ROOT output

Write information on an external file

- For a long time, Geant4 did'nt provide any native data analysis tool. As a general rule, the user was supposed provide his own code to output results to an appropriate analysis format and to use an external analysis tool.
- In the latest Geant4(9.5) releases, a few basics classes for data analysis have been implemented:
  - Support for histograms and ntuples
  - Output in ROOT, CSV (ASCII)
- Usually, people define in RunAction Histograms and Tuples using G4AnalysisManager and Fill them in EventAction after analysing events
- In our case we will :
  - Write our own ROOTManager class to create, write and close root file.
  - Fill hit information for each event (EventAction)
  - Calling ROOtManager in our RunAction to Write(), Save() our root file

- **Download** ROOTManager.cc && ROOTManager.hh from the indico page : put **ROOTManager.cc** in the **/src directory** and **ROOTManager.hh** in the **/include directory**

ROOTManager .cc

```cpp
#include "ROOTManager.hh"
#include <TROOT.h>
#include <TFile.h>
#include <TBranch.h>
#include <TTree.h>
#include <CLHEP/Units/SystemOfUnits.h>
#include "G4UnitsTable.hh"
#include"Messenger.hh"
ROOTManager* ROOTManager::fgInstance = 0;

ROOTManager* ROOTManager::Instance()
{
  return fgInstance;
}

ROOTManager::ROOTManager()
{
  fgInstance = this;
}

ROOTManager::~ROOTManager()
{
  if ( ROOTFile ) delete ROOTFile;
  fgInstance = 0;
}
```

Initialization of instance, constructor and destroctor of ROOTManager

```cpp
void ROOTManager::Init()
{
 // Creating a tree container to handle histograms and ntuples.
 // This tree is associated to an output file.
 //
  // create ROOT file
  G4String input = "Detector.root";
  ROOTFile = new TFile(input, "RECREATE");
  if (!ROOTFile) {
    G4cout << " problem creating the ROOT TFile" << G4endl;
    return;
  }

  ROOTTree = new TTree("Default", "Default");
//Generator
  ROOTTree->Branch("Event", &ROOTTreeStruct.Event, "Event/I");
  ROOTTree->Branch("NGenPart", &ROOTTreeStruct.NGenPart, "NGenPart/I");
  ROOTTree->Branch("GenPartID", &ROOTTreeStruct.GenPartID, "GenPartID[NGenPart]/I");
  ROOTTree->Branch("GenPartPDG", &ROOTTreeStruct.GenPartPDG, "GenPartPDG[NGenPart]/F");
  ROOTTree->Branch("GenPartE", &ROOTTreeStruct.GenPartE, "GenPartE[NGenPart]/F");
  ROOTTree->Branch("GenPartTheta", &ROOTTreeStruct.GenPartTheta, "GenPartTheta[NGenPart]/F");
  ROOTTree->Branch("GenPartPhi", &ROOTTreeStruct.GenPartPhi, "GenPartPhi[NGenPart]/F");
  ROOTTree->Branch("NPannelHit", &ROOTTreeStruct.NPannelHit, "NPannelHit/I");

 //Particles
  ROOTTree->Branch("HitDepE", &ROOTTreeStruct.PannelHitE, "HitDepE[NPannelHit]/F");
  ROOTTree->Branch("HitPosX", &ROOTTreeStruct.PannelHitPosX, "HitPosX[NPannelHit]/F");
  ROOTTree->Branch("HitPosY", &ROOTTreeStruct.PannelHitPosY, "HitPosY[NPannelHit]/F");
  ROOTTree->Branch("HitPosZ", &ROOTTreeStruct.PannelHitPosZ, "HitPosZ[NPannelHit]/F");
  ROOTTree->Branch("HitPDG", &ROOTTreeStruct.HitPDG, "HitPDG[NPannelHit]/F");
  ROOTTree->Branch("HitPannelID", &ROOTTreeStruct.PannelHitID, "HitPannelID[NPannelHit]/I");

}
```

Create ROOTFile, Tree and different branches in the Tree for storing the data(name of the branch in your tree, adrees inside the ROOTTreestructure and the format of the variable(integer , double,double_t ,Float_t...))

```cpp
void ROOTManager::Save()
{
  if (ROOTFile) {
    ROOTFile->Write();
    ROOTFile->Close();
    G4cout << "ROOT Tree closed" << G4endl;
  }
}

void ROOTManager::Fill()
{
  G4cout << "Fill" << G4endl;
  ROOTTree->Fill();

}
```

Write, Fill and Close your root file

ROOTManager.hh

```
#include "globals.hh"
#include "EventAction.hh"

#include <TFile.h>
#include <TTree.h>

class ROOTManager
{

  static const int MaxNGenPart = 500;
  static const int MaxNPannelHit = 500;

  struct ROOTTreeStruct_t {
//Generator
    Int_t Event;
    Int_t NGenPart;
    Int_t GenPartID[MaxNGenPart];
    Int_t GenPartPDG[MaxNGenPart];
    Float_t GenPartE[MaxNGenPart];
    Float_t GenPartTheta[MaxNGenPart];
    Float_t GenPartPhi[MaxNGenPart];

//G4Particle
    Int_t NPannelHit;
    Float_t PannelHitPosX[MaxNPannelHit];
    Float_t PannelHitPosY[MaxNPannelHit];
    Float_t PannelHitPosZ[MaxNPannelHit];
    Int_t PannelHitID[MaxNPannelHit];
    Float_t HitPDG[MaxNPannelHit];

};

public:

  ROOTManager();
  ~ROOTManager();
  static ROOTManager* Instance();
  void Init();
  void Save();
  void Fill();
  struct ROOTTreeStruct_t ROOTTreeStruct;
private:

  static ROOTManager* fgInstance;
  TFile* ROOTFile;
  TTree* ROOTTree;
```

Define the maximum number of generated particles and pannelhits=>Maximum size of array

Define the layout of the data to be stored in the root file

Arrays to hold informations related to particles generated and pannel hits

Constructor and destructor of the class

Static member that returns a pointer to the ROOTManager: ensuring that one instance of the class can exist

Methods by order: Initialization, saving and filling of the data into the ROOT file

Instance of the structure which holds the data that will be written to the ROOT file

Instance for implementing the Singleton pattern

Pointer to the root file and the tree object where data will be stored

Exercice :

- Open ROOTManager.hh and go inside the ROOTManager class :
- Specify the maximum number of Hit and particle by event to 500

ROOTManager.hh

```
static const int MaxNGenPart = 500;
static const int MaxNPannelHit = 500;
```

- Define a structure that contain different variables :

```
struct ROOTTreeStruct_t {
}
```

In your strucrure :

- Define the variables related to the generator : number of events, arrays of E,theta,phi ....

```
struct ROOTTreeStruct_t {
  Int_t Event; //# of events
  Int_t NGenPart; //# of particles by events
  Int_t GenPartID[MaxNGenPart]; //# ID of particle
  Float_t GenPartPDG[MaxNGenPart]; //type of particle
  Float_t GenPartE[MaxNGenPart]; //kinetic energy
  Float_t GenPartTheta[MaxNGenPart]; //zenith angle
  Float_t GenPartPhi[MaxNGenPart]; //azimutal angle
//Add other variables related to hits
};
```

- In the same structure define the informations related to the hits (interactions): NPannelHit, PannelID[MaxNPannelHit], PannelHitPosX[MaxNPannelHit], PannelHitPosY[MaxNPannelHit] PannelHitPosZ[MaxNPannelHit], HitPDG[MaxNPannelHit]

Exercice :

EventAction.cc

- Open EventAction.cc

```
Include "ROOTManager.hh"
```

- Go to EndOfEventAction() :
- Create an instance to ROOTManager

```
auto myrootmanager = ROOTManager::Instance();
```

- Initialize NGenPart && NPannelHit =0

```
myrootmanager->ROOTTreeStruct.NGenPart = 0;
myrootmanager->ROOTTreeStruct.NPannelHit = 0;
```

- Fill event branch:

```
myrootmanager->ROOTTreeStruct.Event = event->GetEventID();
```

- Go to the loop over the hits:
  - Fill your branches with respect to the #of hits :

```
myrootmanager->ROOTTreeStruct.PannelHitPosX[myrootmanager->ROOTTreeStruct.NPannelHit] = (Float_t)pos.x();
```

  - Fill other branches: PannelHitPosY,PannelHitPosZ,HitPDG, PannelHitID
  - Count the number of hit by events:

```
myrootmanager->ROOTTreeStruct.NPannelHit++;
```

  - Print hit :

```
hit->Print();
```

Exercice :

- Same for the generator

EventAction.cc

- Go to the loop over the generated particles:
  - Fill : TrackID , GenPartE, GenPartPDG, GenPartTheta, GenPartPhi

```
{
  myrootmanager->ROOTTreeStruct.GenPartID[myrootmanager->ROOTTreeStruct.NGenPart] = (Int_t)primary->GetTrackID();
//Count the number of hit by events:
      myrootmanager->ROOTTreeStruct.NGenPart++;
}
```

use :

```
(Float_t)primary->GetG4code()->GetPDGEncoding();
(Float_t)primary->GetKineticEnergy();
(Float_t)primary->GetMomentum().theta();
(Float_t)primary->GetMomentum().phi();
```

To fill GenPartPDG, GenPartE, GenPartTheta and GenPartPhi

- Before clossing the bracket of EndOfEventAction() :

```
myrootmanager->Fill();
```

Exercice :

ROOTManager.cc

- Open ROOTManager.cc:
- Go to Init()
  - Create your ROOTFile:

```
G4String input = "Detector.root";
ROOTFile = new TFile(input, "RECREATE");
```

  - Create Tree:

```
ROOTTree = new TTree("Default", "Default");
```

  - Create Branches:

```
//Generator branches
ROOTTree->Branch("Event", &ROOTTreeStruct.Event, "Event/I");
ROOTTree->Branch("NGenPart", &ROOTTreeStruct.NGenPart, "NGenPart/I");
ROOTTree->Branch("GenPartID", &ROOTTreeStruct.GenPartID, "GenPartID[NGenPart]/I");
//Hit branches
ROOTTree->Branch("NPannelHit", &ROOTTreeStruct.NPannelHit, "NPannelHit/I");
ROOTTree->Branch("PannelHitID", &ROOTTreeStruct.PannelHitID, "PannelHitID[NPannelHit]/I");
```

- Add other branches: GenPartE, GenPartTheta;GenPartPhi, PannelHitPosX,PannelHitPosY,PannelHitPosZ,HitPDG

Final touch

- Go to **RunAction.cc** :

RunAction.cc

#include ROOTManager.hh

- **Initialize** your **rootmanager** at the **beginning** of the **run** and **save** it at the **end**

```
void RunAction::BeginOfRunAction(const G4Run* aRun)
{

  ROOTManager::Instance()->Init();
}
```

```
void RunAction::EndOfRunAction(const G4Run*)
{

  ROOTManager::Instance()->Save();
}
```

- Go to **exampleB1.cc** :

exampleB1.cc

#include ROOTManager.hh

- **Call** your **ROOTManager** in your main

```
//Root

  auto theROOTmanager = new ROOTManager();
```

- cd build
- Remove everything : rm -rf *
- micromamba activate geant-root
- cmake ../
- make
- ./example cmd.file
- After that you should see a root output(Detector.root) contain all the variables we need :
- root –l Detector.root
- new TBrowser