimeBin%d_R%d_PHI%d_DT%d",thisTimeBin,ix+1,iy+1,iz+1);
TH3F* h3 = FillRvsPhivsDtHist(ix+1, iy+1, iz+1, thisTimeBin);
h3->SetName(histname.Data());
h3->Scale(1./RvsPhivsDtHist->Integral());
RvsPhivsDtHist_array.push_back(h3);
```

- Add the histogram to an array of histograms. (one for each bin)
- Also add the corresponding spatial histogram

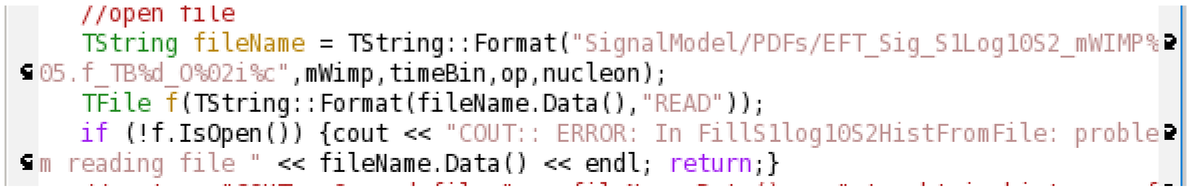
ImportSigHists.h

- FillEFTSigHistFromFile



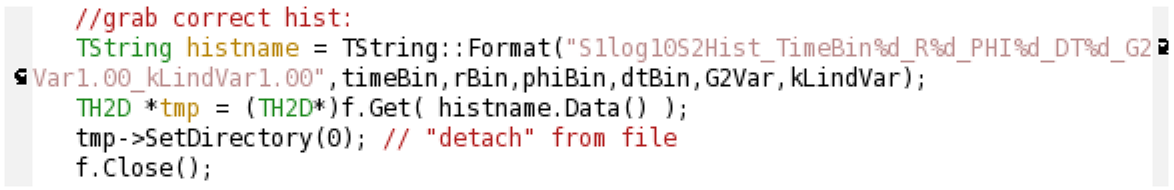
```
double G2Var=1.00;
double kLindVar=1.00;
 h->Reset();
```

- Set a couple of constants
- Clear the histogram of contents if it for some reason already has some.



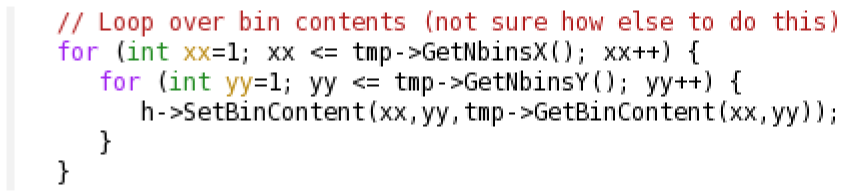
```
    //open file
    TString fileName = TString::Format("SignalModel/PDFs/EFT_Sig_S1Log10S2_mWIMP%
05.f_TB%d_O%02i%c",mWimp,timeBin,op,nucleon);
    TFile f(TString::Format(fileName.Data(),"READ"));
    if (!f.IsOpen()) {cout << "COUT:: ERROR: In FillS1log10S2HistFromFile: proble
m reading file " << fileName.Data() << endl; return;}
```

- Open the correct file
- Spit out an error message if the file couldn’t be opened.



```
    //grab correct hist:
    TString histname = TString::Format("S1log10S2Hist_TimeBin%d_R%d_PHI%d_DT%d_G2
Var1.00_kLindVar1.00",timeBin,rBin,phiBin,dtBin,G2Var,kLindVar);
    TH2D *tmp = (TH2D*)f.Get( histname.Data() );
    tmp->SetDirectory(0); // "detach" from file
    f.Close();
```

- Grab the correct histogram from the file in which it was stored
  - Create a new histogram
  - Fill it with the "get" function
  - Close the file

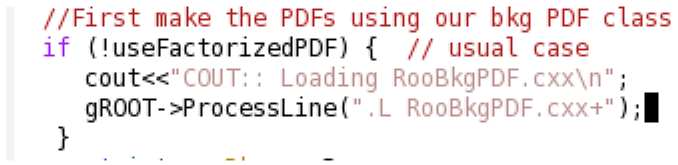


```
// Loop over bin contents (not sure how else to do this)
for (int xx=1; xx <= tmp->GetNbinsX(); xx++) {
   for (int yy=1; yy <= tmp->GetNbinsY(); yy++) {
      h->SetBinContent(xx,yy,tmp->GetBinContent(xx,yy));
   }
}
```

- Grab the data from the temporary histogram and transfer it into the one passed to the function.

ImportBkgModel.h

- ImportBkgModel

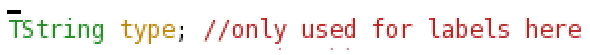


```
//First make the PDFs using our bkg PDF class
if (!useFactorizedPDF) {  // usual case
   cout<<"COUT:: Loading RooBkgPDF.cxx\n";
   gROOT->ProcessLine(".L RooBkgPDF.cxx+");
 }
```

- Compile RooBkgPDF and allow us to use its functions in this file

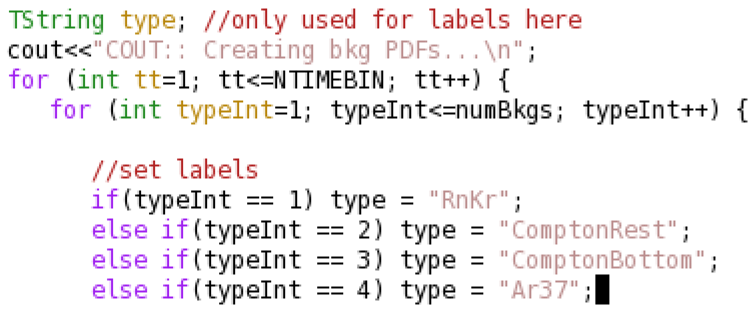
```
const int numBkgs = 3;
```

- ‘nuff said



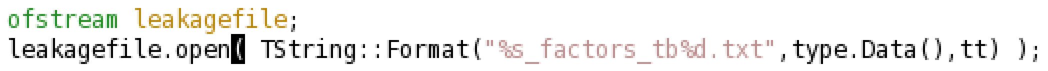
```
TString type; //only used for labels here
```

- ¯\_(ツ)_/¯



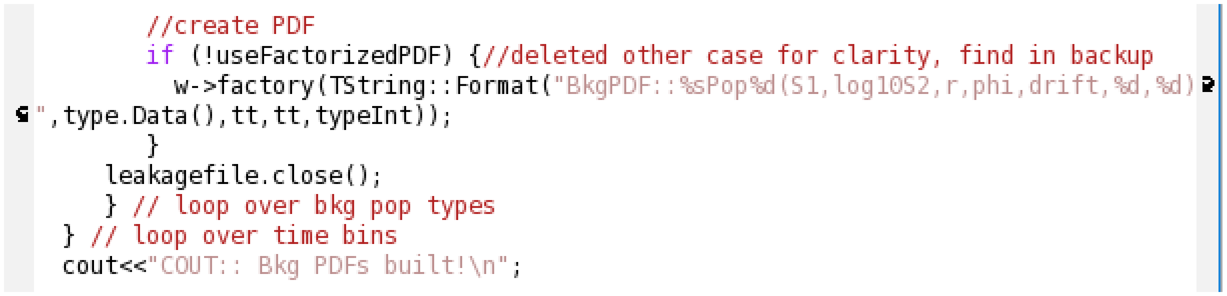
```
TString type; //only used for labels here
cout<<"COUT:: Creating bkg PDFs...\n";
for (int tt=1; tt<=NTIMEBIN; tt++) {
   for (int typeInt=1; typeInt<=numBkgs; typeInt++) {

      //set labels
      if(typeInt == 1) type = "RnKr";
      else if(typeInt == 2) type = "ComptonRest";
      else if(typeInt == 3) type = "ComptonBottom";
      else if(typeInt == 4) type = "Ar37";
```

- Initiate loop through time bins and background types
- Note numBkgs was set to 3 so Ar37 is not currently used.



```
ofstream leakagefile;
leakagefile.open( TString::Format("%s_factors_tb%d.txt",type.Data(),tt) );
```

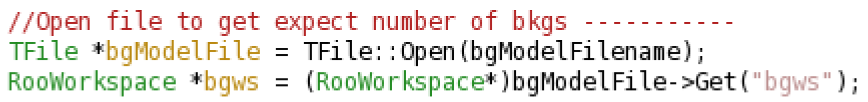
- Opens a file to write information into based on bkg type and time bin



```
        //create PDF
        if (!useFactorizedPDF) {//deleted other case for clarity, find in backup
          w->factory(TString::Format("BkgPDF::%sPop%d(S1,log10S2,r,phi,drift,%d,%d)
",type.Data(),tt,tt,typeInt));
        }
      leakagefile.close();
      } // loop over bkg pop types
    } // loop over time bins
    cout<<"COUT:: Bkg PDFs built!\n";
```

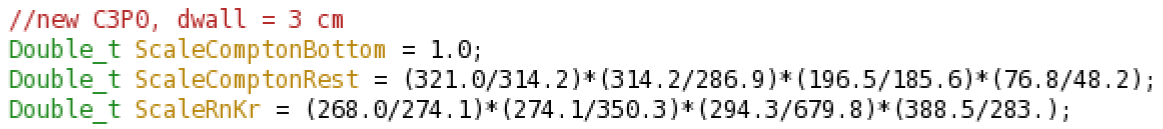
- Creates a PDF of type “RooBkgPDF” named [bkg type]Pop[time bin] and adds it to the workspace. (Utilizes RooBkgPDF constructor)
- Closes the file opened earlier to log things.

```
//Now create the scaling factors for each bkg type
//Each bkg term in the likelihood looks like:
// r_bkg,timebin * N_bkg,timebin * PDF_bkg,timebin
// where:
// r_bkg,timebin = N_bkg,timebin / sumOverTimeBins( N_bkg,timebin );
//
//Note: the expected number of events stored in the bgws file
//      are normalized to the full run 4 exposure, so we actually
//      multiply by the livetime fraction of each time bin here
//      to get the correct expectation.
```



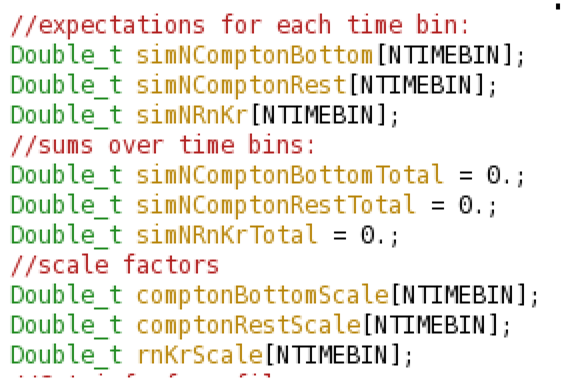
```
//Open file to get expect number of bkgs -----------
TFile *bgModelFile = TFile::Open(bgModelFilename);
RooWorkspace *bgws = (RooWorkspace*)bgModelFile->Get("bgws");
```

- Opens the file with the background model. This file just contains a rooWorkspace.
- Grabs a new workspace “bgws” from the file just for backgrounds.



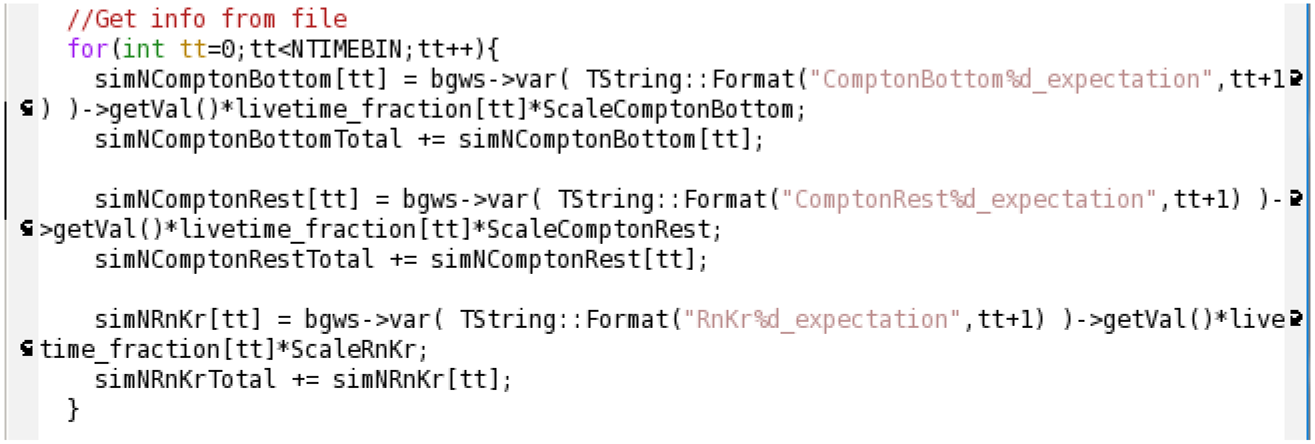
```
//new C3P0, dwall = 3 cm
Double_t ScaleComptonBottom = 1.0;
Double_t ScaleComptonRest = (321.0/314.2)*(314.2/286.9)*(196.5/185.6)*(76.8/48.2);
Double_t ScaleRnKr = (268.0/274.1)*(274.1/350.3)*(294.3/679.8)*(388.5/283.);
```

- Hard-code scale factor variables.



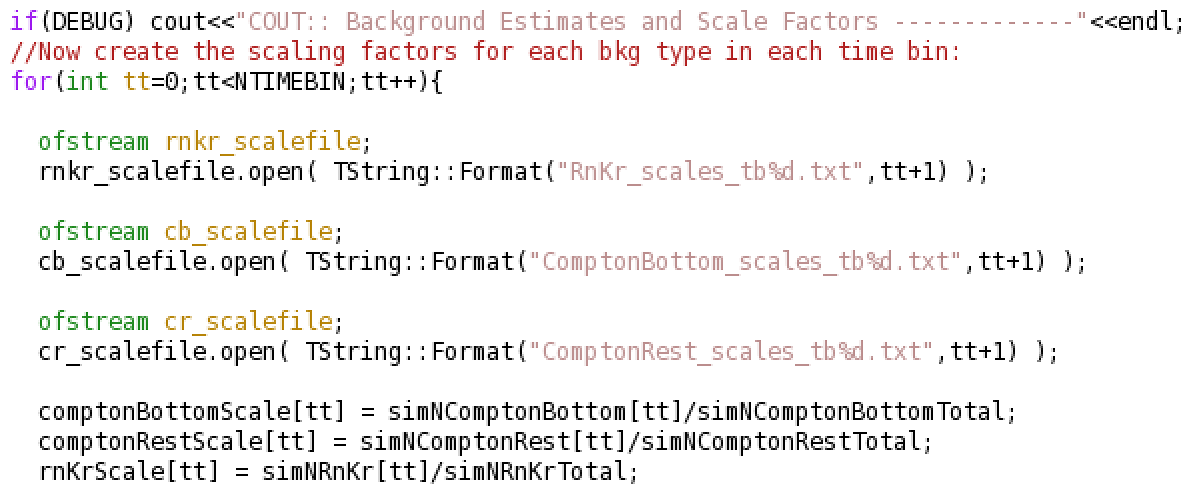
```
//expectations for each time bin:
Double_t simNComptonBottom[NTIMEBIN];
Double_t simNComptonRest[NTIMEBIN];
Double_t simNRnKr[NTIMEBIN];
//sums over time bins:
Double_t simNComptonBottomTotal = 0.;
Double_t simNComptonRestTotal = 0.;
Double_t simNRnKrTotal = 0.;
//scale factors
Double_t comptonBottomScale[NTIMEBIN];
Double_t comptonRestScale[NTIMEBIN];
Double_t rnKrScale[NTIMEBIN];
```

- Creates a number of arrays (and some doubles, just for the totals).
  - Expectations
  - Scale factors
  - (whatever this means)



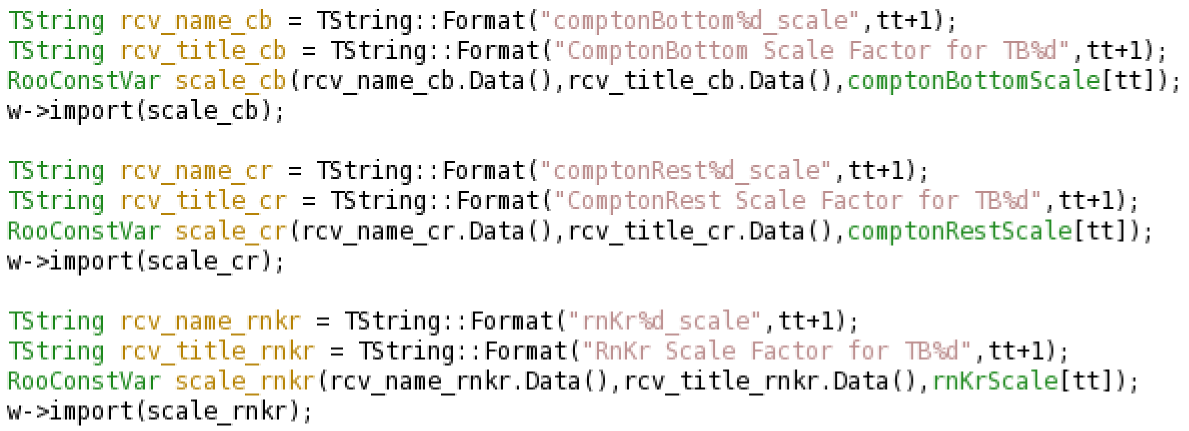
```
    //Get info from file
    for(int tt=0;tt<NTIMEBIN;tt++){
      simNComptonBottom[tt] = bgws->var( TString::Format("ComptonBottom%d_expectation",tt+1
) )->getVal()*livetime_fraction[tt]*ScaleComptonBottom;
      simNComptonBottomTotal += simNComptonBottom[tt];

      simNComptonRest[tt] = bgws->var( TString::Format("ComptonRest%d_expectation",tt+1) )-
>getVal()*livetime_fraction[tt]*ScaleComptonRest;
      simNComptonRestTotal += simNComptonRest[tt];

      simNRnKr[tt] = bgws->var( TString::Format("RnKr%d_expectation",tt+1) )->getVal()*live
time_fraction[tt]*ScaleRnKr;
      simNRnKrTotal += simNRnKr[tt];
    }
```

- Populate the simN arrays with data from the background model.
- Determine the totals from these arrays.



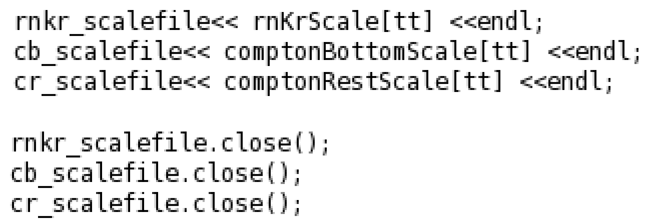
```
if(DEBUG) cout<<"COUT:: Background Estimates and Scale Factors -------------"<<endl;
//Now create the scaling factors for each bkg type in each time bin:
for(int tt=0;tt<NTIMEBIN;tt++){

  ofstream rnkr_scalefile;
  rnkr_scalefile.open( TString::Format("RnKr_scales_tb%d.txt",tt+1) );

  ofstream cb_scalefile;
  cb_scalefile.open( TString::Format("ComptonBottom_scales_tb%d.txt",tt+1) );

  ofstream cr_scalefile;
  cr_scalefile.open( TString::Format("ComptonRest_scales_tb%d.txt",tt+1) );

  comptonBottomScale[tt] = simNComptonBottom[tt]/simNComptonBottomTotal;
  comptonRestScale[tt] = simNComptonRest[tt]/simNComptonRestTotal;
  rnKrScale[tt] = simNRnKr[tt]/simNRnKrTotal;
```

- Open files to keep track of the scale factors.
- Determine the scale factors for each background based on the simN numbers for each time bin and the totals.



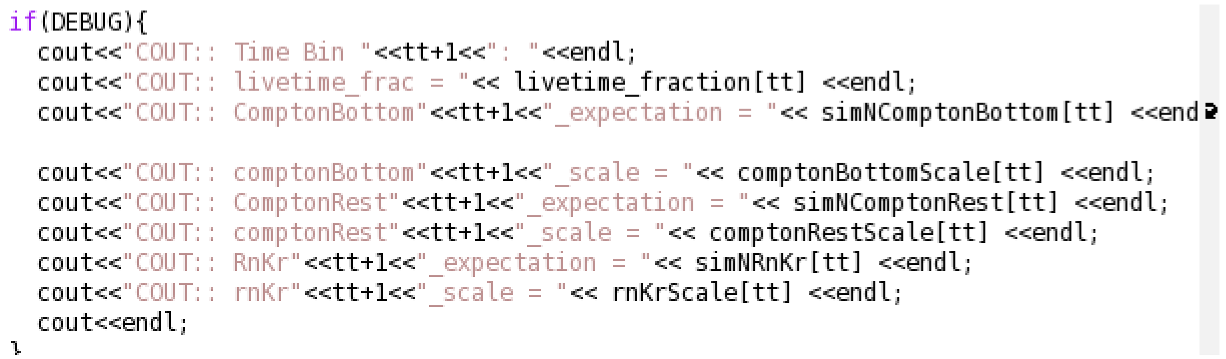
```
TString rcv_name_cb = TString::Format("comptonBottom%d_scale",tt+1);
TString rcv_title_cb = TString::Format("ComptonBottom Scale Factor for TB%d",tt+1);
RooConstVar scale_cb(rcv_name_cb.Data(),rcv_title_cb.Data(),comptonBottomScale[tt]);
w->import(scale_cb);

TString rcv_name_cr = TString::Format("comptonRest%d_scale",tt+1);
TString rcv_title_cr = TString::Format("ComptonRest Scale Factor for TB%d",tt+1);
RooConstVar scale_cr(rcv_name_cr.Data(),rcv_title_cr.Data(),comptonRestScale[tt]);
w->import(scale_cr);

TString rcv_name_rnkr = TString::Format("rnKr%d_scale",tt+1);
TString rcv_title_rnkr = TString::Format("RnKr Scale Factor for TB%d",tt+1);
RooConstVar scale_rnkr(rcv_name_rnkr.Data(),rcv_title_rnkr.Data(),rnKrScale[tt]);
w->import(scale_rnkr);
```

- Create RooConstVars for each of the background scale factors and import them into the workspace.



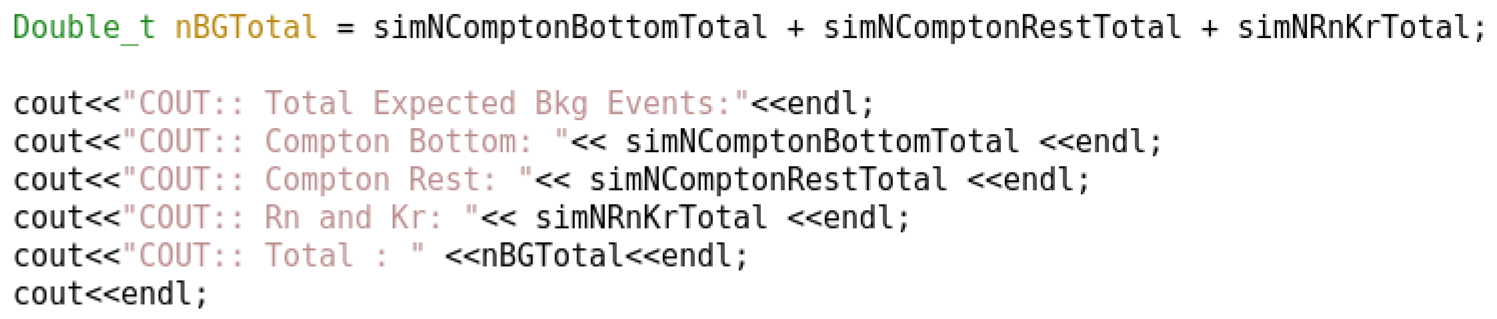
```
rnkr_scalefile<< rnKrScale[tt] <<endl;
cb_scalefile<< comptonBottomScale[tt] <<endl;
cr_scalefile<< comptonRestScale[tt] <<endl;

rnkr_scalefile.close();
cb_scalefile.close();
cr_scalefile.close();
```

- Write the scale factors to the previously opened files and close them.



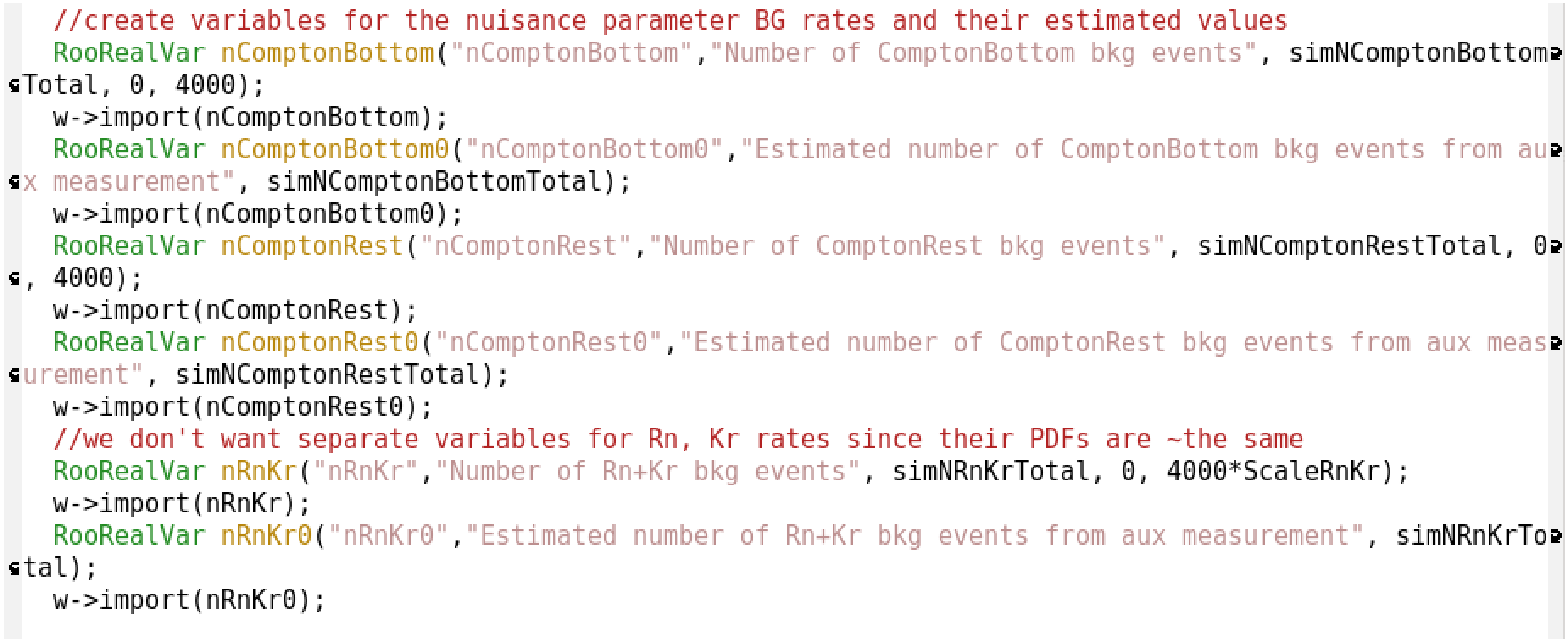
```
if(DEBUG){
  cout<<"COUT:: Time Bin "<<tt+1<<": "<<endl;
  cout<<"COUT:: livetime_frac = "<< livetime_fraction[tt] <<endl;
  cout<<"COUT:: ComptonBottom"<<tt+1<<"_expectation = "<< simNComptonBottom[tt] <<end

  cout<<"COUT:: comptonBottom"<<tt+1<<"_scale = "<< comptonBottomScale[tt] <<endl;
  cout<<"COUT:: ComptonRest"<<tt+1<<"_expectation = "<< simNComptonRest[tt] <<endl;
  cout<<"COUT:: comptonRest"<<tt+1<<"_scale = "<< comptonRestScale[tt] <<endl;
  cout<<"COUT:: RnKr"<<tt+1<<"_expectation = "<< simNRnKr[tt] <<endl;
  cout<<"COUT:: rnKr"<<tt+1<<"_scale = "<< rnKrScale[tt] <<endl;
  cout<<endl;
}
```

- Print debug information if DEBUG flag is on



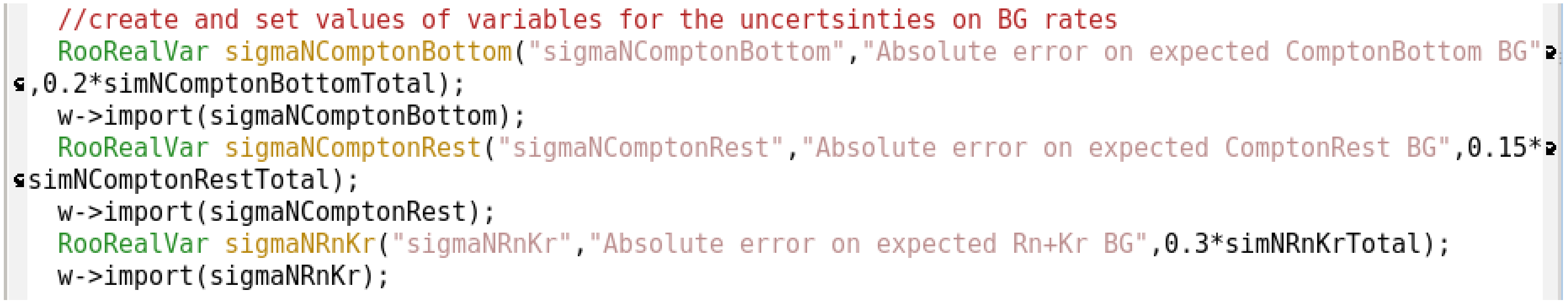
```
Double_t nBGTotal = simNComptonBottomTotal + simNComptonRestTotal + simNRnKrTotal;

cout<<"COUT:: Total Expected Bkg Events:"<<endl;
cout<<"COUT:: Compton Bottom: "<< simNComptonBottomTotal <<endl;
cout<<"COUT:: Compton Rest: "<< simNComptonRestTotal <<endl;
cout<<"COUT:: Rn and Kr: "<< simNRnKrTotal <<endl;
cout<<"COUT:: Total : " <<nBGTotal<<endl;
cout<<endl;
```

- Record the total number of background events
- Print out the background numbers to console



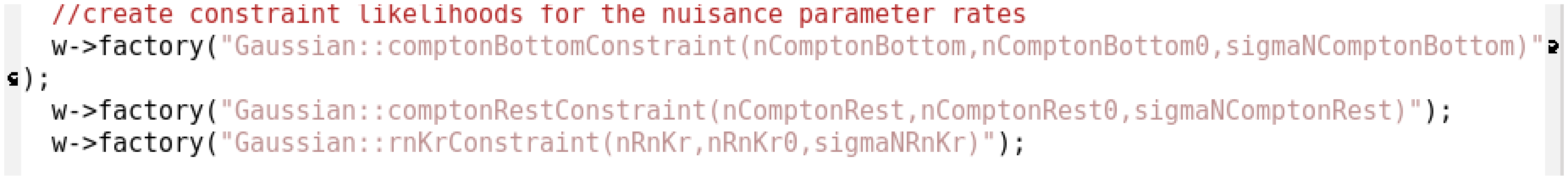
```
  //create variables for the nuisance parameter BG rates and their estimated values
  RooRealVar nComptonBottom("nComptonBottom","Number of ComptonBottom bkg events", simNComptonBottomTotal, 0, 4000);
  w->import(nComptonBottom);
  RooRealVar nComptonBottom0("nComptonBottom0","Estimated number of ComptonBottom bkg events from aux measurement", simNComptonBottomTotal);
  w->import(nComptonBottom0);
  RooRealVar nComptonRest("nComptonRest","Number of ComptonRest bkg events", simNComptonRestTotal, 0, 4000);
  w->import(nComptonRest);
  RooRealVar nComptonRest0("nComptonRest0","Estimated number of ComptonRest bkg events from aux measurement", simNComptonRestTotal);
  w->import(nComptonRest0);
  //we don't want separate variables for Rn, Kr rates since their PDFs are ~the same
  RooRealVar nRnKr("nRnKr","Number of Rn+Kr bkg events", simNRnKrTotal, 0, 4000*ScaleRnKr);
  w->import(nRnKr);
  RooRealVar nRnKr0("nRnKr0","Estimated number of Rn+Kr bkg events from aux measurement", simNRnKrTotal);
  w->import(nRnKr0);
```

- Create RooRealVars for each background number of events.
- Create a RooRealVar that will remain constant for the mean of each constraing (ones ending in 0)



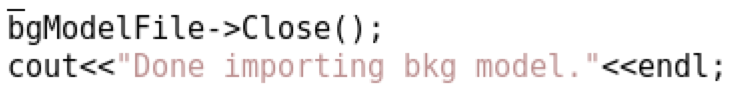
```
  //create and set values of variables for the uncertsinties on BG rates
  RooRealVar sigmaNComptonBottom("sigmaNComptonBottom","Absolute error on expected ComptonBottom BG",0.2*simNComptonBottomTotal);
  w->import(sigmaNComptonBottom);
  RooRealVar sigmaNComptonRest("sigmaNComptonRest","Absolute error on expected ComptonRest BG",0.15*simNComptonRestTotal);
  w->import(sigmaNComptonRest);
  RooRealVar sigmaNRnKr("sigmaNRnKr","Absolute error on expected Rn+Kr BG",0.3*simNRnKrTotal);
  w->import(sigmaNRnKr);
```

- Create RooRealVars for the spread of the gaussian constraints



```
  //create constraint likelihoods for the nuisance parameter rates
  w->factory("Gaussian::comptonBottomConstraint(nComptonBottom,nComptonBottom0,sigmaNComptonBottom)");
  w->factory("Gaussian::comptonRestConstraint(nComptonRest,nComptonRest0,sigmaNComptonRest)");
  w->factory("Gaussian::rnKrConstraint(nRnKr,nRnKr0,sigmaNRnKr)");
```

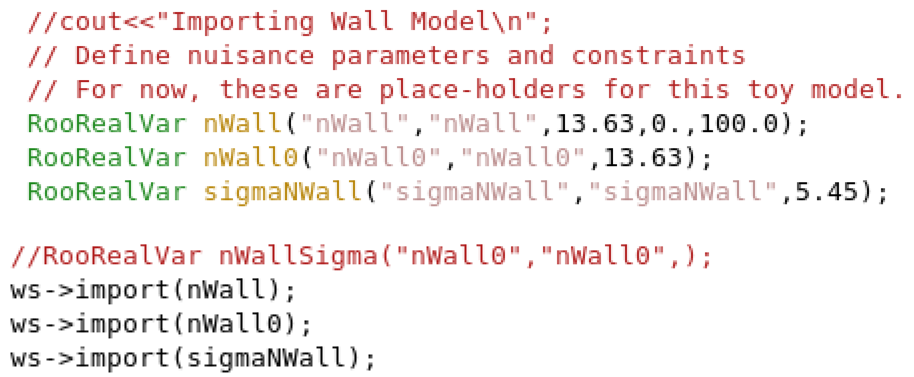
- Creates PDFs (gaussian pdfs, names comptonBottomConstraint etc) to put a constraint on these backgrounds.



```
bgModelFile->Close();
cout<<"Done importing bkg model."<<endl;
```

- Close the file containing background information.

ImportWallModel

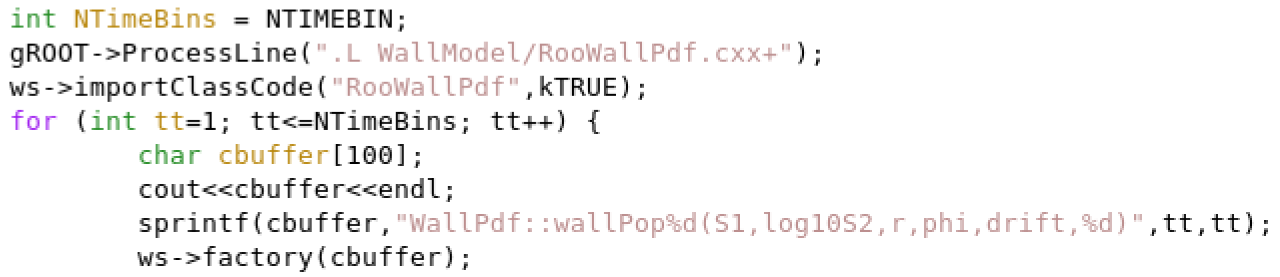


```
 //cout<<"Importing Wall Model\n";
 // Define nuisance parameters and constraints
 // For now, these are place-holders for this toy model.
 RooRealVar nWall("nWall","nWall",13.63,0.,100.0);
 RooRealVar nWall0("nWall0","nWall0",13.63);
 RooRealVar sigmaNWall("sigmaNWall","sigmaNWall",5.45);

//RooRealVar nWallSigma("nWall0","nWall0",);
ws->import(nWall);
ws->import(nWall0);
ws->import(sigmaNWall);
```

- Creates rooRealVars for the number of wall events, mean, and deviation of the gaussian constraing pdf.
- Then add these variables to the workspace.

```
ws->factory("Gaussian::WallConstraint(nWall,nWall0,sigmaNWall)");
```

- Creates a gaussian pdf called “WallConstraint” using the aforementioned variables.

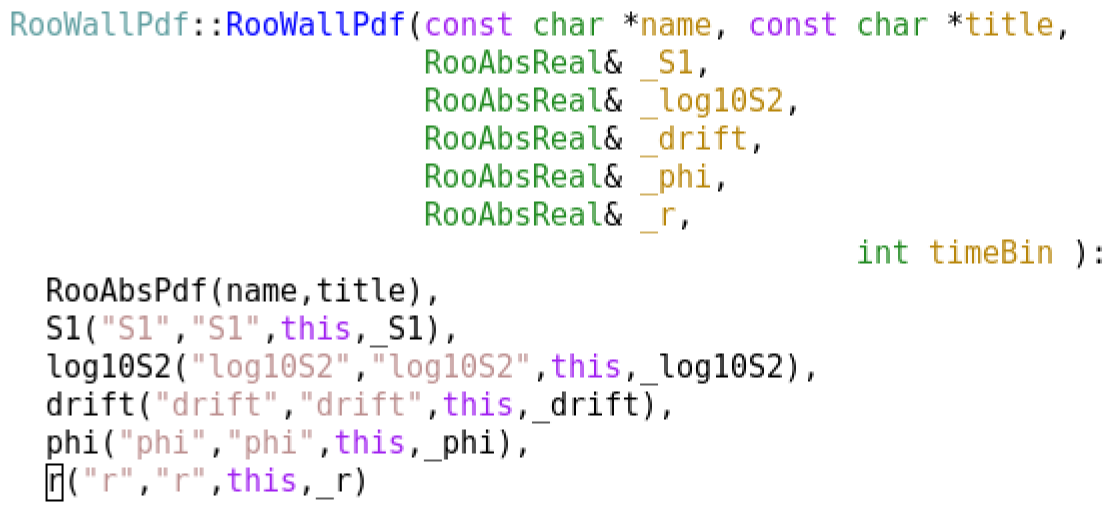


```
int NTimeBins = NTIMEBIN;
gROOT->ProcessLine(".L WallModel/RooWallPdf.cxx+");
ws->importClassCode("RooWallPdf",kTRUE);
for (int tt=1; tt<=NTimeBins; tt++) {
        char cbuffer[100];
        cout<<cbuffer<<endl;
        sprintf(cbuffer,"WallPdf::wallPop%d(S1,log10S2,r,phi,drift,%d)",tt,tt);
        ws->factory(cbuffer);
```

- Makes the functions in RooWallPdf accessable to this file
- Makes a PDF of type “WallPdf” called “wallPop[time bin]” for each time bin using the constructor found for the class RooWallPdf and adds it to the workspace.

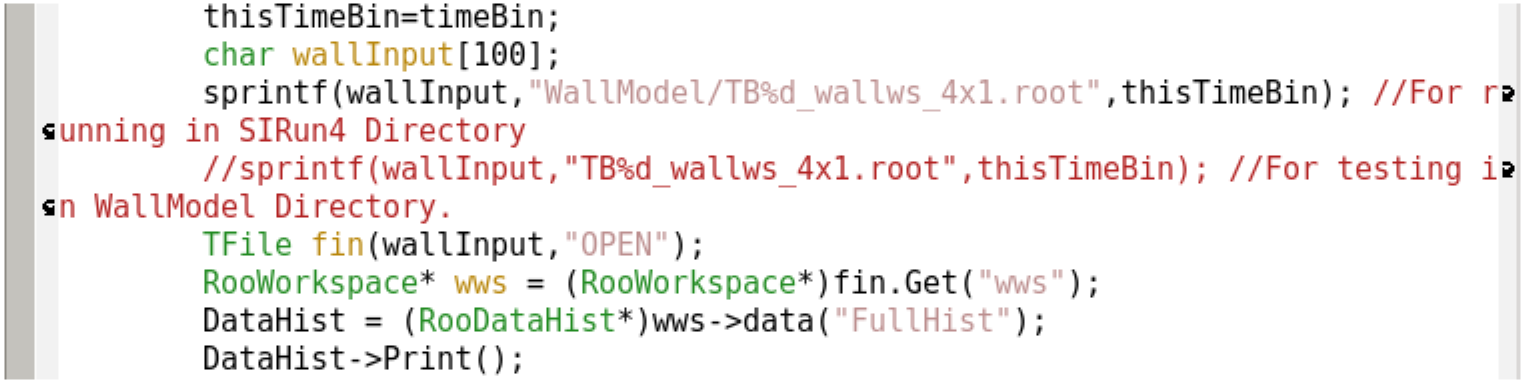
RooWallPdf.cxx

- constructor



```
RooWallPdf::RooWallPdf(const char *name, const char *title,
                       RooAbsReal& _S1,
                       RooAbsReal& _log10S2,
                       RooAbsReal& _drift,
                       RooAbsReal& _phi,
                       RooAbsReal& _r,
                                               int timeBin ):
  RooAbsPdf(name,title),
  S1("S1","S1",this,_S1),
  log10S2("log10S2","log10S2",this,_log10S2),
  drift("drift","drift",this,_drift),
  phi("phi","phi",this,_phi),
  r("r","r",this,_r)
```

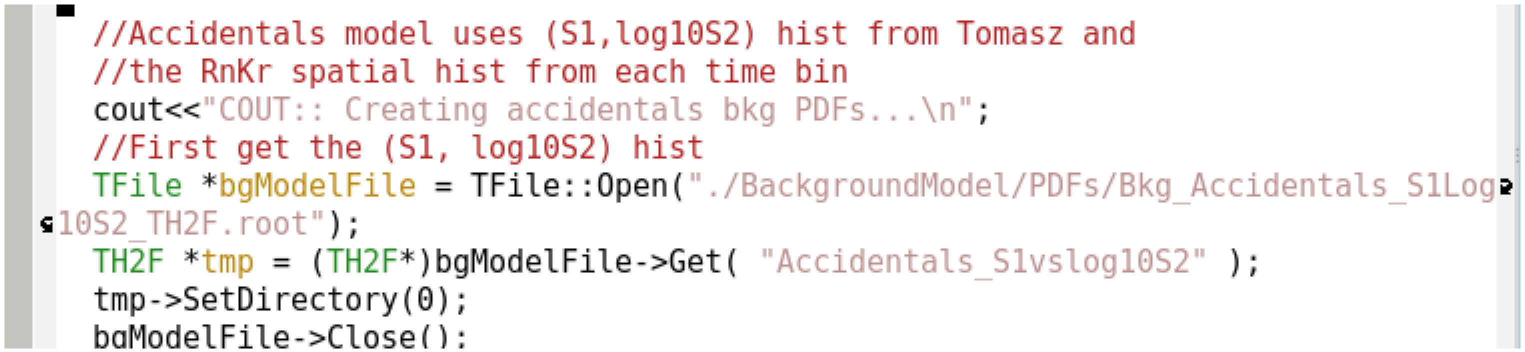
- Initialize roofit variables to fed-in values



```
        thisTimeBin=timeBin;
        char wallInput[100];
        sprintf(wallInput,"WallModel/TB%d_wallws_4x1.root",thisTimeBin); //For r
unning in SIRun4 Directory
        //sprintf(wallInput,"TB%d_wallws_4x1.root",thisTimeBin); //For testing i
n WallModel Directory.
        TFile fin(wallInput,"OPEN");
        RooWorkspace* wws = (RooWorkspace*)fin.Get("wws");
        DataHist = (RooDataHist*)wws->data("FullHist");
        DataHist->Print();
```

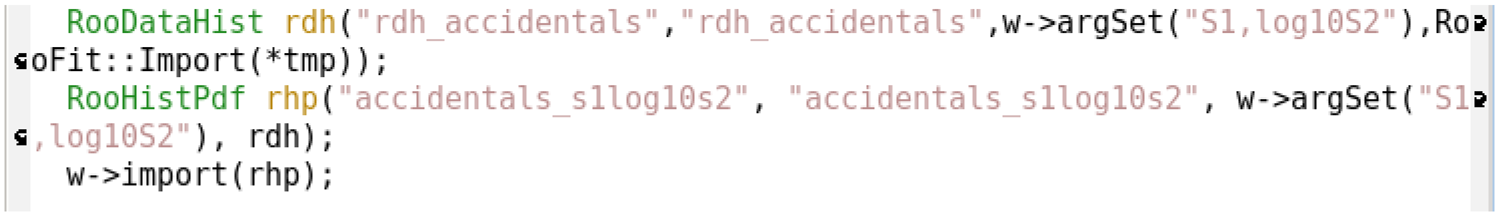
- Open the appropriate wall model file.
- Retrieve the workspace from the file
- Retrieve a histogram with the background model from the workspace.

ImportAccidentalModel



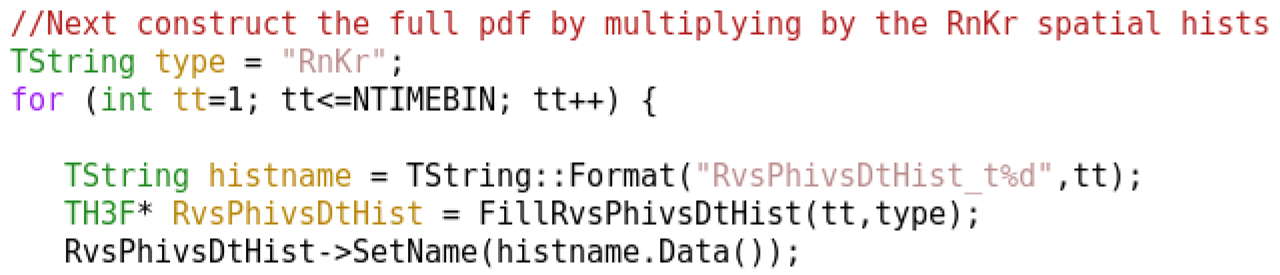
```
//Accidentals model uses (S1,log10S2) hist from Tomasz and
//the RnKr spatial hist from each time bin
cout<<"COUT:: Creating accidentals bkg PDFs...\n";
//First get the (S1, log10S2) hist
TFile *bgModelFile = TFile::Open("./BackgroundModel/PDFs/Bkg_Accidentals_S1Log
10S2_TH2F.root");
TH2F *tmp = (TH2F*)bgModelFile->Get( "Accidentals_S1vslog10S2" );
tmp->SetDirectory(0);
bgModelFile->Close();
```

- Opens the file containing the accidental S1 vs S2 model.
- Gets a 2D histogram with the model from the file (called “tmp”)



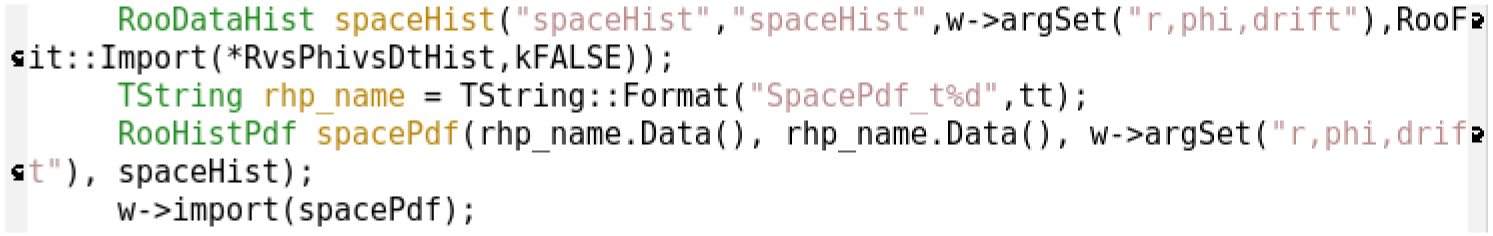
```
   RooDataHist rdh("rdh_accidentals","rdh_accidentals",w->argSet("S1,log10S2"),Ro
oFit::Import(*tmp));
   RooHistPdf rhp("accidentals_s1log10s2", "accidentals_s1log10s2", w->argSet("S1
,log10S2"), rdh);
   w->import(rhp);
```

- Creates a PDF out of the previously loaded root histogram and adds it to the workspace.
  - Does this by creating a data hist out of the histogram, then a rooHistPdf out of that dataHist.



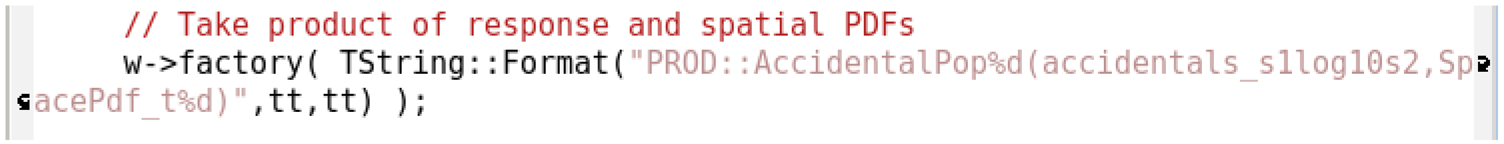
```
//Next construct the full pdf by multiplying by the RnKr spatial hists
TString type = "RnKr";
for (int tt=1; tt<=NTIMEBIN; tt++) {

   TString histname = TString::Format("RvsPhivsDtHist_t%d",tt);
   TH3F* RvsPhivsDtHist = FillRvsPhivsDtHist(tt,type);
   RvsPhivsDtHist->SetName(histname.Data());
```

- Creates a 3D histogram (intended for use as the spatial pdf) for each time bin.
- Fills these histograms using the “FillRvsPhivsDtHist” method.
  - Uses the same spatial distribution as RnKr which it feeds to this method as an argument.



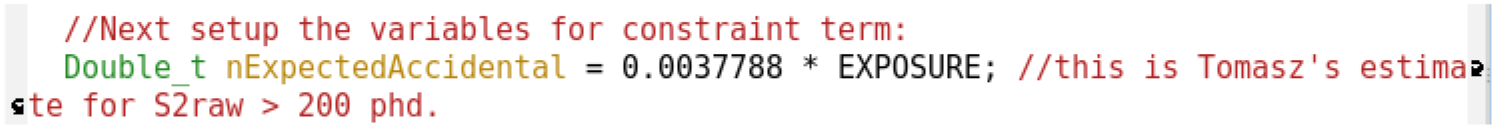
```
     RooDataHist spaceHist("spaceHist","spaceHist",w->argSet("r,phi,drift"),RooF
it::Import(*RvsPhivsDtHist,kFALSE));
     TString rhp_name = TString::Format("SpacePdf_t%d",tt);
     RooHistPdf spacePdf(rhp_name.Data(), rhp_name.Data(), w->argSet("r,phi,drif
t"), spaceHist);
     w->import(spacePdf);
```

- Creates a roofit pdf (of type HistPdf) out of this 3d histogram.
- Imports this pdf to the workspace.



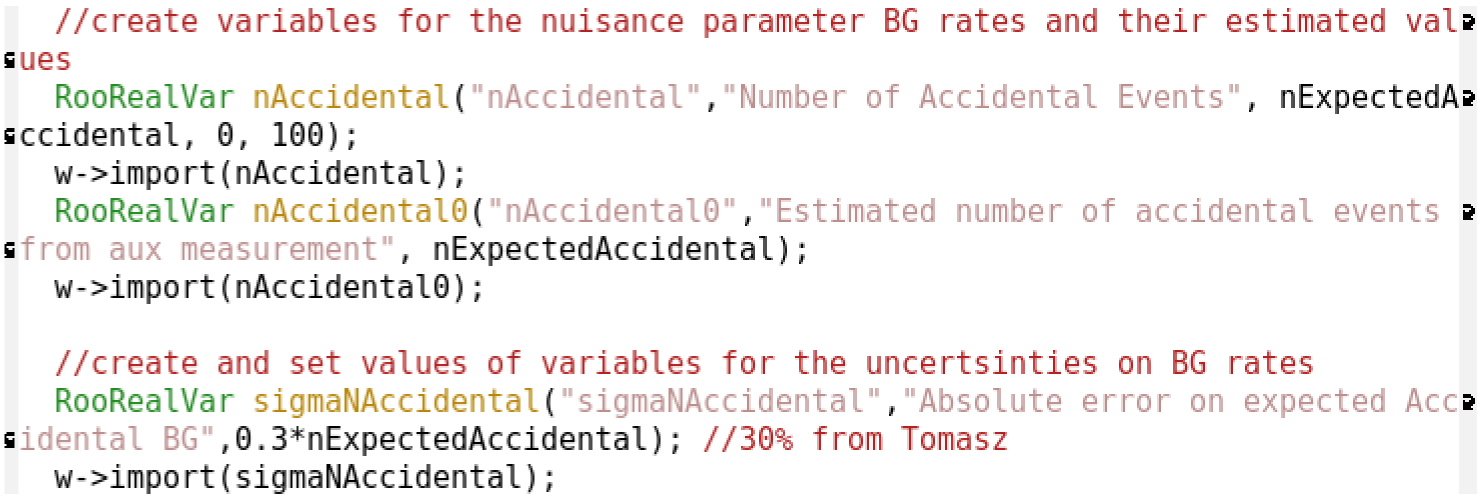
```
     // Take product of response and spatial PDFs
     w->factory( TString::Format("PROD::AccidentalPop%d(accidentals_s1log10s2,Sp
acePdf_t%d)",tt,tt) );
```

- Creates a full 5D pdf (called AccidentalPop[time bin]) by taking the direct product of the accidentals pdf and the spatial pdf just created.



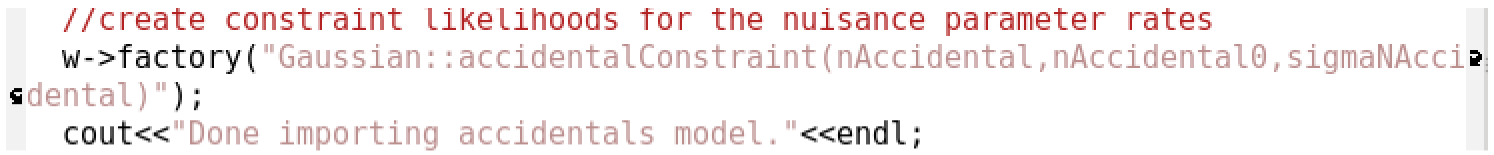
```
   //Next setup the variables for constraint term:
   Double_t nExpectedAccidental = 0.0037788 * EXPOSURE; //this is Tomasz's estima
te for S2raw > 200 phd.
```

- Hard coded estimate of the number of expected accidental background events.



```
  //create variables for the nuisance parameter BG rates and their estimated val
ues
  RooRealVar nAccidental("nAccidental","Number of Accidental Events", nExpectedA
ccidental, 0, 100);
  w->import(nAccidental);
  RooRealVar nAccidental0("nAccidental0","Estimated number of accidental events 
from aux measurement", nExpectedAccidental);
  w->import(nAccidental0);

  //create and set values of variables for the uncertsinties on BG rates
  RooRealVar sigmaNAccidental("sigmaNAccidental","Absolute error on expected Acc
idental BG",0.3*nExpectedAccidental); //30% from Tomasz
  w->import(sigmaNAccidental);
```

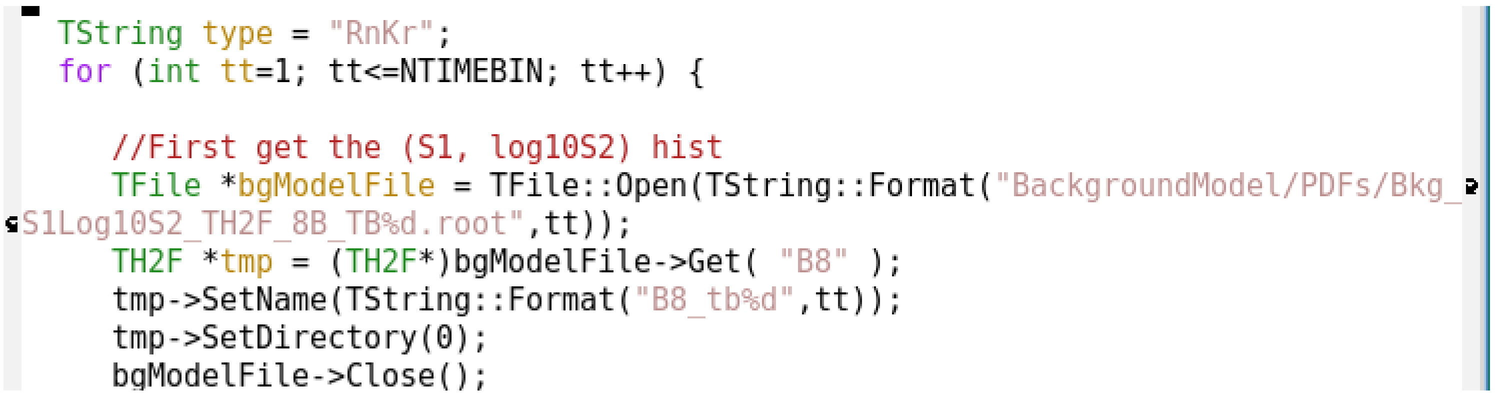
- Create roofit variables for the number of accidental events, the mean, and the deviation of the gaussian constraint and add them to the workspace



```
  //create constraint likelihoods for the nuisance parameter rates
  w->factory("Gaussian::accidentalConstraint(nAccidental,nAccidental0,sigmaNAcci
dental)");
  cout<<"Done importing accidentals model."<<endl;
```

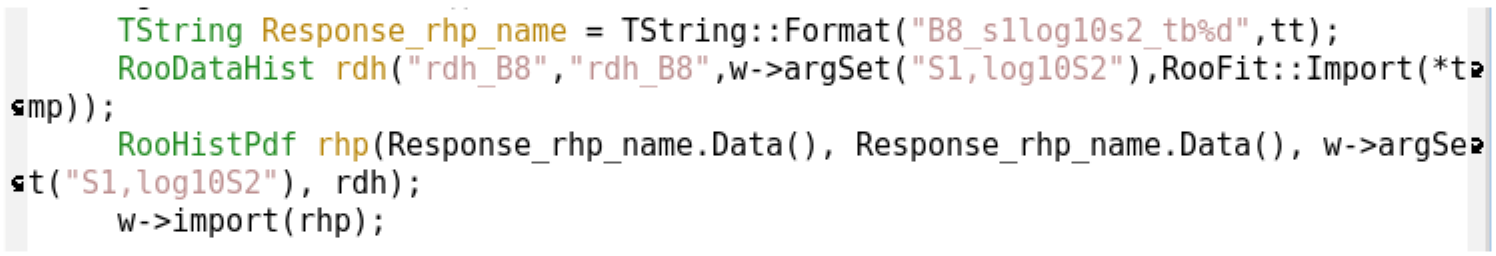
- Create a gaussian pdf (called "accidentalConstraint") based on the estimates of the number of accidental events and the constraints and add it to the workspace.

Import8BModel



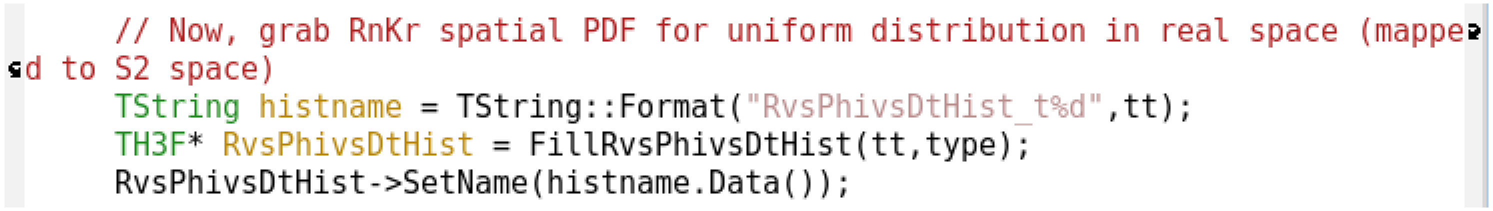
```
TString type = "RnKr";
for (int tt=1; tt<=NTIMEBIN; tt++) {

    //First get the (S1, log10S2) hist
    TFile *bgModelFile = TFile::Open(TString::Format("BackgroundModel/PDFs/Bkg_
S1Log10S2_TH2F_8B_TB%d.root",tt));
    TH2F *tmp = (TH2F*)bgModelFile->Get( "B8" );
    tmp->SetName(TString::Format("B8_tb%d",tt));
    tmp->SetDirectory(0);
    bgModelFile->Close();
```

- For each time bin, opens the appropriate root file with the background model.
- Gets the 2D histogram with the B8 S1 vs S2 model from the file (names it tmp)



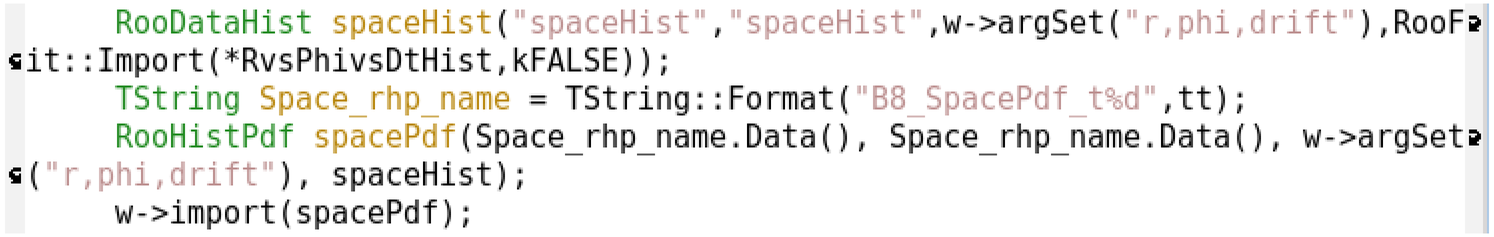
```
      TString Response_rhp_name = TString::Format("B8_s1log10s2_tb%d",tt);
      RooDataHist rdh("rdh_B8","rdh_B8",w->argSet("S1,log10S2"),RooFit::Import(*t
mp));
      RooHistPdf rhp(Response_rhp_name.Data(), Response_rhp_name.Data(), w->argSe
t("S1,log10S2"), rdh);
      w->import(rhp);
```

- Create a roofit pdf out of this histogram and add it to the workspace (called rhp)



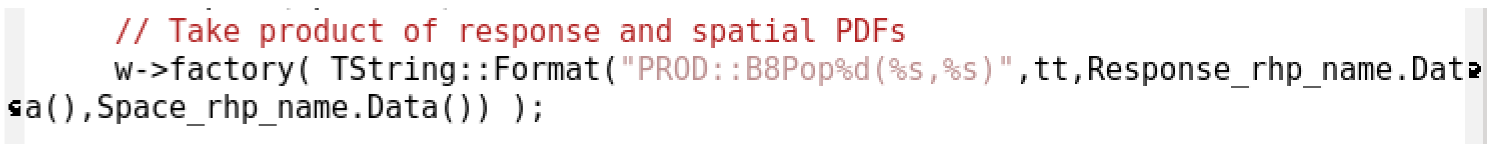
```
      // Now, grab RnKr spatial PDF for uniform distribution in real space (mappe
d to S2 space)
      TString histname = TString::Format("RvsPhivsDtHist_t%d",tt);
      TH3F* RvsPhivsDtHist = FillRvsPhivsDtHist(tt,type);
      RvsPhivsDtHist->SetName(histname.Data());
```

- Creates a 3D histogram (intended for use as the spatial pdf) for each time bin.
- Fills these histograms using the “FillRvsPhivsDtHist” method.
  – Uses the same spatial distribution as RnKr which it feeds to this method as an argument.



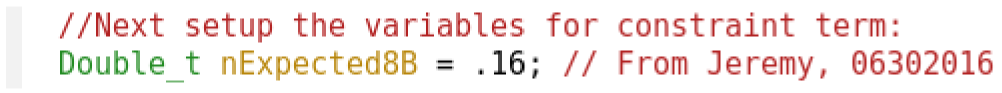
```
     RooDataHist spaceHist("spaceHist","spaceHist",w->argSet("r,phi,drift"),RooF
it::Import(*RvsPhivsDtHist,kFALSE));
     TString Space_rhp_name = TString::Format("B8_SpacePdf_t%d",tt);
     RooHistPdf spacePdf(Space_rhp_name.Data(), Space_rhp_name.Data(), w->argSet
("r,phi,drift"), spaceHist);
     w->import(spacePdf);
```

- Creates a roofit pdf (of type HistPdf) out of this 3d histogram.
- Imports this pdf to the workspace.



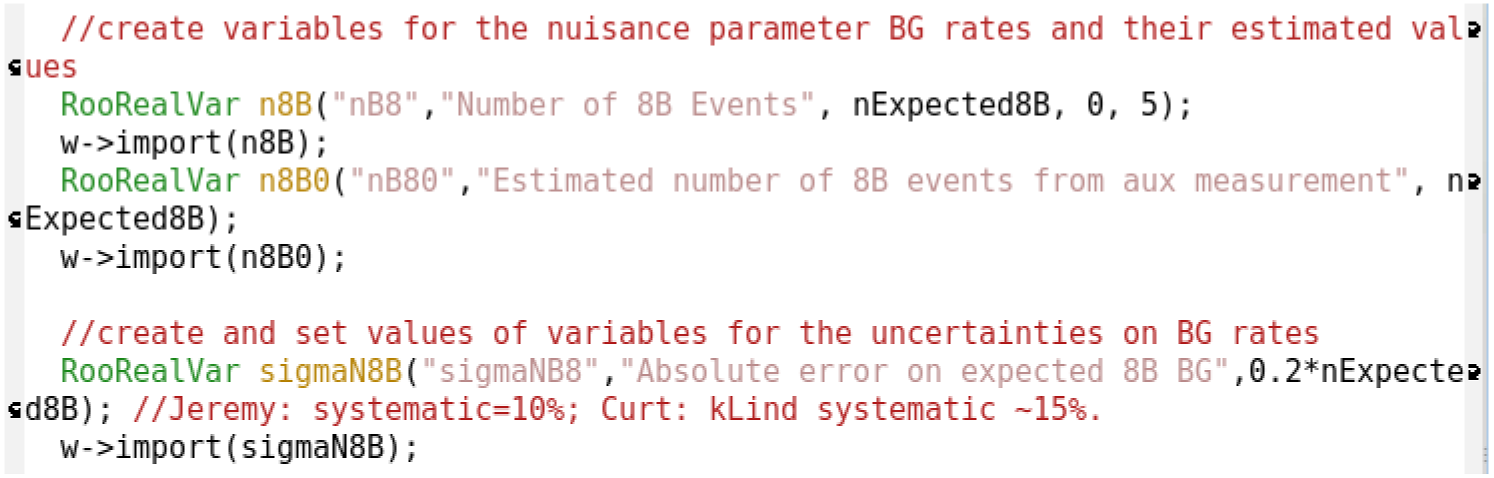
```
      // Take product of response and spatial PDFs
      w->factory( TString::Format("PROD::B8Pop%d(%s,%s)",tt,Response_rhp_name.Dat
a(),Space_rhp_name.Data()) );
```

- Creates a 5D pdf (called B8Pop) by taking the direct product of the two histograms just created.



```
//Next setup the variables for constraint term:
Double_t nExpected8B = .16; // From Jeremy, 06302016
```

- Hard coded estimate of the number of expected 8B background events.



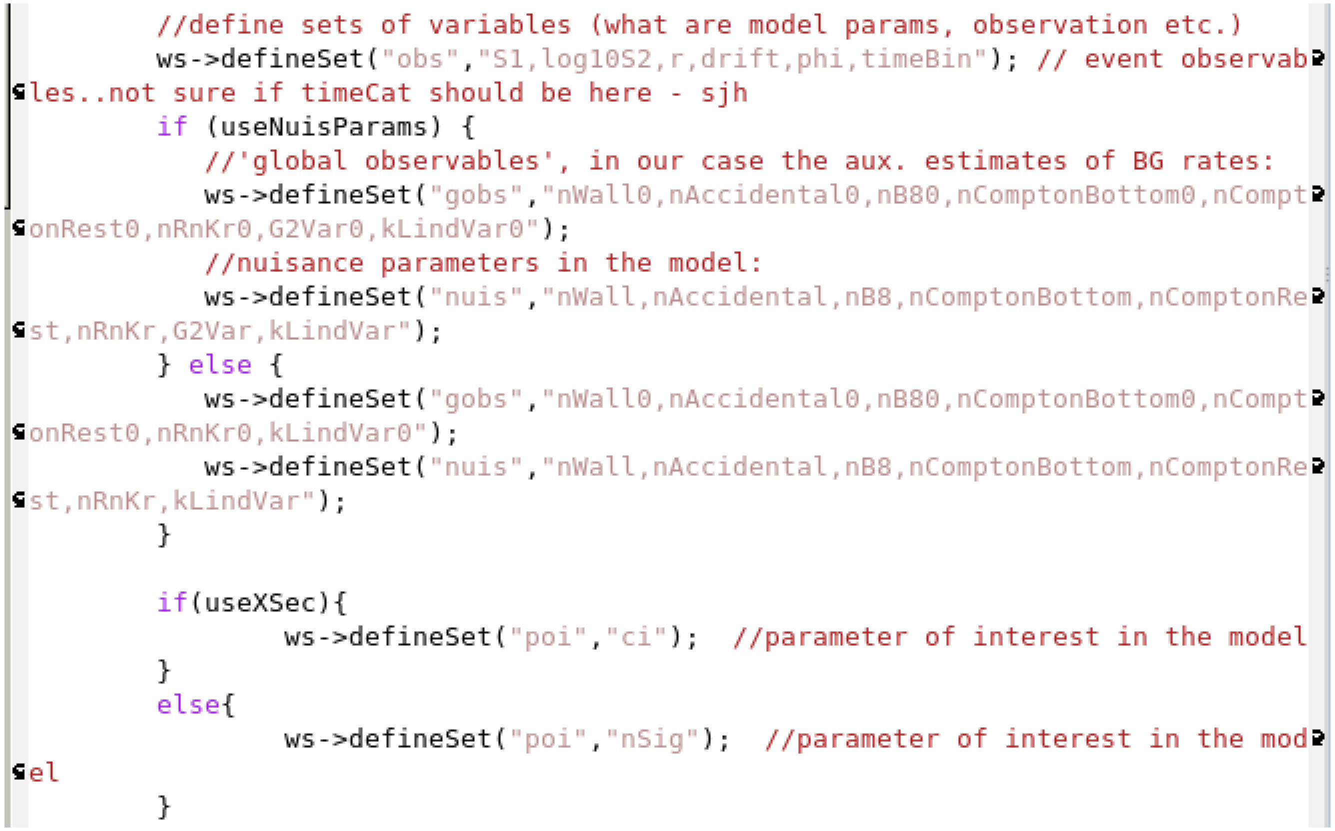
```
   //create variables for the nuisance parameter BG rates and their estimated val
ues
   RooRealVar n8B("nB8","Number of 8B Events", nExpected8B, 0, 5);
   w->import(n8B);
   RooRealVar n8B0("nB80","Estimated number of 8B events from aux measurement", n
Expected8B);
   w->import(n8B0);

   //create and set values of variables for the uncertainties on BG rates
   RooRealVar sigmaN8B("sigmaNB8","Absolute error on expected 8B BG",0.2*nExpecte
d8B); //Jeremy: systematic=10%; Curt: kLind systematic ~15%.
   w->import(sigmaN8B);
```

- Create roofit variables for the number of background vents, and the mean expected number and deviation for a gaussian constraint.
- Import these variables into the workspace.

```
//create constraint likelihoods for the nuisance parameter rates
w->factory("Gaussian::B8Constraint(nB8,nB80,sigmaNB8)");

cout<<"Done importing 8Bs model."<<endl;
```

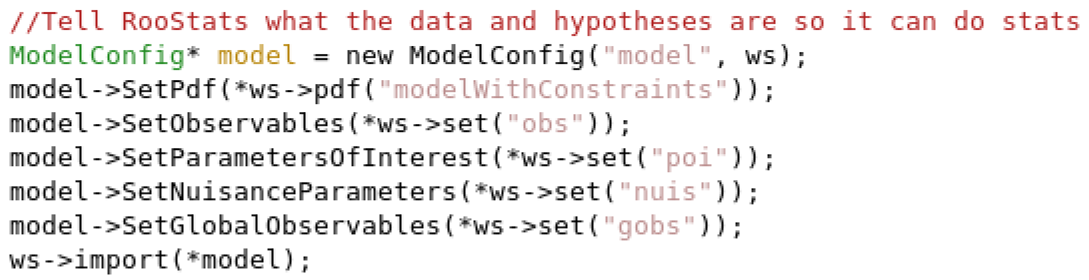
- Create a pdf (called B8Constraint) that is meant to constrain the number of these background events.

ConfigureModel



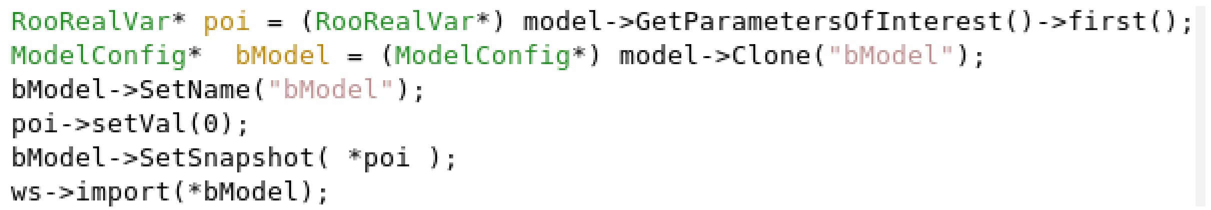
```
        //define sets of variables (what are model params, observation etc.)
        ws->defineSet("obs","S1,log10S2,r,drift,phi,timeBin"); // event observab
les..not sure if timeCat should be here - sjh
        if (useNuisParams) {
           //'global observables', in our case the aux. estimates of BG rates:
           ws->defineSet("gobs","nWall0,nAccidental0,nB80,nComptonBottom0,nCompt
onRest0,nRnKr0,G2Var0,kLindVar0");
           //nuisance parameters in the model:
           ws->defineSet("nuis","nWall,nAccidental,nB8,nComptonBottom,nComptonRe
st,nRnKr,G2Var,kLindVar");
        } else {
           ws->defineSet("gobs","nWall0,nAccidental0,nB80,nComptonBottom0,nCompt
onRest0,nRnKr0,kLindVar0");
           ws->defineSet("nuis","nWall,nAccidental,nB8,nComptonBottom,nComptonRe
st,nRnKr,kLindVar");
        }

        if(useXSec){
                ws->defineSet("poi","ci");  //parameter of interest in the model
        }
        else{
                ws->defineSet("poi","nSig");  //parameter of interest in the mod
el
        }
```

- Define sets of variables (not sure what this means, just copying comment ☺)
  - Event observables (things events consist of)
  - Global observables (expected numbers for each nuisance parameter)
  - Nuisance parameters (the parameters themselves)
  - Parameters of Interest



```
//Tell RooStats what the data and hypotheses are so it can do stats
ModelConfig* model = new ModelConfig("model", ws);
model->SetPdf(*ws->pdf("modelWithConstraints"));
model->SetObservables(*ws->set("obs"));
model->SetParametersOfInterest(*ws->set("poi"));
model->SetNuisanceParameters(*ws->set("nuis"));
model->SetGlobalObservables(*ws->set("gobs"));
ws->import(*model);
```

- Create a ModelConfig object that knows about each of the things we just defined.
- It apparently has functions called “setX” which is how it knows what is what and what it can do with what.

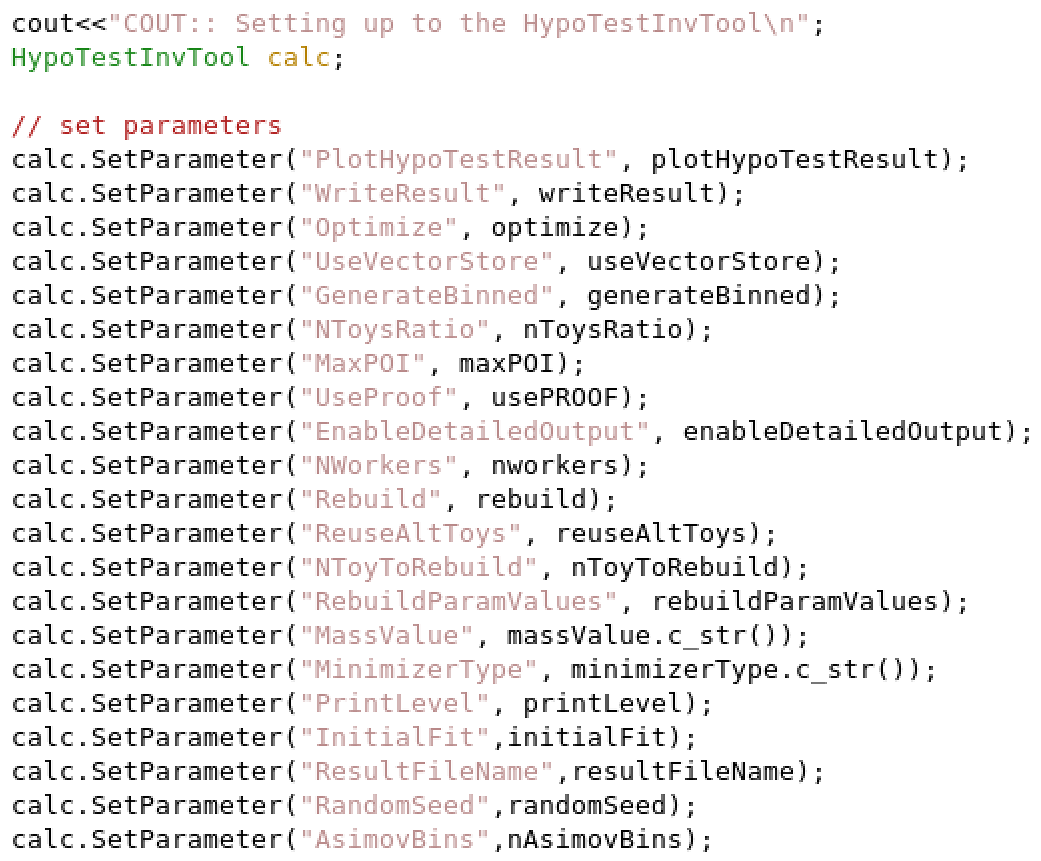


```
RooRealVar* poi = (RooRealVar*) model->GetParametersOfInterest()->first();
ModelConfig*  bModel = (ModelConfig*) model->Clone("bModel");
bModel->SetName("bModel");
poi->setVal(0);
bModel->SetSnapshot( *poi );
ws->import(*bModel);
```

- No idea about the first line. Apparently returns a pointer to the parameter of interest (the first parameter of interest? Can it take multiple? How many if that is what first refers to? Presumably not arbitrarily many if there is a separate function to access each).
- Creates a model called “bModel.” Sets poi to 0 and adds it to the workspace.
- Not sure what SetSnapshot is all about.

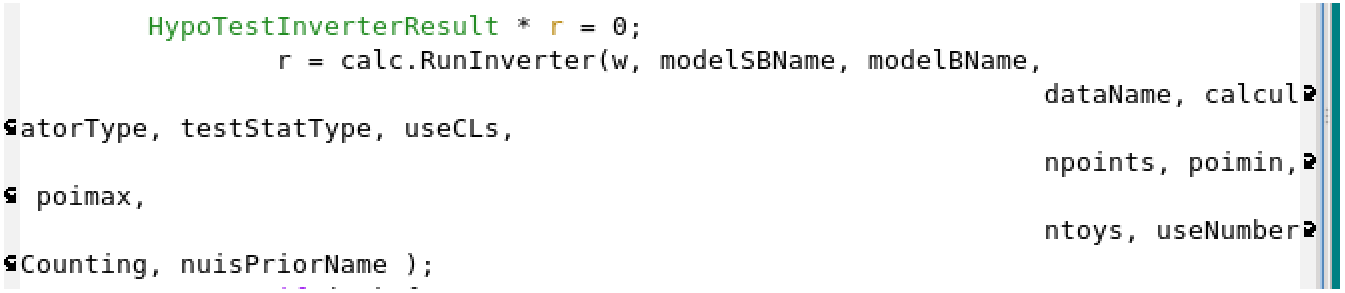
StandardHypoTestInv.C

- StandardHypoTestInvDemo



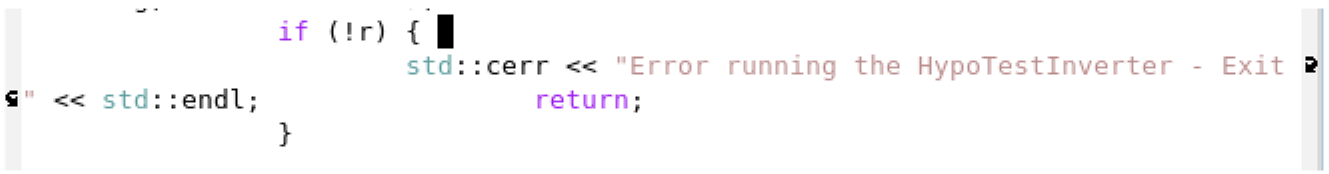
```
cout<<"COUT:: Setting up to the HypoTestInvTool\n";
HypoTestInvTool calc;

// set parameters
calc.SetParameter("PlotHypoTestResult", plotHypoTestResult);
calc.SetParameter("WriteResult", writeResult);
calc.SetParameter("Optimize", optimize);
calc.SetParameter("UseVectorStore", useVectorStore);
calc.SetParameter("GenerateBinned", generateBinned);
calc.SetParameter("NToysRatio", nToysRatio);
calc.SetParameter("MaxPOI", maxPOI);
calc.SetParameter("UseProof", usePROOF);
calc.SetParameter("EnableDetailedOutput", enableDetailedOutput);
calc.SetParameter("NWorkers", nworkers);
calc.SetParameter("Rebuild", rebuild);
calc.SetParameter("ReuseAltToys", reuseAltToys);
calc.SetParameter("NToyToRebuild", nToyToRebuild);
calc.SetParameter("RebuildParamValues", rebuildParamValues);
calc.SetParameter("MassValue", massValue.c_str());
calc.SetParameter("MinimizerType", minimizerType.c_str());
calc.SetParameter("PrintLevel", printLevel);
calc.SetParameter("InitialFit",initialFit);
calc.SetParameter("ResultFileName",resultFileName);
calc.SetParameter("RandomSeed",randomSeed);
calc.SetParameter("AsimovBins",nAsimovBins);
```

- Creates an instance “calc” of the HypoTestInvTool class.
- Sets relevant parameters of the object.



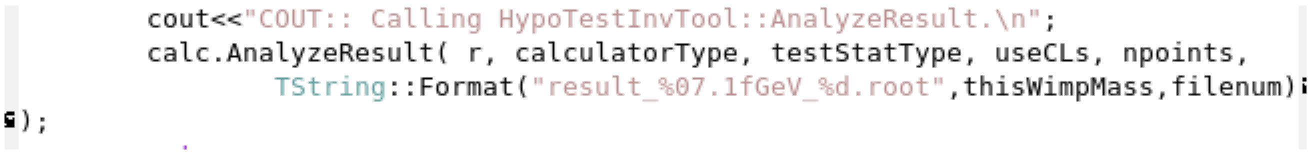
```
        HypoTestInverterResult * r = 0;
                r = calc.RunInverter(w, modelSBName, modelBName,
                                                                        dataName, calcul
atorType, testStatType, useCLs,
                                                                        npoints, poimin,
 poimax,
                                                                        ntoys, useNumber
Counting, nuisPriorName );
```

- Run the HypoTestInverter and stick the result in a HypoTestInverterResult called “r”



```
               if (!r) {
                       std::cerr << "Error running the HypoTestInverter - Exit " << std::endl;
                       return;
               }
```

- Spits out an error message if running the hypoTestInverter (or saving the result) didn’t work.

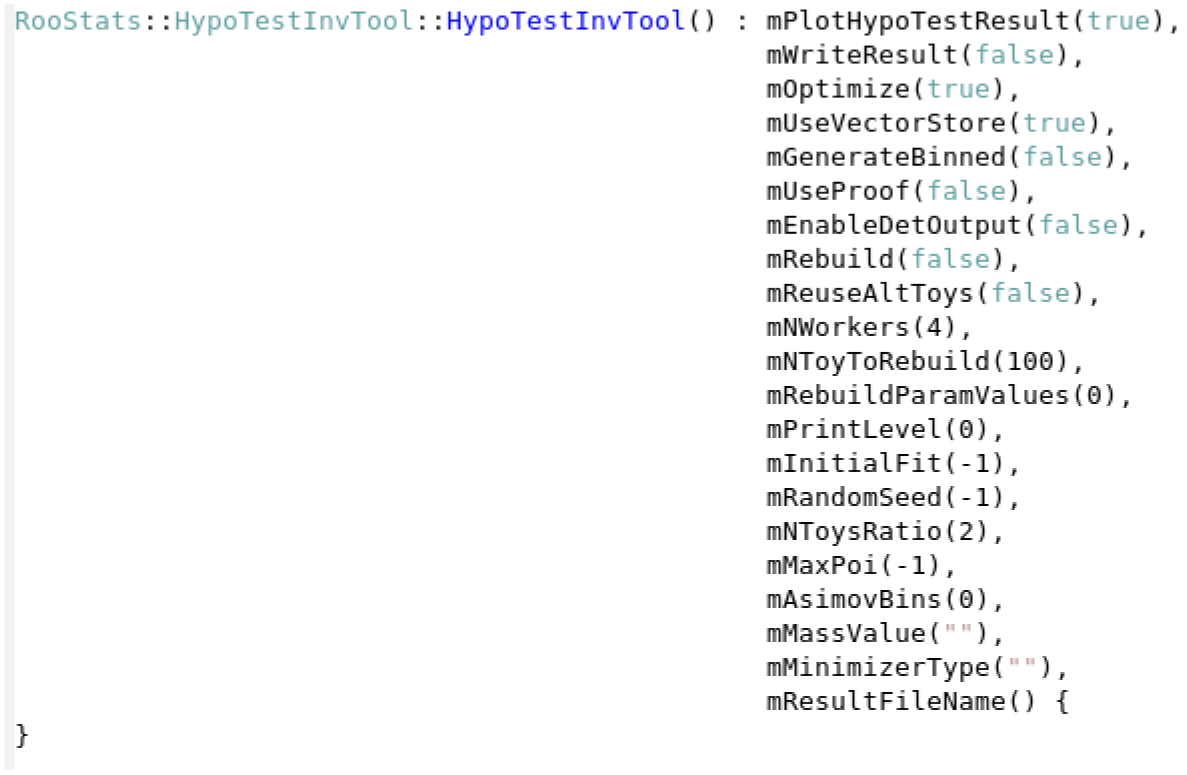


```
        cout<<"COUT:: Calling HypoTestInvTool::AnalyzeResult.\n";
        calc.AnalyzeResult( r, calculatorType, testStatType, useCLs, npoints,
                TString::Format("result_%07.1fGeV_%d.root",thisWimpMass,filenum)
s);
```

- Analyze the reults using the “AnalyzeResult” function from the HypoTestInverterTool.

StandardHypoTestInv.C

- HypoTestInvTool - constructor



```cpp
RooStats::HypoTestInvTool::HypoTestInvTool() : mPlotHypoTestResult(true),
                                               mWriteResult(false),
                                               mOptimize(true),
                                               mUseVectorStore(true),
                                               mGenerateBinned(false),
                                               mUseProof(false),
                                               mEnableDetOutput(false),
                                               mRebuild(false),
                                               mReuseAltToys(false),
                                               mNWorkers(4),
                                               mNToyToRebuild(100),
                                               mRebuildParamValues(0),
                                               mPrintLevel(0),
                                               mInitialFit(-1),
                                               mRandomSeed(-1),
                                               mNToysRatio(2),
                                               mMaxPoi(-1),
                                               mAsimovBins(0),
                                               mMassValue(""),
                                               mMinimizerType(""),
                                               mResultFileName() {
}
```

- Just initialize a bunch of variables.

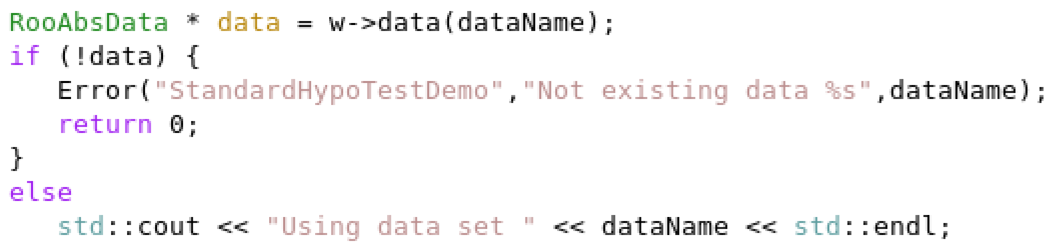
StandardHypoTestInv.C

- HypoTestInvTool::SetParameter

Just looks for the parameter given as a string and changes it to the value passed.

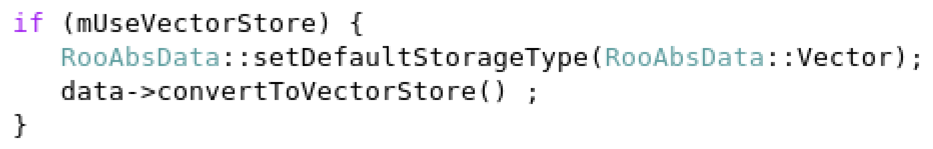
StandardHypoTestInv.C

- HypoTestInvTool::RunInverter



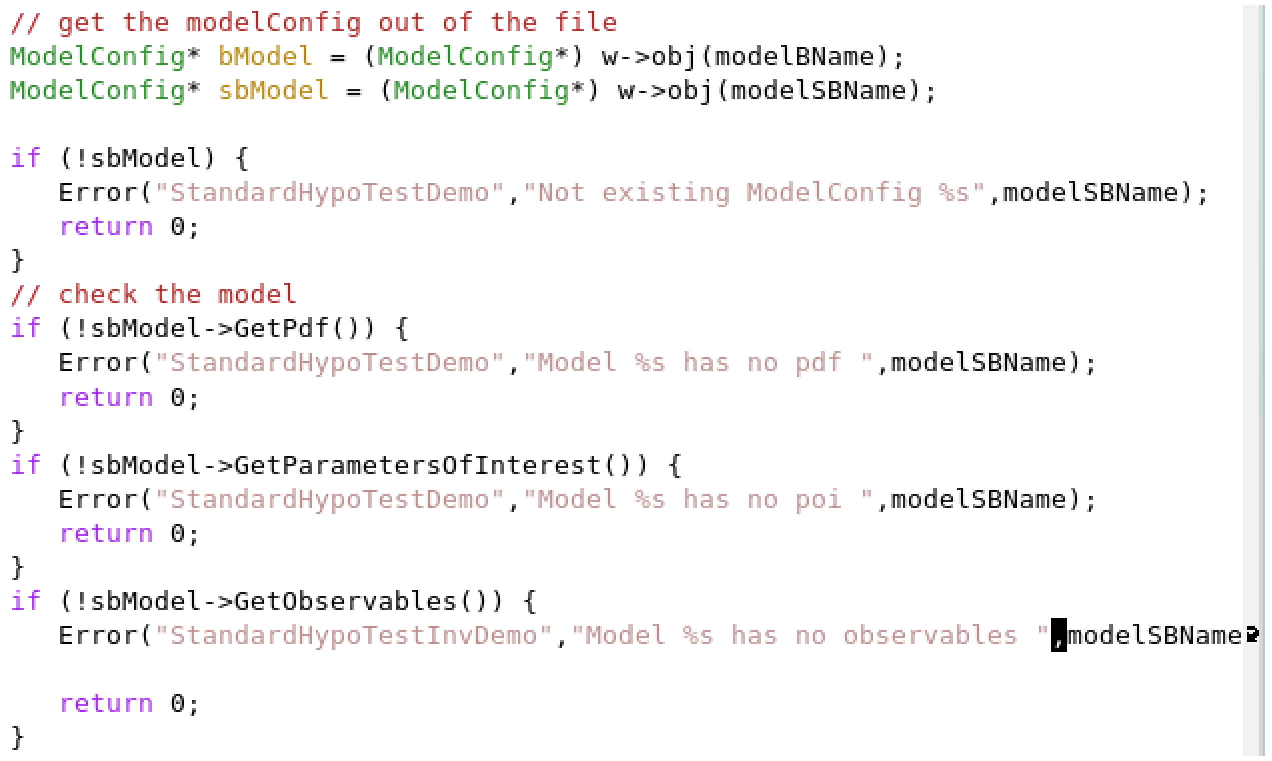
```
RooAbsData * data = w->data(dataName);
if (!data) {
   Error("StandardHypoTestDemo","Not existing data %s",dataName);
   return 0;
}
else
   std::cout << "Using data set " << dataName << std::endl;
```

- Loads the data to be used from the workspace
- Spits out an error if it can’t find it, otherwise, tells you the name of the data set it’s using



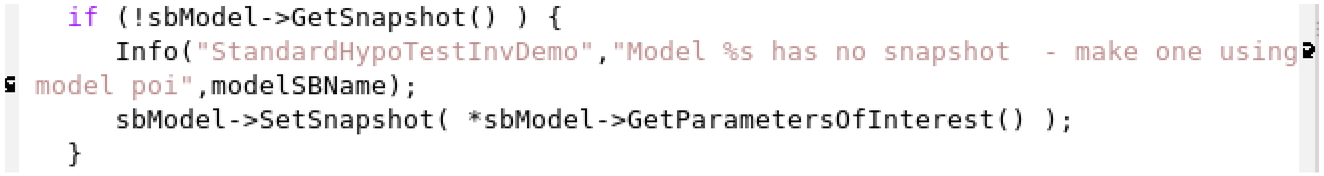
```
if (mUseVectorStore) {
   RooAbsData::setDefaultStorageType(RooAbsData::Vector);
   data->convertToVectorStore() ;
}
```

- ¯_(ツ)_/¯



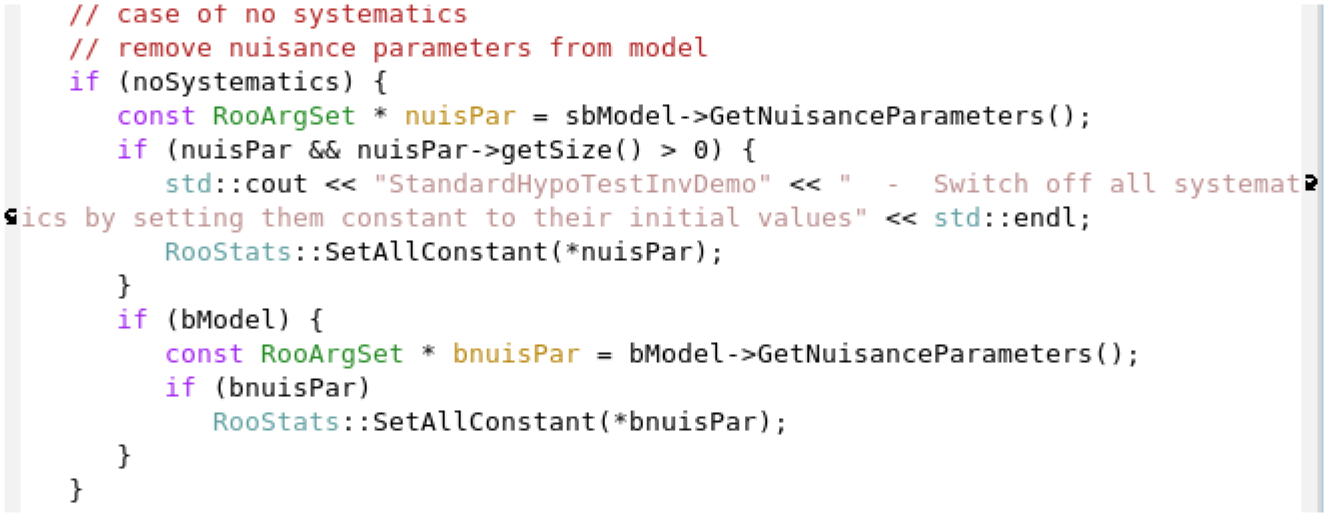
```
// get the modelConfig out of the file
ModelConfig* bModel = (ModelConfig*) w->obj(modelBName);
ModelConfig* sbModel = (ModelConfig*) w->obj(modelSBName);

if (!sbModel) {
   Error("StandardHypoTestDemo","Not existing ModelConfig %s",modelSBName);
   return 0;
}
// check the model
if (!sbModel->GetPdf()) {
   Error("StandardHypoTestDemo","Model %s has no pdf ",modelSBName);
   return 0;
}
if (!sbModel->GetParametersOfInterest()) {
   Error("StandardHypoTestDemo","Model %s has no poi ",modelSBName);
   return 0;
}
if (!sbModel->GetObservables()) {
   Error("StandardHypoTestInvDemo","Model %s has no observables ",modelSBName

   return 0;
}
```

- Loads in the background model (bModel) and the signal + background model (sbModel) from the workspace
- Checks to make sure that the signal + bkg model has all the components it's supposed to have



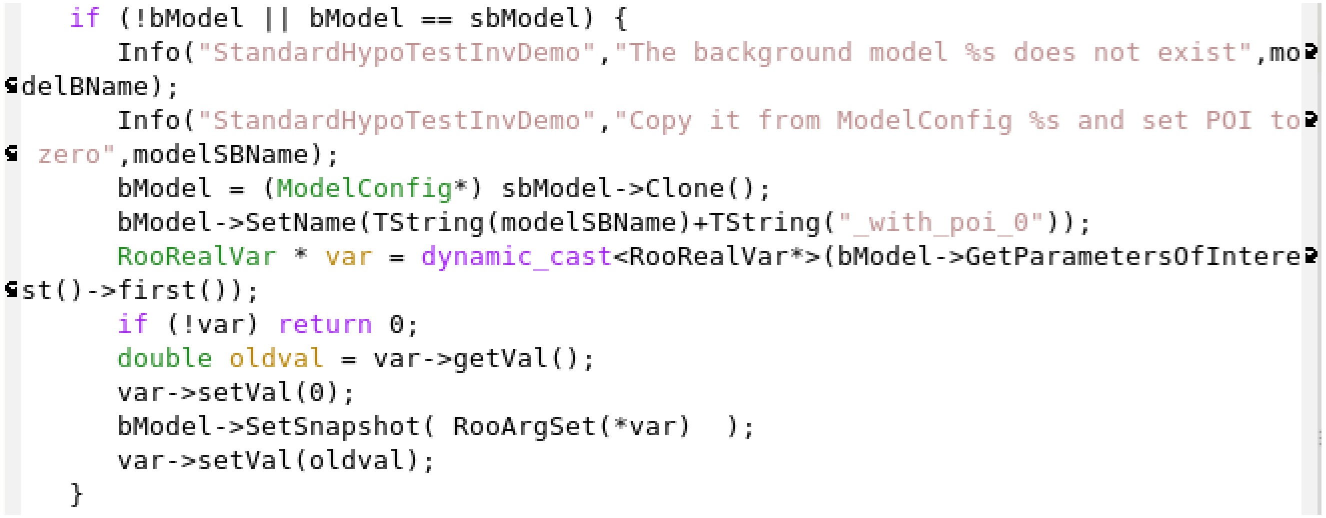
```
   if (!sbModel->GetSnapshot() ) {
      Info("StandardHypoTestInvDemo","Model %s has no snapshot  - make one using
 model poi",modelSBName);
      sbModel->SetSnapshot( *sbModel->GetParametersOfInterest() );
   }
```

- Generates a snapshot if the s+b model doesn’t already have one.
  - I have no idea what this actually means, however.



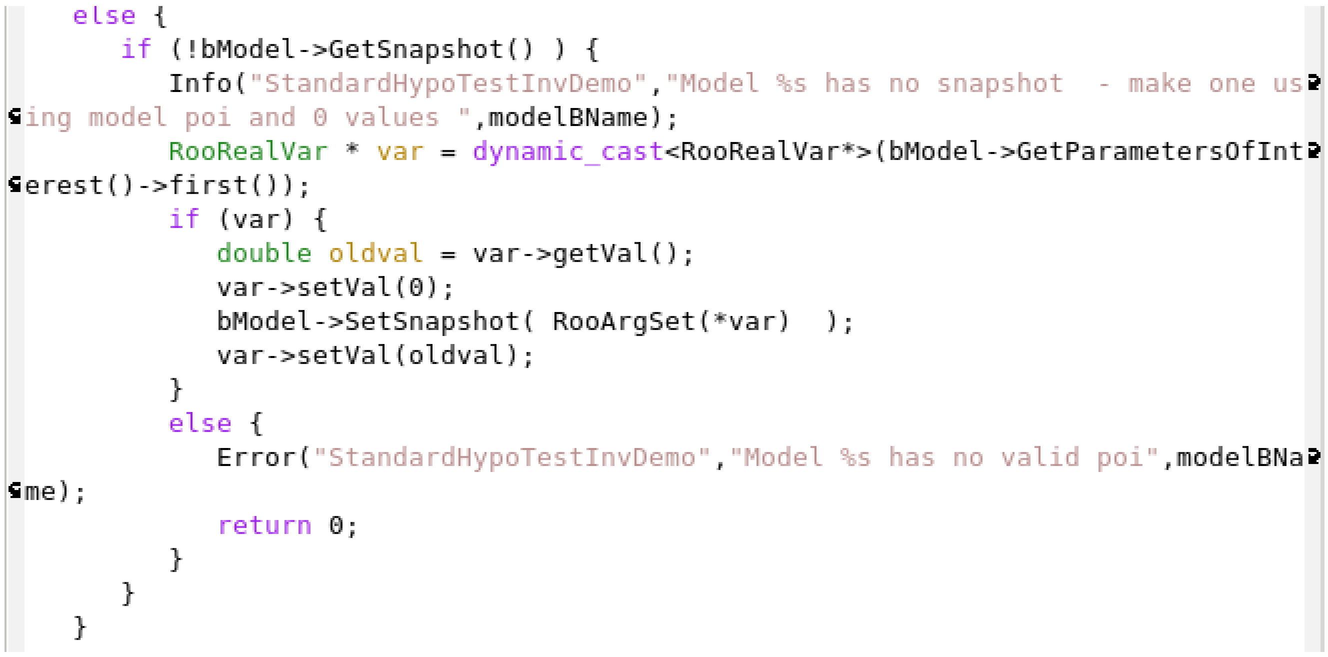
```
    // case of no systematics
    // remove nuisance parameters from model
    if (noSystematics) {
       const RooArgSet * nuisPar = sbModel->GetNuisanceParameters();
       if (nuisPar && nuisPar->getSize() > 0) {
          std::cout << "StandardHypoTestInvDemo" << "  -  Switch off all systemat
ics by setting them constant to their initial values" << std::endl;
          RooStats::SetAllConstant(*nuisPar);
       }
       if (bModel) {
          const RooArgSet * bnuisPar = bModel->GetNuisanceParameters();
          if (bnuisPar)
             RooStats::SetAllConstant(*bnuisPar);
       }
    }
```

- Sets all nuisance parameters to constant if the “noSystematics” flag is set true.



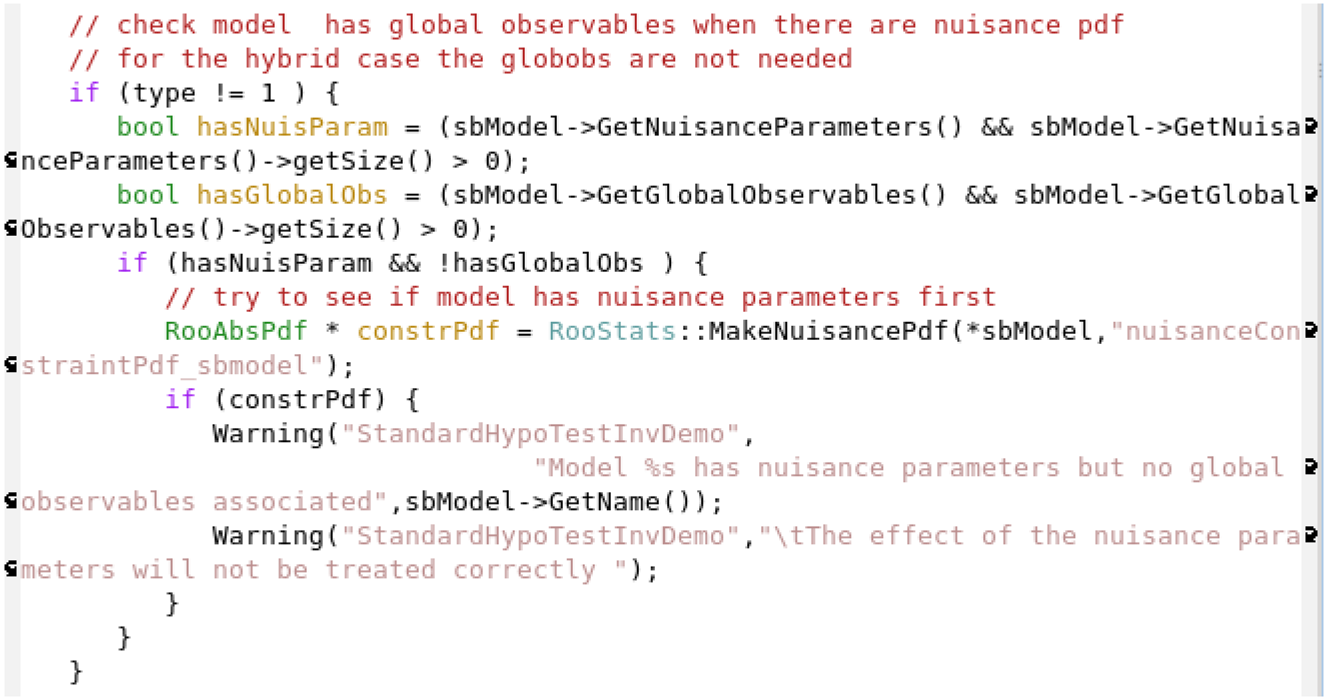
```
    if (!bModel || bModel == sbModel) {
       Info("StandardHypoTestInvDemo","The background model %s does not exist",modelBName);
       Info("StandardHypoTestInvDemo","Copy it from ModelConfig %s and set POI to zero",modelSBName);
       bModel = (ModelConfig*) sbModel->Clone();
       bModel->SetName(TString(modelSBName)+TString("_with_poi_0"));
       RooRealVar * var = dynamic_cast<RooRealVar*>(bModel->GetParametersOfInterest()->first());
       if (!var) return 0;
       double oldval = var->getVal();
       var->setVal(0);
       bModel->SetSnapshot( RooArgSet(*var)  );
       var->setVal(oldval);
    }
```

- If there is no background model (or the same model was provided as both signal and background), generate one by cloning the s+b model and setting poi to 0.
- Set snapshot of the background model (whatever that means) with poi set to 0, then reset to old value (not sure why this is done (maybe because it’s a pointer and so leaving it 0 would damage the s+b model)).
- If the s+b model had poi set to 0, just quit.



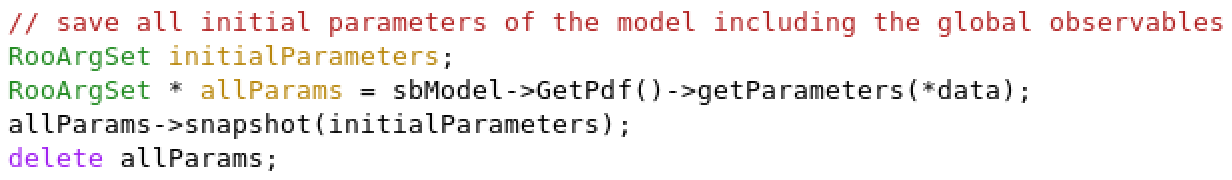
```
    else {
       if (!bModel->GetSnapshot() ) {
          Info("StandardHypoTestInvDemo","Model %s has no snapshot  - make one us
ing model poi and 0 values ",modelBName);
          RooRealVar * var = dynamic_cast<RooRealVar*>(bModel->GetParametersOfInt
erest()->first());
          if (var) {
             double oldval = var->getVal();
             var->setVal(0);
             bModel->SetSnapshot( RooArgSet(*var)  );
             var->setVal(oldval);
          }
          else {
             Error("StandardHypoTestInvDemo","Model %s has no valid poi",modelBNa
me);
             return 0;
          }
       }
    }
```

- If background model doesn’t have a snapshot already, generate one by taking one of the s+b model with poi set to 0.



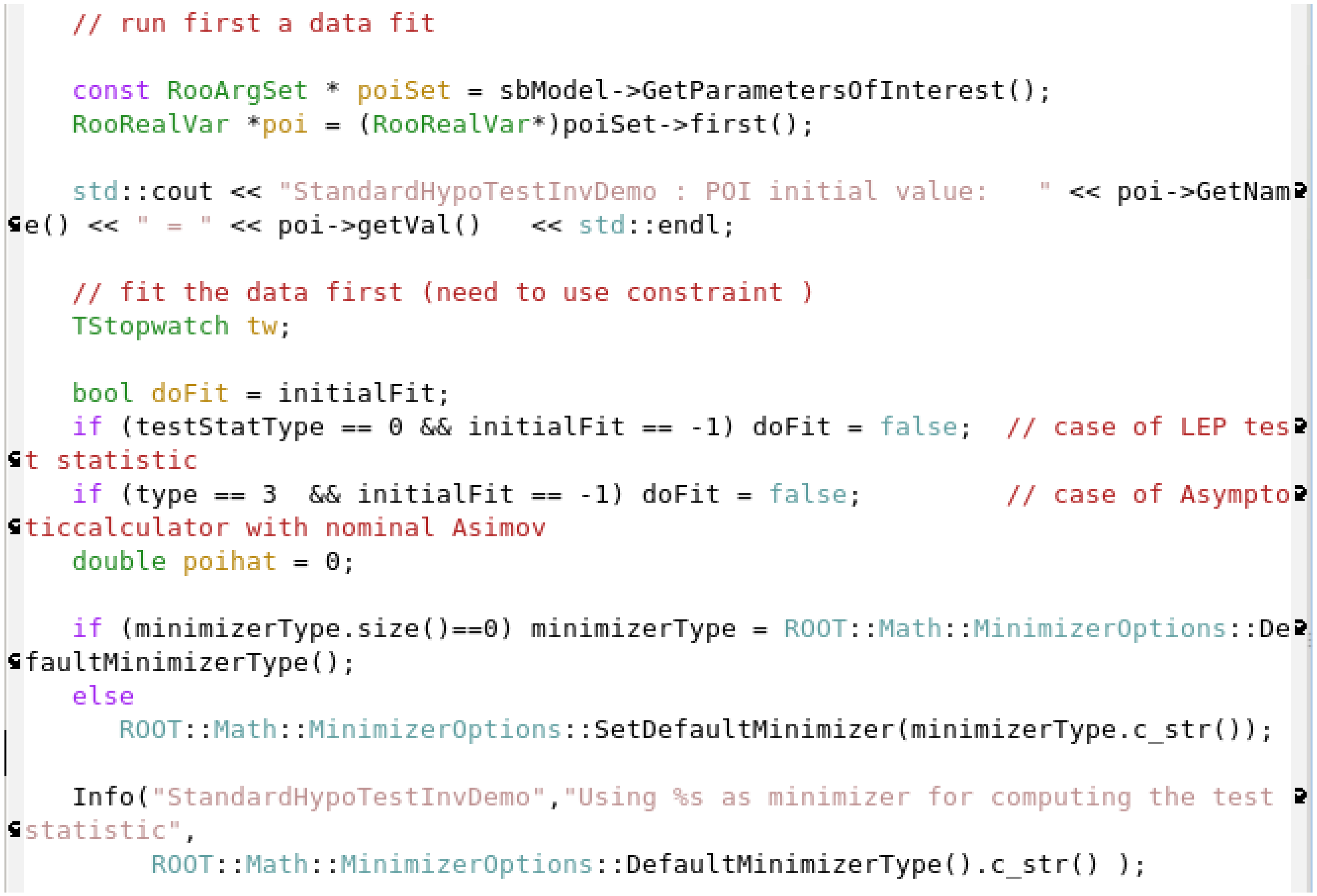
```
   // check model  has global observables when there are nuisance pdf
   // for the hybrid case the globobs are not needed
   if (type != 1 ) {
      bool hasNuisParam = (sbModel->GetNuisanceParameters() && sbModel->GetNuisanceParameters()->getSize() > 0);
      bool hasGlobalObs = (sbModel->GetGlobalObservables() && sbModel->GetGlobalObservables()->getSize() > 0);
      if (hasNuisParam && !hasGlobalObs ) {
         // try to see if model has nuisance parameters first
         RooAbsPdf * constrPdf = RooStats::MakeNuisancePdf(*sbModel,"nuisanceConstraintPdf_sbmodel");
         if (constrPdf) {
            Warning("StandardHypoTestInvDemo",
                                 "Model %s has nuisance parameters but no global observables associated",sbModel->GetName());
            Warning("StandardHypoTestInvDemo","\tThe effect of the nuisance parameters will not be treated correctly ");
         }
      }
   }
```

- Gives you a warning if the s+b model has nuisance parameters, but does not have estimates for the values of these parameters (what this calls “global observables”).



```
// save all initial parameters of the model including the global observables
RooArgSet initialParameters;
RooArgSet * allParams = sbModel->GetPdf()->getParameters(*data);
allParams->snapshot(initialParameters);
delete allParams;
```

- Sure wish I knew what a snapshot was…
  - Surprisingly doesn’t seem very easy to look up.



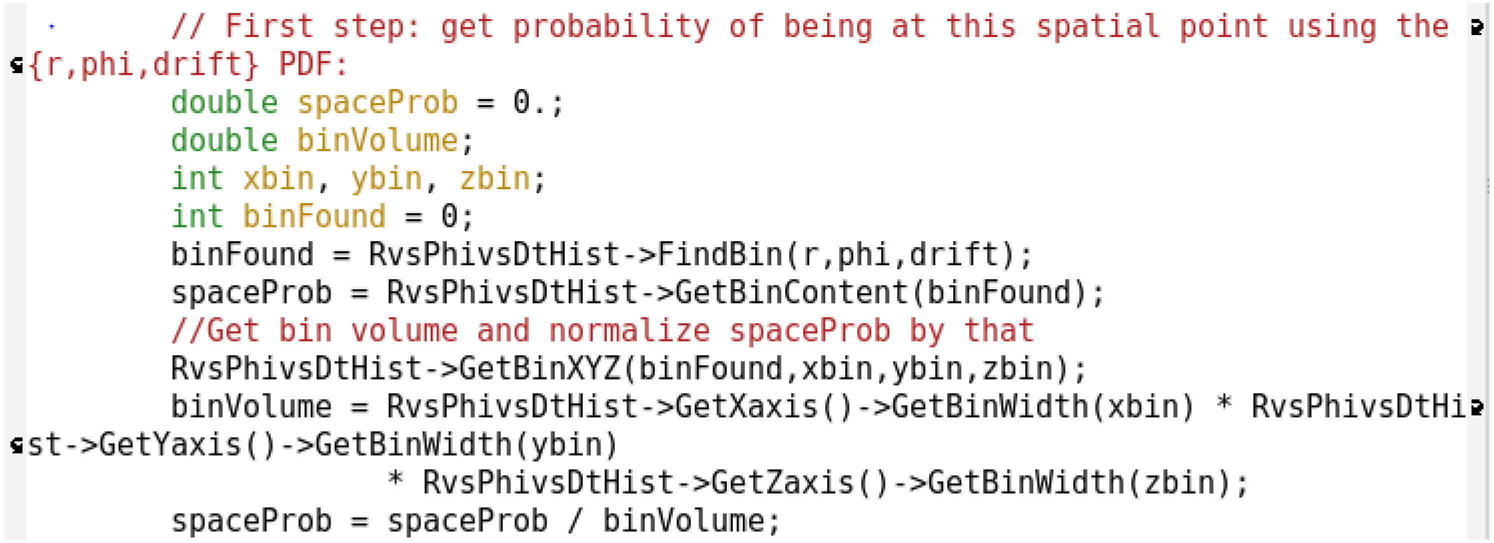
```
    // run first a data fit

    const RooArgSet * poiSet = sbModel->GetParametersOfInterest();
    RooRealVar *poi = (RooRealVar*)poiSet->first();

    std::cout << "StandardHypoTestInvDemo : POI initial value:   " << poi->GetNam
e() << " = " << poi->getVal()   << std::endl;

    // fit the data first (need to use constraint )
    TStopwatch tw;

    bool doFit = initialFit;
    if (testStatType == 0 && initialFit == -1) doFit = false;  // case of LEP tes
t statistic
    if (type == 3  && initialFit == -1) doFit = false;         // case of Asympto
ticcalculator with nominal Asimov
    double poihat = 0;

    if (minimizerType.size()==0) minimizerType = ROOT::Math::MinimizerOptions::De
faultMinimizerType();
    else
       ROOT::Math::MinimizerOptions::SetDefaultMinimizer(minimizerType.c_str());

    Info("StandardHypoTestInvDemo","Using %s as minimizer for computing the test 
statistic",
         ROOT::Math::MinimizerOptions::DefaultMinimizerType().c_str() );
```

- Get parameter of interest from s+b model
- Determine whether you want to do a fit based on your analysis
- Chose the type of minimizer you want to use.

- Leave off here for now, super confusing and hopefully I can get away without understanding it fully.

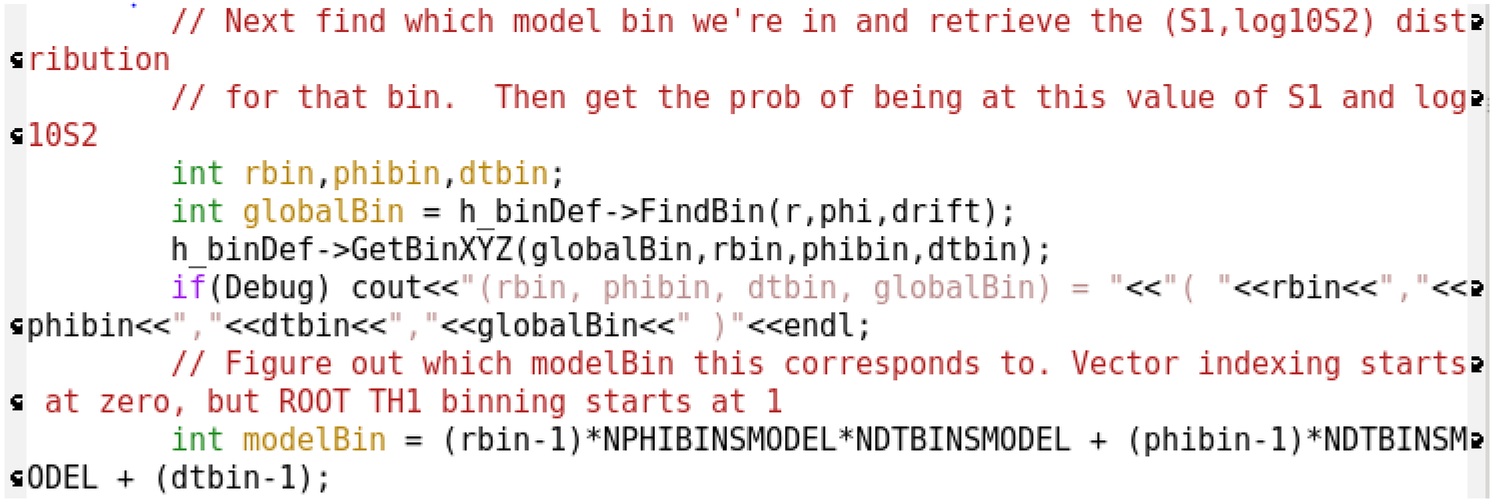
RooBkgPDF.cxx

- RooBkgPDF::evaluate
- So far as I can tell, this is called whenever the value of the PDF needs to be evaluated at a point (go figure). Never explicitly called in the limit code, but invoked implicitly during the hypo test inversion



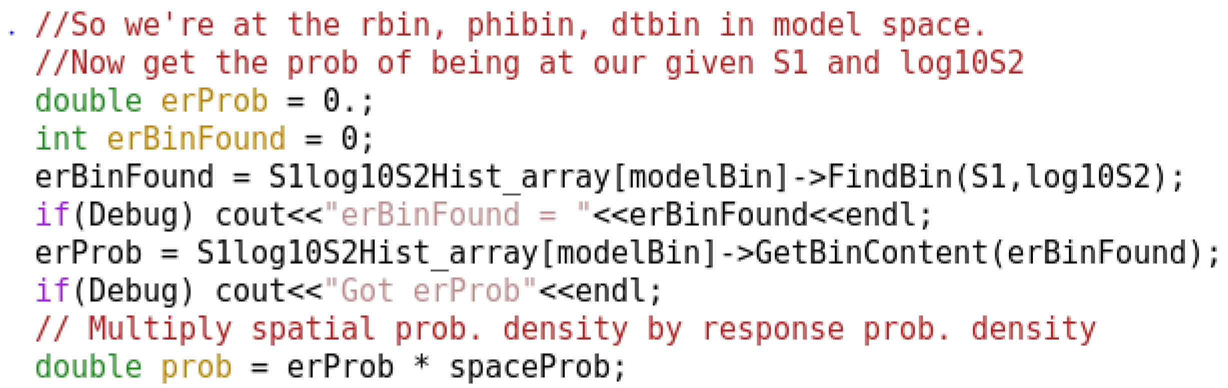
```
        // First step: get probability of being at this spatial point using the 
{r,phi,drift} PDF:
        double spaceProb = 0.;
        double binVolume;
        int xbin, ybin, zbin;
        int binFound = 0;
        binFound = RvsPhivsDtHist->FindBin(r,phi,drift);
        spaceProb = RvsPhivsDtHist->GetBinContent(binFound);
        //Get bin volume and normalize spaceProb by that
        RvsPhivsDtHist->GetBinXYZ(binFound,xbin,ybin,zbin);
        binVolume = RvsPhivsDtHist->GetXaxis()->GetBinWidth(xbin) * RvsPhivsDtHi
st->GetYaxis()->GetBinWidth(ybin)
                    * RvsPhivsDtHist->GetZaxis()->GetBinWidth(zbin);
        spaceProb = spaceProb / binVolume;
```

- Figure out what bin you're in (in the spatial 3D histogram PDF) for a given r, phi, and drift.
  - FindBin returns a global bin number
  - GetBinXYZ fills the bin number in each dimension into the argumetns
  - GetBinContent returns the value of that bin.
- The volume of the bin in question is calculated.
- Spatial probability is calculated as the value of the bin divided by the volume of the bin.



```
        // Next find which model bin we're in and retrieve the (S1,log10S2) dist
ribution
        // for that bin.  Then get the prob of being at this value of S1 and log
10S2
        int rbin,phibin,dtbin;
        int globalBin = h_binDef->FindBin(r,phi,drift);
        h_binDef->GetBinXYZ(globalBin,rbin,phibin,dtbin);
        if(Debug) cout<<"(rbin, phibin, dtbin, globalBin) = "<<"( "<<rbin<<","<<
phibin<<","<<dtbin<<","<<globalBin<<" )"<<endl;
        // Figure out which modelBin this corresponds to. Vector indexing starts
 at zero, but ROOT TH1 binning starts at 1
        int modelBin = (rbin-1)*NPHIBINSMODEL*NDTBINSMODEL + (phibin-1)*NDTBINSM
ODEL + (dtbin-1);
```

- Each "model bin" (combination of x, y, and dt) has its own s1 vs s2 2D histogram.
- Figure out which model bin we are in (a global number just incrimented each dt then phi then r)



```
//So we're at the rbin, phibin, dtbin in model space.
//Now get the prob of being at our given S1 and log10S2
double erProb = 0.;
int erBinFound = 0;
erBinFound = S1log10S2Hist_array[modelBin]->FindBin(S1,log10S2);
if(Debug) cout<<"erBinFound = "<<erBinFound<<endl;
erProb = S1log10S2Hist_array[modelBin]->GetBinContent(erBinFound);
if(Debug) cout<<"Got erProb"<<endl;
// Multiply spatial prob. density by response prob. density
double prob = erProb * spaceProb;
```

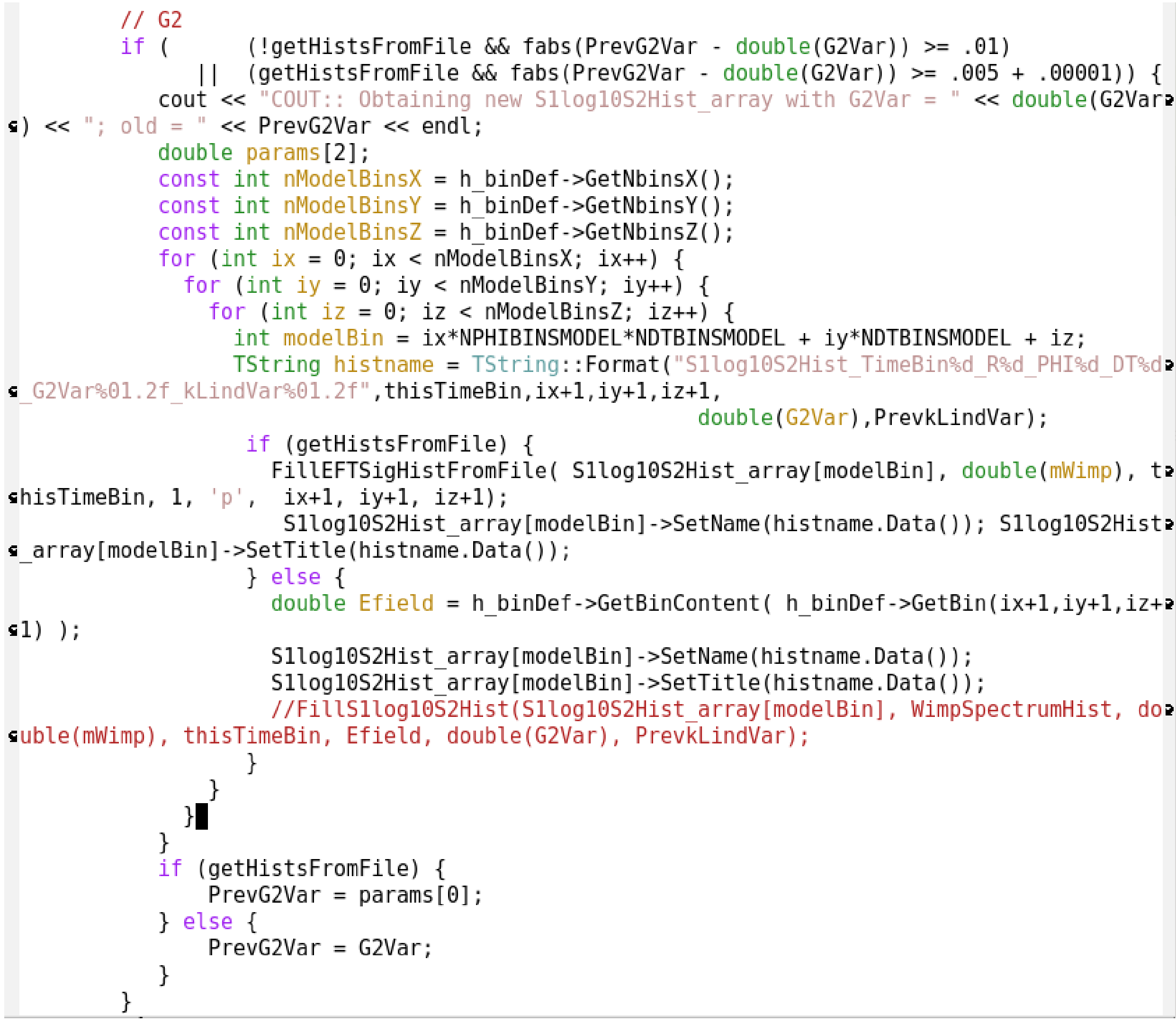
- Determine the s1 vs s2 PDF probability (for the given modelBin).
- Total probability determined by taking the spatial probability times the energy space probability (here “erProb”).

RooSignalPDF.cxx

- RooSignalPDF::evaluate

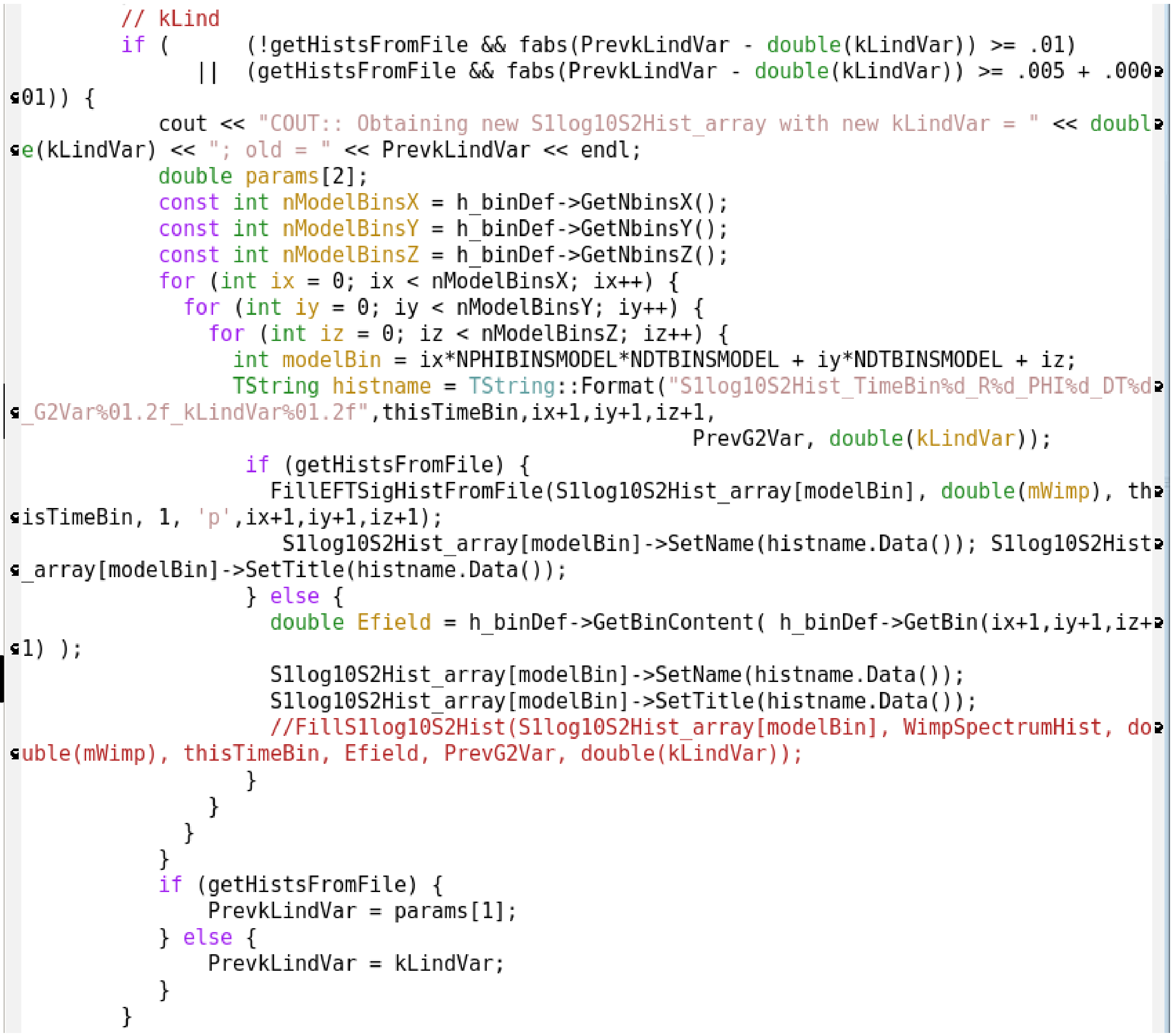
- So far as I can tell, this is called whenever the value of the PDF needs to be evaluated at a point (go figure). Never explicitly called in the limit code, but invoked implicitly during the hypo test inversion

- Re-Fill the signal model hists with the models from the appropriate file if the G2Var has changed from the value used in the currently used file.

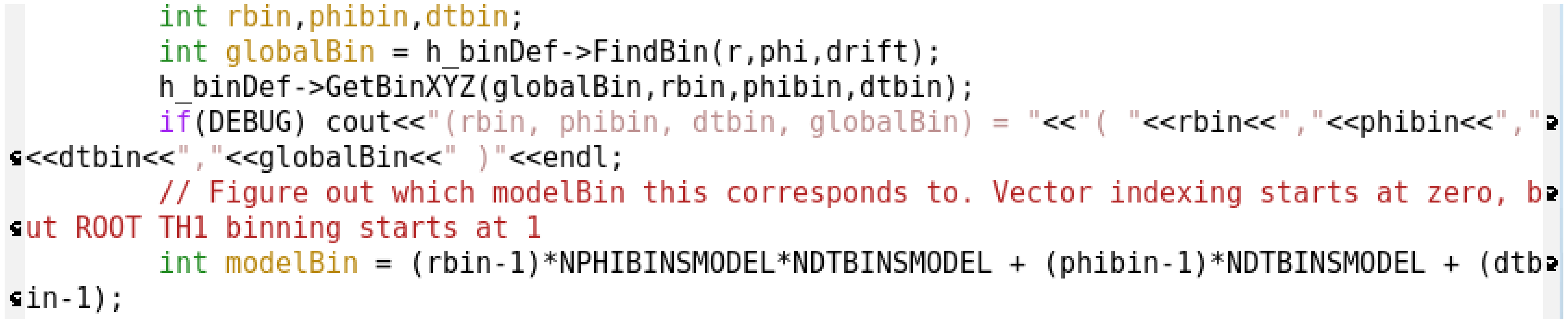


```
        // G2
        if (      (!getHistsFromFile && fabs(PrevG2Var - double(G2Var)) >= .01)
              ||  (getHistsFromFile && fabs(PrevG2Var - double(G2Var)) >= .005 + .00001)) {
           cout << "COUT:: Obtaining new S1log10S2Hist_array with G2Var = " << double(G2Var
) << "; old = " << PrevG2Var << endl;
           double params[2];
           const int nModelBinsX = h_binDef->GetNbinsX();
           const int nModelBinsY = h_binDef->GetNbinsY();
           const int nModelBinsZ = h_binDef->GetNbinsZ();
           for (int ix = 0; ix < nModelBinsX; ix++) {
             for (int iy = 0; iy < nModelBinsY; iy++) {
               for (int iz = 0; iz < nModelBinsZ; iz++) {
                 int modelBin = ix*NPHIBINSMODEL*NDTBINSMODEL + iy*NDTBINSMODEL + iz;
                 TString histname = TString::Format("S1log10S2Hist_TimeBin%d_R%d_PHI%d_DT%d
_G2Var%01.2f_kLindVar%01.2f",thisTimeBin,ix+1,iy+1,iz+1,
                                                      double(G2Var),PrevkLindVar);
                  if (getHistsFromFile) {
                    FillEFTSigHistFromFile( S1log10S2Hist_array[modelBin], double(mWimp), t
hisTimeBin, 1, 'p',  ix+1, iy+1, iz+1);
                     S1log10S2Hist_array[modelBin]->SetName(histname.Data()); S1log10S2Hist
_array[modelBin]->SetTitle(histname.Data());
                  } else {
                    double Efield = h_binDef->GetBinContent( h_binDef->GetBin(ix+1,iy+1,iz+
1) );

                    S1log10S2Hist_array[modelBin]->SetName(histname.Data());
                    S1log10S2Hist_array[modelBin]->SetTitle(histname.Data());
                    //FillS1log10S2Hist(S1log10S2Hist_array[modelBin], WimpSpectrumHist, do
uble(mWimp), thisTimeBin, Efield, double(G2Var), PrevkLindVar);
                  }
               }
             }
           }
           if (getHistsFromFile) {
               PrevG2Var = params[0];
           } else {
               PrevG2Var = G2Var;
           }
        }
```

- Re-Fill the signal model hists with the models from the appropriate file if the kLind has changed from the value used in the currently used file.

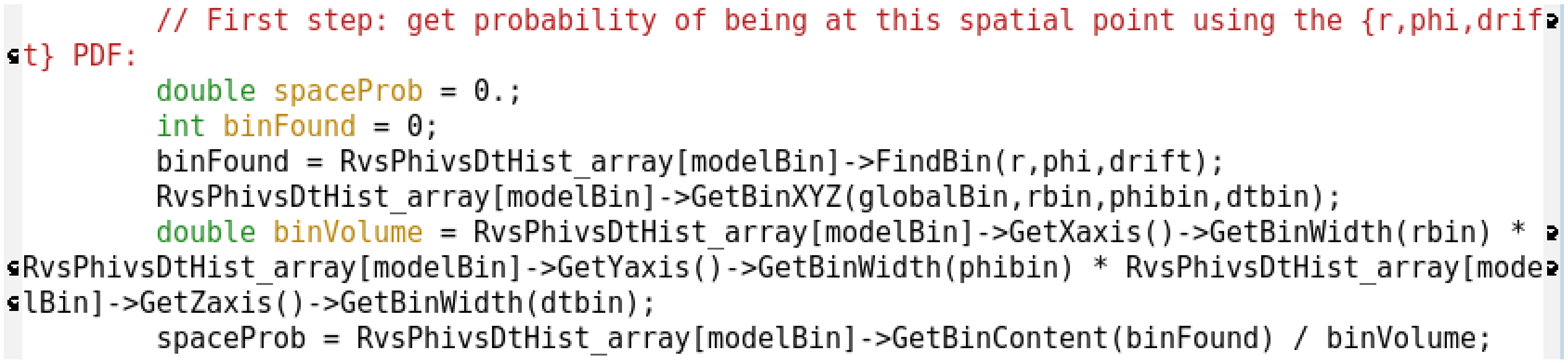


```
        // kLind
        if (      (!getHistsFromFile && fabs(PrevkLindVar - double(kLindVar)) >= .01)
               ||  (getHistsFromFile && fabs(PrevkLindVar - double(kLindVar)) >= .005 + .000
01)) {
           cout << "COUT:: Obtaining new S1log10S2Hist_array with new kLindVar = " << doubl
e(kLindVar) << "; old = " << PrevkLindVar << endl;
           double params[2];
           const int nModelBinsX = h_binDef->GetNbinsX();
           const int nModelBinsY = h_binDef->GetNbinsY();
           const int nModelBinsZ = h_binDef->GetNbinsZ();
           for (int ix = 0; ix < nModelBinsX; ix++) {
             for (int iy = 0; iy < nModelBinsY; iy++) {
               for (int iz = 0; iz < nModelBinsZ; iz++) {
                 int modelBin = ix*NPHIBINSMODEL*NDTBINSMODEL + iy*NDTBINSMODEL + iz;
                 TString histname = TString::Format("S1log10S2Hist_TimeBin%d_R%d_PHI%d_DT%d
_G2Var%01.2f_kLindVar%01.2f",thisTimeBin,ix+1,iy+1,iz+1,
                                                         PrevG2Var, double(kLindVar));
                  if (getHistsFromFile) {
                    FillEFTSigHistFromFile(S1log10S2Hist_array[modelBin], double(mWimp), th
isTimeBin, 1, 'p',ix+1,iy+1,iz+1);
                     S1log10S2Hist_array[modelBin]->SetName(histname.Data()); S1log10S2Hist
_array[modelBin]->SetTitle(histname.Data());
                  } else {
                    double Efield = h_binDef->GetBinContent( h_binDef->GetBin(ix+1,iy+1,iz+
1) );

                    S1log10S2Hist_array[modelBin]->SetName(histname.Data());
                    S1log10S2Hist_array[modelBin]->SetTitle(histname.Data());
                    //FillS1log10S2Hist(S1log10S2Hist_array[modelBin], WimpSpectrumHist, do
uble(mWimp), thisTimeBin, Efield, PrevG2Var, double(kLindVar));
                  }
               }
             }
           }
           if (getHistsFromFile) {
               PrevkLindVar = params[1];
           } else {
               PrevkLindVar = kLindVar;
           }
        }
```



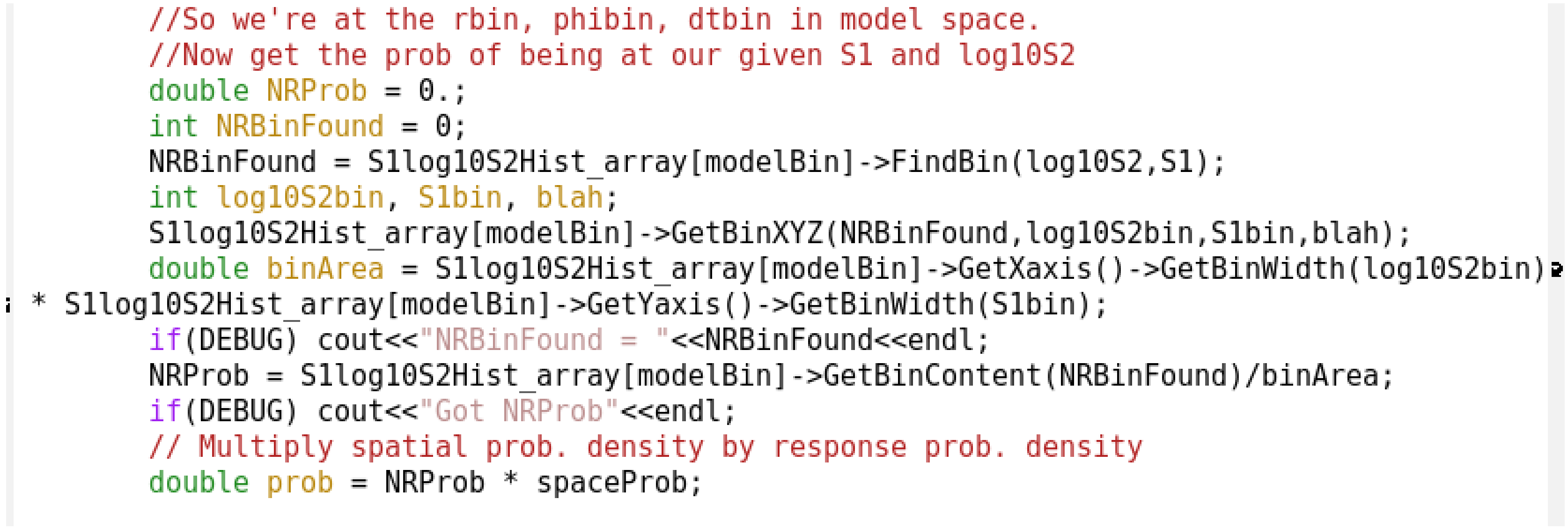
```
        int rbin,phibin,dtbin;
        int globalBin = h_binDef->FindBin(r,phi,drift);
        h_binDef->GetBinXYZ(globalBin,rbin,phibin,dtbin);
        if(DEBUG) cout<<"(rbin, phibin, dtbin, globalBin) = "<<"( "<<rbin<<","<<phibin<<","
<<dtbin<<","<<globalBin<<" )"<<endl;
        // Figure out which modelBin this corresponds to. Vector indexing starts at zero, b
ut ROOT TH1 binning starts at 1
        int modelBin = (rbin-1)*NPHIBINSMODEL*NDTBINSMODEL + (phibin-1)*NDTBINSMODEL + (dtb
in-1);
```

- Each "model bin" (combination of x, y, and dt) has its own s1 vs s2 2D histogram.
- Figure out which model bin we are in (a global number just incremented each dt then phi then r)



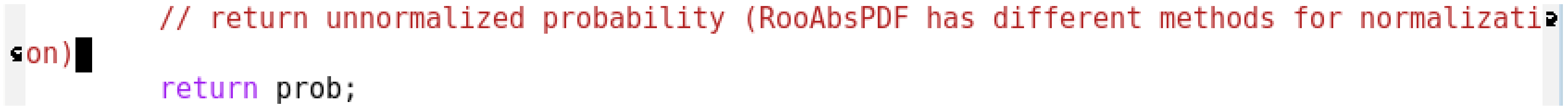
```
        // First step: get probability of being at this spatial point using the {r,phi,drif
t} PDF:
        double spaceProb = 0.;
        int binFound = 0;
        binFound = RvsPhivsDtHist_array[modelBin]->FindBin(r,phi,drift);
        RvsPhivsDtHist_array[modelBin]->GetBinXYZ(globalBin,rbin,phibin,dtbin);
        double binVolume = RvsPhivsDtHist_array[modelBin]->GetXaxis()->GetBinWidth(rbin) * 
RvsPhivsDtHist_array[modelBin]->GetYaxis()->GetBinWidth(phibin) * RvsPhivsDtHist_array[mode
lBin]->GetZaxis()->GetBinWidth(dtbin);
        spaceProb = RvsPhivsDtHist_array[modelBin]->GetBinContent(binFound) / binVolume;
```

- Figure out what bin you’re in (in the spatial 3D histogram PDF) for a given r, phi, and drift.
  - FindBin returns a global bin number
  - GetBinXYZ fills the bin number in each dimension into the argumetns
  - GetBinContent returns the value of that bin.
- The volume of the bin in question is calculated.
- Spatial probability is calculated as the value of the bin divided by the volume of the bin.



```
        //So we're at the rbin, phibin, dtbin in model space.
        //Now get the prob of being at our given S1 and log10S2
        double NRProb = 0.;
        int NRBinFound = 0;
        NRBinFound = S1log10S2Hist_array[modelBin]->FindBin(log10S2,S1);
        int log10S2bin, S1bin, blah;
        S1log10S2Hist_array[modelBin]->GetBinXYZ(NRBinFound,log10S2bin,S1bin,blah);
        double binArea = S1log10S2Hist_array[modelBin]->GetXaxis()->GetBinWidth(log10S2bin)
 * S1log10S2Hist_array[modelBin]->GetYaxis()->GetBinWidth(S1bin);
        if(DEBUG) cout<<"NRBinFound = "<<NRBinFound<<endl;
        NRProb = S1log10S2Hist_array[modelBin]->GetBinContent(NRBinFound)/binArea;
        if(DEBUG) cout<<"Got NRProb"<<endl;
        // Multiply spatial prob. density by response prob. density
        double prob = NRProb * spaceProb;
```

- Determine the s1 vs s2 PDF probability (for the given modelBin).
- Total probability determined by taking the spatial probability times the energy space probability (here “NRProb”).



```
        // return unnormalized probability (RooAbsPDF has different methods for normalizati
on)
        return prob;
```

- Return the evaluated probability.