```
#include "src/GeomParameters.hpp"
#include "TGeoManager.h"
#include "TGLViewer.h"
#include "TGLCamera.h"
#include "TGLPerspectiveCamera.h"


Bool_t Build_Geom(const TGeoManager* geoManager);
Bool_t BuildFields(const TGeoManager* geoManager);

using namespace GeomParameters;

//_____________________________________________________________________________
Int_t sourcetube_geom()
{
  // Load project's shared library (see $ROOTALIAS for function definition)
  gSystem->Load("libUCN.so");
  // Create the geoManager
  TGeoManager* geoManager = new TGeoManager("GeoManager","Geometry Manager");
  // Build and write to file the simulation and visualisation geoms
  Build_Geom(geoManager);
  // Draw Geom
  TCanvas* geomCanvas = new TCanvas("GeomCanvas","Canvas for visualisation of EDM
Geom",60,40,200,200);
  geomCanvas->cd();
  double camera[3] = {0., 0., 0.};
  Analysis::Geometry::DrawGeometry(*geomCanvas, *geoManager, camera);
        /// Build Fields
        /// Comment this out if you do not want Magnetic, eklectric or gravitational fields to be
created.
  BuildFields(geoManager);
  return 0;
}

//_____________________________________________________________________________
Bool_t Build_Geom(const TGeoManager* geoManager)
{
  // --------------------------------------
  // BUILDING GEOMETRY
  // Materials - Define the materials used. Leave the neutron properties to be defined on a run-by-
run basis

  Materials::BuildMaterials(geoManager);
  TGeoMedium* beryllium = geoManager->GetMedium("Beryllium");
  TGeoMedium* blackhole = geoManager->GetMedium("BlackHole");
  TGeoMedium* heliumII = geoManager->GetMedium("HeliumII");

  // --------------------------------------
  // -- Making Top Volume
  Box* topShape = new Box("Top",100,100,100);
  BlackHole* top = new BlackHole("Top", topShape, blackhole);
  geoManager->SetTopVolume(top);
```

```
  top->SetVisibility(kFALSE);

  // -- Make the boundary volume in which all the others sit
  // -- This is what we will be reflecting off all the time
  Double_t surfaceRoughness = 0.1;
  Box* chamberShape = new Box("Chamber",10,10,10);
  Boundary* chamber = new Boundary("Chamber", chamberShape, beryllium, surfaceRoughness);
  chamber->SetLineColor(kOrange-7);
  chamber->SetLineWidth(1);
  chamber->SetVisibility(kFALSE);
  chamber->SetTransparency(80);
  // Add to geom
  top->AddNode(chamber,1);

  // ------------------------------------
  // -- SOURCE TUBE
  // -- Source tube has 13 segments, all of which are identical (except one which has a hole in the
top)

  // -- Make a SourceTube Segment
  Tube *sourceSegShape = new Tube("SourceSeg", sourceSegRMin, sourceSegRMax,
sourceSegHalfLength);
  TrackingVolume* sourceSeg = new TrackingVolume("SourceSeg", sourceSegShape, heliumII);
  sourceSeg->SetLineColor(kAzure-4);
  sourceSeg->SetLineWidth(1);
  sourceSeg->SetVisibility(kTRUE);
  sourceSeg->SetTransparency(20);

  Double_t segmentYPosition = sourceSegYPos;
  for (Int_t segNum = 1; segNum <= 13; segNum++) {
    // -- Define SourceTube matrix
    // Rotation seems to be applied before translation so bear that in mind when choosing which
    // coordinate to transform in translation
    TGeoRotation segmentRot("SegmentRot",sourceSegPhi,sourceSegTheta,sourceSegPsi); // phi,
theta, psi
    TGeoTranslation
segmentTra("SegmentTra",sourceSegXPos,segmentYPosition,sourceSegZPos); // x, y, z
    TGeoCombiTrans segmentCom(segmentTra,segmentRot);
    TGeoHMatrix segmentMat = segmentCom;
    Char_t sourceMatrixName[20];
    sprintf(sourceMatrixName, "SourceMatrix%d", segNum);
    segmentMat.SetName(sourceMatrixName);
    chamber->AddNode(sourceSeg, segNum, new TGeoHMatrix(segmentMat));
    segmentYPosition += 2.*sourceSegHalfLength; // Shift next segment along by length of
segment
  }

  // ------------------------------------
  // -- SOURCE VALVE
  // Valve entrance volume is a shorter source-tube segment-like that connects
  // the valve volume to the source
  Tube *valveVolEntranceShape = new Tube("ValveVolEntrance", valveVolEntranceRMin,
```

```
valveVolEntranceRMax, valveVolEntranceHalfLength);
  TrackingVolume* valveVolEntrance = new TrackingVolume("ValveVolEntrance",
valveVolEntranceShape, heliumII);
  valveVolEntrance->SetLineColor(kTeal-3);
  valveVolEntrance->SetLineWidth(1);
  valveVolEntrance->SetVisibility(kTRUE);
  valveVolEntrance->SetTransparency(20);
  // -- Define the Valve volume entrance
  TGeoRotation
valveVolEntranceRot("ValveVolEntranceRot",valveVolEntrancePhi,valveVolEntranceTheta,valveVo
lEntrancePsi);
  TGeoTranslation
valveVolEntranceTra("ValveVolEntranceTra",valveVolEntranceXPos,valveVolEntranceYPos,valve
VolEntranceZPos);
  TGeoCombiTrans valveVolEntranceCom(valveVolEntranceTra,valveVolEntranceRot);
  TGeoHMatrix valveVolEntranceMat = valveVolEntranceCom;
  chamber->AddNode(valveVolEntrance, 1, new TGeoHMatrix(valveVolEntranceMat));

  // ---------------------------------
  // -- Close Geometry
  geoManager->CloseGeometry();

  // ---------------------------------
  // -- Write out geometry to file
  const char *fileName = "sourcetube_geom.root";
  cerr << "Simulation Geometry Built... Writing to file: " << fileName << endl;
  geoManager->Export(fileName);

  return kTRUE;
}
/*
///_____________________________________________________________________________
/// Use this method to include GRAVITY and a MAGNETIC FIELD from an input map.
/// The Field map should have 6 columns of data of the form
/// x(m) y(m) z(m) Bx(T as 0.000000n) By(T as 0.000000n) Bz(T as 0.000000n)
///_____________________________________________________________________________
Bool_t BuildFields(const TGeoManager* geoManager)
{
  cout << "--------------------------------" << endl;
  cout << "Building Fields" << endl;
  cout << "--------------------------------" << endl;
  // --------------------------------
  // -- Gravitational Field
  GravField* gravField = new GravField(-1.0,0.0,0.0);
  printf("Gravitational Field - nx: %.4f \t ny: %.4f \t nz: %.4f\n",gravField->Nx(),gravField-
>Ny(),gravField->Nz());

  // --------------------------------
  // -- Create a Magnetic field and write it to file

        /// Input field map of name magfilename
  string magfilename = "runs/SourceTubeStudies/06_08_10_MattsT2_Map.txt";
```

```
  /// Define shape and size of field
  TGeoShape* magFieldShape = new Box("FieldShape",10, 10, 10);
  /// Define transformation that locates field in local geometry (in this case TGeoMatrix hvcellmat
is used)
  TGeoMatrix* fieldMatrix = static_cast<TGeoMatrix*>(geoManager->GetListOfMatrices()-
>FindObject("hvcellmat"));
        cout << "here" << endl;
  TGeoMatrix* magFieldPosition = new TGeoHMatrix(*fieldMatrix);
        cout << "and here" << endl;
  MagFieldMap* field = new MagFieldMap("Field", magFieldShape, magFieldPosition);
        cout << "and here 2" << endl;
  if (field->BuildMap(magfilename) == kFALSE) {
    Error("BuildFieldMap","Cannot open file: %s", magfilename);
    return kFALSE;
  }
  // Add field to magfield manager
  MagFieldArray* magFieldArray = new MagFieldArray();
  magFieldArray->AddField(field);


  // --------------------------------
  // -- Write magfieldmanager to geometry file
  const char *outputFileName = "runs/SourceTubeStudies/sourcetube_fieldsIncMag.root";
  TFile *file = Analysis::DataFile::OpenRootFile(outputFileName, "recreate");
  gravField->Write(gravField->GetName());
  magFieldArray->Write(magFieldArray->GetName());
  file->Close();
  delete gravField;
  delete magFieldArray;
  return kTRUE;
}
*/

///_____________________________________________________________________________
/// Use this method to include only GRAVITY.
///_____________________________________________________________________________
Bool_t BuildFields(const TGeoManager* geoManager)
{
  cout << "--------------------------------" << endl;
  cout << "Building Fields" << endl;
  cout << "--------------------------------" << endl;
  // --------------------------------
  // -- Gravitational Field
  GravField* gravField = new GravField(-1.0,0.0,0.0); // Pointing down the X-Axis for this geom
  printf("Gravitational Field - nx: %.4f \t ny: %.4f \t nz: %.4f\n",gravField->Nx(),gravField-
>Ny(),gravField->Nz());
  // -- Write fields to file
  const char *fieldsFileName = "sourcetube_fields.root";
  TFile *file = Analysis::DataFile::OpenRootFile(fieldsFileName, "recreate");
  cout << "Fields Created... Writing to file: " << fieldsFileName << endl;
  gravField->Write(gravField->GetName());
  file->Close();
```

```
  // Cleanup
  delete gravField;
  return kTRUE;
}

/*
//---------------Build gravitational and Magnetic (from file) Fields ------------------------------
//_____________________________________________________________________________
Bool_t BuildFields(const TGeoManager* geoManager)
{
  cout << "--------------------------------" << endl;
  cout << "Building Fields" << endl;
  cout << "--------------------------------" << endl;
  // ---------------------------------
  // -- Gravitational Field
  GravField* gravField = new GravField(-1.0,0.0,0.0);
  printf("Gravitational Field - nx: %.4f \t ny: %.4f \t nz: %.4f\n",gravField->Nx(),gravField-
>Ny(),gravField->Nz());

  // ---------------------------------
  // -- Create a Magnetic field and write it to file
  string magfilename = "runs/SourceTubeStudies/25_Apr_2012_Phil_Map.txt";
  // Define shape of field
  TGeoShape* magFieldShape = new Box("FieldShape", 100, 100, 100);
  // Define transformation that locates field in geometry
  TGeoMatrix* fieldMatrix = static_cast<TGeoMatrix*>(geoManager->GetListOfMatrices()-
>FindObject("segmentMat"));
  TGeoMatrix* magFieldPosition = new TGeoHMatrix(*fieldMatrix);
  MagFieldMap* field = new MagFieldMap("Field", magFieldShape, magFieldPosition);
        cout << "and here" << endl;
  if (field->BuildMap(magfilename) == kFALSE) {
    Error("BuildFieldMap","Cannot open file: %s", magfilename);
    return kFALSE;
  }
  // Add field to magfield manager
  MagFieldArray* magFieldArray = new MagFieldArray();
        cout << "here too" << endl;
  magFieldArray->AddField(field);

  // ---------------------------------
  // -- Write magfieldmanager to geometry file
  const char *outputFileName = "runs/SourceTubeStudies/sourcetube_fieldsIncMag.root";
  TFile *file = Analysis::DataFile::OpenRootFile(outputFileName, "recreate");
  gravField->Write(gravField->GetName());
  magFieldArray->Write(magFieldArray->GetName());
  file->Close();
  delete gravField;
  delete magFieldArray;
  return kTRUE;
}*/
```