Muography Workshop

End-to-end simulation framework

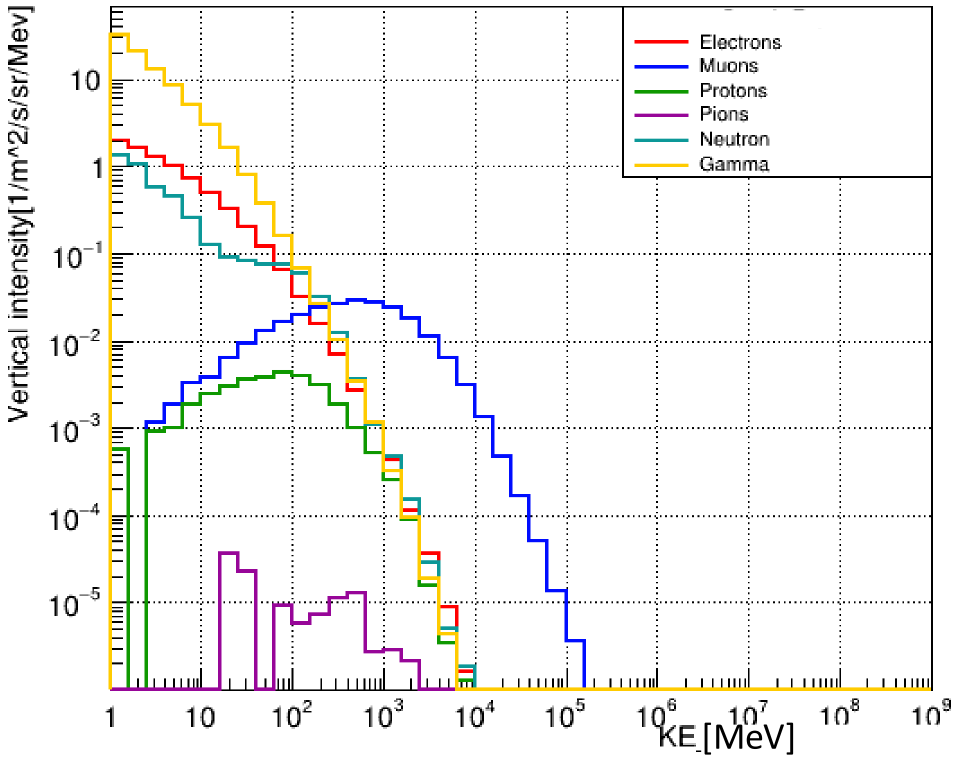
Generator

CRY: **C**osmic **R**a**Y** Shower Library

- A Monte Carlo parametric simulation:
- The flux and the kinematics of all **secondaries** (μ, n, p, e, γ, π, K) tabulated from MCNPX*, assuming showers from protons (1 GeV-100 TeV)
- Take into acccount geomagnetic effects on the cosmic flux depending on the **time** (solar cycle), **latitude** and **altitude** (provide 3 options : 0, 2100, 11300 m)
- Limited to flat surface (with surface = **subboxLength**^{2} [m^2])

```
returnNeutrons 1
returnProtons 1
returnGammas 1
returnKaons 1
returnPions 1
returnElectrons 1
returnMuons 1
date 7-1-2012 # month-day-year(solar activity effect)
latitude 48.0  # depend from the region(magnetic field effect)
altitude 0     # 0,2100,11300 m
subboxLength 0.16  # this quantity is chose with respect to your detector(maximum value = 300m)
```

Input setup file

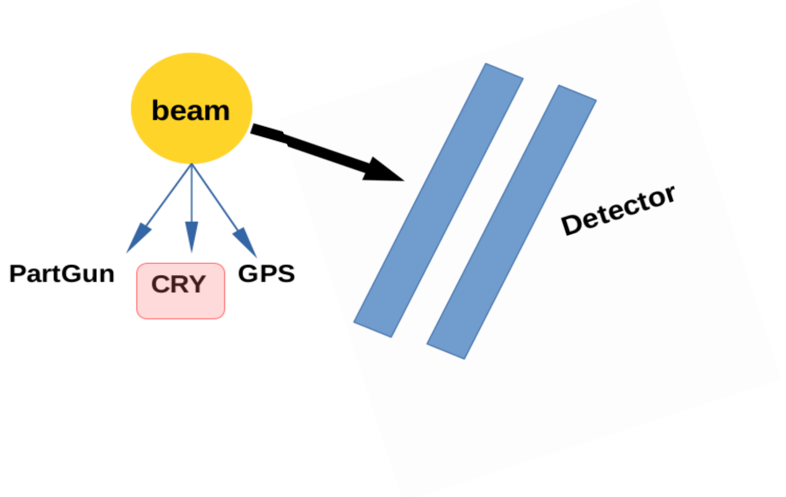


Secondaries flux at sea level

(*)Monte Carlo N-Particle eXtended: is a widely used computer code for simulating the transport of particles, such as neutrons, photons, electrons, and other charged particles, through various materials and complex geometries.

Generator : PrimaryGeneratorAction

- PartGun:
  - G4ParticleGun is a generator provided by Geant4.
  - This class generates primary particle(s) with a given momentum and position.
- GPS:
  - The G4GeneralParticleSource (GPS) is part of the Geant4 toolkit for Monte-Carlo, high-energy particle transport, GPS allows the user to control the following characteristics of primary particles:
    - Spatial sampling (2D or 3D surfaces such as discs, spheres, and boxes)
    - Angular distribution (unidirectional, isotropic, cosine-law, beam or arbitrary..)
    - Spectrum (linear, exponential, power-law, Gaussian,..)
- CRY :
  - Real flux generator (we need to link it to our Geant4 code):
    - Shape: flat surface
    - Energy: real spectrum of energy
    - Momentum direction: zenith angle [0°,90°] and flat azimuthal angle [0°,180°]
    - Created geometry should be all the time in the negative Z direction
- Other cosmic rays generator: EcoMug and CORSIKA



Generator : cmd.file

```
/run/initialize
/CRY/input returnNeutrons 0
/CRY/input returnProtons 0
/CRY/input returnGammas 0
/CRY/input returnPions 0
/CRY/input returnKaons 0
/CRY/input returnElectrons 0
/CRY/input returnMuons 1
/CRY/input date 7-1-2012
/CRY/input latitude 48.0
/CRY/input altitude 0
/CRY/input subboxLength 0.2
/CRY/input nParticlesMin 1
/CRY/input nParticlesMax 2
/CRY/update

/control/execute vis.mac
/vis/viewer/set/viewpointThetaPhi 90. 0.
/vis/filtering/trajectories/create/particleFilter
/vis/filtering/trajectories/particleFilter-0/add mu+

/run/beamOn 1000
```

Initialize the run

Defined already in PrimaryGeneratorMessenger.cc

Specify the generated secondaries : 0== false && 1== true, in this case only muon are generated (we can specify the charge of particle in case we need only positive or only negative)

Date: month-day-year

Latitude: depend on the region

Altitude sea level=> all your geometry should be below zero (Z<0)

SubboxLength : usually same as active area of detector

Number of particles by event: one by event

For visualisation : view points, filter on particle : in this case we see only muon+ trajectories

Number of events

Change the subbox length to 1

PhysicsList

PhysicsList

- There are many different physics models, corresponding to a variety of approximations of the real phenomena.
- According to the application, one can be better than another.
- Simulation speed is important.
- A user can create their own PhysicsList.

- FTFP_BERT : Recommended physics list for High-Energy Physics.
  Its main components are :
  - FTF (Fritiof) hadronic string model, used above 3 GeV
  - BERT (Bertini-like) intra-nuclear cascade model, used below 6 GeV
  - Nucleus de-excitation : Precompound + evaporation models
  - Neutron capture
  - Nuclear capture of negatively charged hadrons at rest
  - Hadron elastic
  - Gamma- , electron- , and muon-nuclear
  - Standard electromagnetic physics

Few Other Physics Lists			
FTFP_BERT_HP	FTFP_INCLXX	QGSP_BERT	QGSP_BIC

Exercice 1 : Upload new Project

- **Upload** the new **B1_BNDScool1** and **placed** in the same directory of B1 basic example **(/home/usr/micromamba/envs/geant-root/share/Geant4-11.0.3/examples/basic/)**

CMackeLists.txt

- **Open CMackList.txt** : **change** the **path** to **CRY** (/home/usr/micromamba/envs/geant-root/cry_v1.7)

PrimaryGeneratorAction.cc

- **Open PrimaryGeneratorAction.cc** (in src directory) : **change** the **path** to the **data** of **CRY** (/home/usr/micromamba/envs/geant-root/cry_v1.7/data)

Save

- Open terminal
  - cd /home/usr/micromamba/envs/geant-root/share/Geant4-11.0.3/examples/basic/B1_BNDSchool1
  - mkdir build
  - cd build
  - micromamba activate geant-root
  - cmake ../
  - make
  - ./exampleB1

Exercice 2: Change the Physics List

exampleB1.cc

- Open exampleB1.cc:
  1. #include "FTFP_BERT.hh"
  2. Change your PhysicsList from QBBC to FTFP_BERT

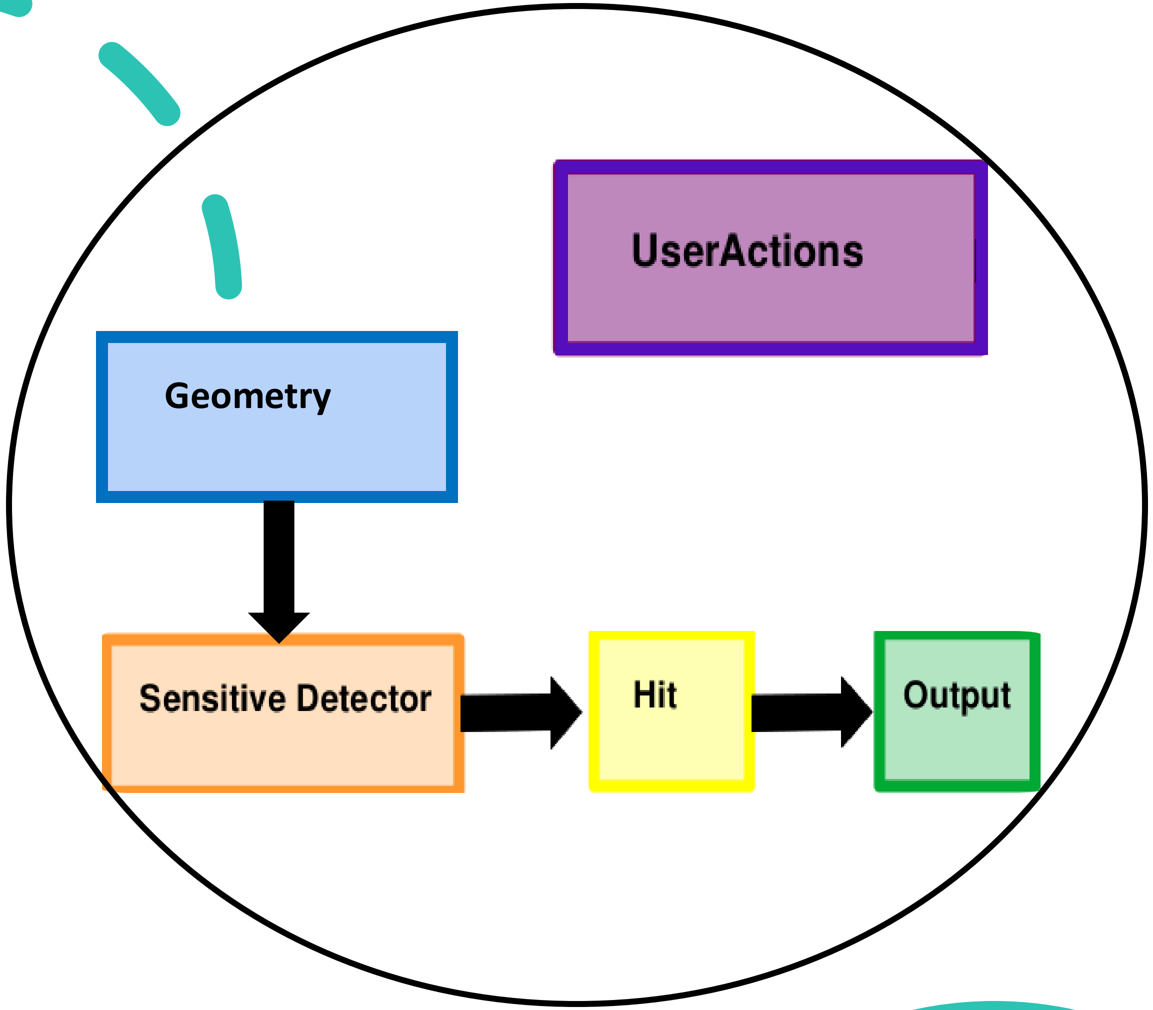
```
// Physics list
G4VModularPhysicsList* physicsList = new FTFP_BERT;
```

Save , then make and compile

Sensitive detector "SD" and Hits

Hit and sensitive detector

- A SD can be used to simulate the "read-out" of your detector:
  - A way to declare a geometric element "sensitive" to the passage of particles, for example : scintillator bar (for scintillator detector), gas-gap (for RPC detector)..
  - Gives the user a handle to collect physical quantities from the interaction of particle with those elements, for example : energy deposited, kinetic energy, position, time information ....

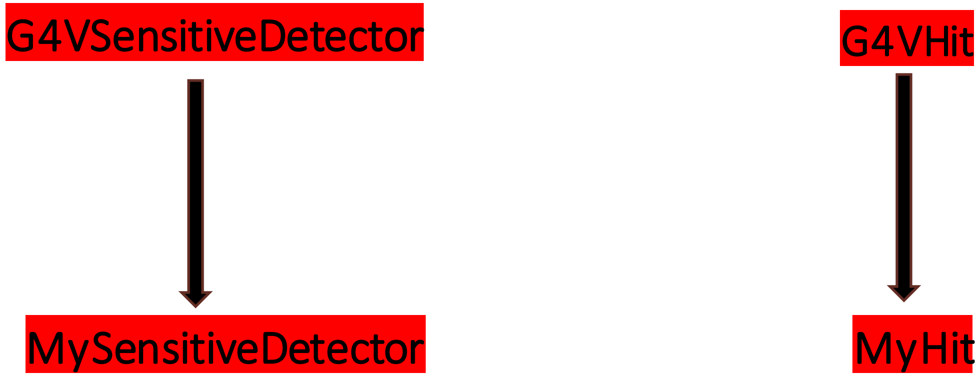


=>The principal mandate of a sensitive detector is the construction of hit objects using information from steps along a particle track.

Create a SD

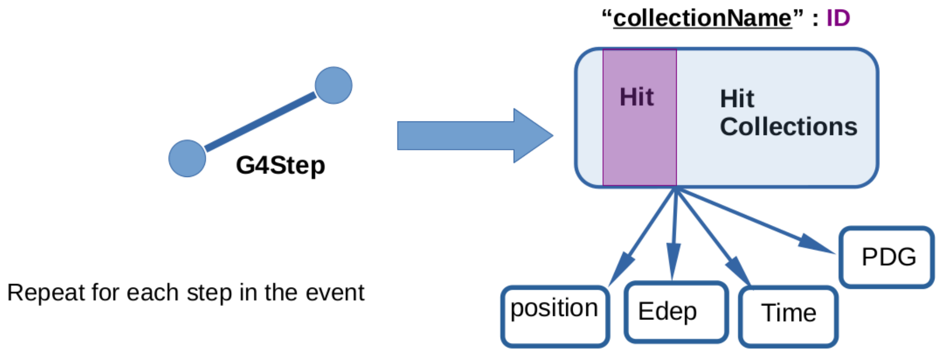
Strategy:

- Create your detector geometry
    - Solids, logical volumes, physical volumes
- Implement a sensitive detector and assign an instance of it to the logical volume of your geometry set-up
    - Then this volume becomes “sensitive”
    - Sensitive detectors are active for each particle steps, if the step starts in this volume
- Create hits objects in your sensitive detector using information from the particle step
    - You need to create the hit class(es) according to your requirements
- Store hits in hits collections (automatically associated to the G4Event object)
- Finally, process the information contained in the hit in user action classes (e.g.G4UserEventAction) to obtain results to be stored in the analysis object



Hits

- A hit is a snapshot of the physical interaction of a track in the sensitive region of a detector. In it you can store information associated with a G4Step object. This information can be:
    - the position and time of the step,
    - the momentum and energy of the track,
    - the energy deposition of the step,
    - geometrical information



- Hit is represented by **G4VHit** class that have 2 virtual methods

Draw(): To draw your concrete hits   **Print():** To print out your concrete hits

- During the processing of given event represented by G4Event, many objects of the hit class will be produced, collected and associated with the event.
- Therefore, for each concrete hit class you must also prepare a concrete class derived from **G4VHitsCollection**, a class which represents a vector collection of user defined hits.
- **G4THitsCollection**: is a template class derived from **G4VHitsCollection**, and the concrete hit collection class of a particular **G4VHit** concrete class can be instantiated from this template class.
- Each object of a **hit collection** must have a **unique name** for each event.
- **G4Event** has a **G4HCofThisEvent** class object, that is a **container class of collections of hits.** Hit collections are stored by their pointers, whose type is that of the base class.

Hits

Hit.hh

```
#ifndef Hits_h
#define Hits_h 1

#include "G4VHit.hh"
#include "G4THitsCollection.hh"
#include "G4Allocator.hh"
#include "G4ThreeVector.hh"

class G4AttDef;
class G4AttValue;

class PannelHit : public G4VHit
{
public:
  PannelHit();
  virtual ~PannelHit();

  inline void *operator new(size_t);
  inline void operator delete(void *aHit);

  virtual void Draw();
  virtual void Print();

  virtual const std::map<G4String,G4AttDef>* GetAttDefs() const;
  virtual std::vector<G4AttValue>* CreateAttValues() const;


  void SetPos( G4ThreeVector xyz ) { fPos = xyz; };
  G4ThreeVector GetPos() const { return fPos; };

  void SetPannelID( G4int z ) { fPannelID = z; };
  G4int GetPannelID() const { return fPannelID; };
```

Constructor and destructor of the class

Draw and print methods

Define the Set and Get method

```
private:
  G4int fPannelID;
  G4double fEdep;
  G4double fPDG;
  G4double fKe;
  G4double fPz;
  G4double fPx;
  G4double fPy;
  G4double fPhi;
  G4double fTime;
  G4ThreeVector fPos;
  G4int fTrackID;
};

typedef G4THitsCollection<PannelHit> PannelHitsCollection;

extern G4ThreadLocal G4Allocator<PannelHit>* PannelHitAllocator;

inline void* PannelHit::operator new(size_t)
{
  if (!PannelHitAllocator) {
      PannelHitAllocator = new G4Allocator<PannelHit>;
  }
  return (void*)PannelHitAllocator->MallocSingle();
}

inline void PannelHit::operator delete(void* aHit)
{
  PannelHitAllocator->FreeSingle((PannelHit*) aHit);
}


#endif
```

Define variables

Hit collection

Create a SD

Write your sensitive detector class using G4VSensitiveDetector.hh

```
#ifndef MySensitiveDetector_HH
#define MySensitiveDetector_HH

#include "G4VSensitiveDetector.hh"
#include "Hits.hh"

class G4Step;
class G4HCofThisEvent;
class G4TouchableHistory;

class MySensitiveDetector : public G4VSensitiveDetector
{
public:
    MySensitiveDetector(const G4String &SDname);
    ~MySensitiveDetector();


  virtual void Initialize( G4HCofThisEvent *hitcollection );
  virtual G4bool ProcessHits( G4Step *step, G4TouchableHistory *history );
  virtual void EndOfEvent( G4HCofThisEvent* hitCollection );

private:
  PannelHitsCollection* fHitsCollection;
  G4int fHCID;

};
#endif
```

MySensitiveDetector.hh

ProcessHits():

- This method is invoked by G4SteppingManager: when a step is composed in the G4LogicalVolume which has the pointer to this sensitive detector.
- The first argument of this method is a G4Step object of the current step.
- The second argument is a G4TouchableHistory object for the readout geometry described in the next slides.
- In this method, one or more G4VHit objects should be constructed if the current step is meaningful for your detector.

Initialize():

- This method is invoked at the beginning of each event. The argument of this method is an object of the G4HCofThisEvent class.
- Hit collections, where hits produced in this particular event are stored,can be associated with the G4HCofThisEvent object in this method.

EndOfEvent():

- This method is invoked at the end of each event.
- The argument of this method is the same object as Initialize() method.
- Hit collections occasionally created in your sensitive detector can be associated with the G4HCofThisEvent object.

Create a SD

```
MySensitiveDetector::MySensitiveDetector(const G4String &SDname) :
G4VSensitiveDetector(SDname),fHitsCollection(0), fHCID(-1)
{
   collectionName.insert("SensitiveDetectorColl");
}
MySensitiveDetector::~MySensitiveDetector()
{}
```

Specify a hits collection (by its unique name) for each type of hits considered in the sensitive detector: Insert the name(s) in the collectionName vector

```
void MySensitiveDetector::Initialize( G4HCofThisEvent *hitcollection )
{
  fHitsCollection = new PannelHitsCollection(SensitiveDetectorName, collectionName[0]);
  if ( fHCID < 0 ) {
    fHCID = G4SDManager::GetSDMpointer()->GetCollectionID(fHitsCollection);
  }
  hitcollection->AddHitsCollection( fHCID, fHitsCollection);
  G4cout << "Initialize PannelSD hitcoll ID: " << fHCID << G4endl;
}
```

The unique collection ID can be obtained with GetCollectionID():

- GetCollectionID() cannot be invoked in the constructor of this SD class (It is required that the SD is instantiated and registered to the SD manager first).
- we defined a private data member (collectionID), which is set at the first call of the Initialize() function.

The AddHitsCollection() method of G4HCofThisEvent requires the collection ID

```
G4bool MySensitiveDetector::ProcessHits(G4Step *aStep, G4TouchableHistory *R0hist)
{

   auto prestep = aStep->GetPreStepPoint();
   auto touchable = aStep->GetPreStepPoint()->GetTouchableHandle();
   auto track = aStep->GetTrack();
   const G4DynamicParticle* dynParticle = track->GetDynamicParticle();
   G4ThreeVector posMuon = prestep->GetPosition();
   G4double edep = aStep->GetTotalEnergyDeposit();
   G4double kinEnergy = track->GetDynamicParticle()->GetKineticEnergy();
   G4double Px = track->GetMomentumDirection().x();
   G4double Py = track->GetMomentumDirection().y();
   G4double Pz = track->GetMomentumDirection().z();
   G4double pdg = track->GetDefinition()->GetPDGEncoding();
   G4double time = prestep->GetGlobalTime();

   G4VPhysicalVolume* volume
   = aStep->GetPreStepPoint()->GetTouchableHandle()->GetVolume();

   G4int PannelCopyNo = volume->GetCopyNo();
   auto hit = new PannelHit();
   hit->SetPos(prestep->GetPosition());
   hit->SetEdep(edep);
   hit->SetPDG(pdg);
   hit->SetPannelID(PannelCopyNo);
   hit->Print();
   fHitsCollection->insert(hit);
   return true;
}
```

Generate hit(s) or accumulate data to existing hit objects, by using information from the current step (Use some methods for tracking detector)

Attach hit to each variable (implement in Hit class)

Insert the hit to the collection

```
void MySensitiveDetector::EndOfEvent( G4HCofThisEvent *hitCollection ) {}
```

Invoked at the end of each event

MySensitiveDetector.cc

Exercice 3: Attach sensitive detector to your detector pannel

DetectorConstruction.hh

- Open DetectorConstruction.hh:
  1. Add a **public virtual void** method **ConstructSDandField()** in the DetectorConstruction class:

```
public:
   virtual void ConstructSDandField();
```

DetectorConstruction.cc

- Open DetectorConstruction.cc :
  1. **Include** your **SD** header and **SDManager:**

```
#include "MySensitiveDetector.hh"
#include "G4SDManager.hh"
```

  2. **Call** your **ConstructSDandField()** method at the end of the code (outside DetectorConstruction::Construct()):

- Create an instance to your SD class,
- Register your SD to SDManager
- Assign the sensitive logic volume in your geometry

GetScoringVolume(): return fScoringVolume ==> logicDetector

```
void DetectorConstruction::ConstructSDandField()
{
  G4String SDname;
  G4SDManager::GetSDMpointer()->SetVerboseLevel(1);
  //Declare SensitiveDetector
  MySensitiveDetector *SensitiveDetector = new MySensitiveDetector("SensitiveDetector");

  G4SDManager::GetSDMpointer()->AddNewDetector(SensitiveDetector);
  GetScoringVolume()->SetSensitiveDetector(SensitiveDetector);
}
```

Exercice 4: Get Pannel ID

Hit.hh

- Open Hit.hh:

1. Add private variable:

```
private:
G4int fPannelID;
```

2. Define public Set and Get methods :

```
void SetPannelID( G4int z ) { fPannelID = z; };
G4int GetPannelID() const { return fPannelID; };
```

MySensitiveDetector.cc

- Open MySensitiveDetector.cc go to ProcessHits(G4Step *aStep, G4TouchableHistory *R0hist):
- Access to the physical volume of your SD :

```
G4VPhysicalVolume* volume   = aStep->GetPreStepPoint()->GetTouchableHandle()->GetVolume();
```

- Get the ID of volume using GetCopyNo:

```
G4int PannelCopyNo = volume->GetCopyNo();
```

- Set the Pannel Copy :

```
hit->SetPannelID(PannelCopyNo);
```

Exercice 5.a: Get Position information

- In Hit.hh:

1. I **already define** the variable and the Set && Get methods :

```
G4ThreeVector fPos;
```

&&

```
void SetPos( G4ThreeVector xyz ) { fPos = xyz; };
G4ThreeVector GetPos() const { return fPos; };
```

MySensitiveDetector.cc

- Open MySensitiveDetector.cc go to ProcessHit():
- **Use Steping** method to **follow the particle** :

```
auto prestep = aStep->GetPreStepPoint();
```

- **Get** the **position** of the **particle** at **each step**

```
G4ThreeVector posHit = prestep->GetPosition();
```

- **Set** the **position** :

```
hit->SetPos(posHit);
```

Exercice 5.b: Get PDG

- In Hit.hh:

1. I **already define** the variable and the Set && Get methods :

```
G4double fPDG;
```

&&

```
void SetPos( G4double id) { fPDG = id; };
G4double GetPDG() const { return fPDG; };
```

MySensitiveDetector.cc

- Open MySensitiveDetector.cc go to ProcessHit():
- Get your **track**

```
auto track = aStep->GetTrack();
```

- Get the PDG for this **track**

```
G4double pdg = track->GetDefinition()->GetPDGEncoding();
```

- Set the PDG:

```
hit->SetPDG(pdg);
```

Create a SD

```
namespace {

// Utility function which finds a hit collection with the given Id
// and print warnings if not found
G4VHitsCollection* GetHC(const G4Event* event, G4int collId) {

  auto hce = event->GetHCofThisEvent();
  if (!hce) {
      G4ExceptionDescription msg;
      msg << "No hits collection of this event found." << G4endl;
      G4Exception("EventAction::EndOfEventAction()",
                  "Code001", JustWarning, msg);
      return nullptr;
  }

  auto hc = hce->GetHC(collId);
  if ( ! hc) {
    G4ExceptionDescription msg;
    msg << "Hits collection " << collId << " of this event not found." << G4endl;
    G4Exception("EventAction::EndOfEventAction()",
                "Code001", JustWarning, msg);
  }
  return hc;
void EventAction::BeginOfEventAction(const G4Event* event)
{
 G4int evtNb = event->GetEventID();


  if (evtNb%fPrintModulo == 0)
    G4cout << "\n---> Begin of event: " << evtNb << G4endl;


 // Find hit collections by names (just once)
  if ( fPannelHCID == -1 ) {
    auto SDManager = G4SDManager::GetSDMpointer();
    fPannelHCID = SDManager->GetCollectionID("SensitiveDetector/SensitiveDetectorColl");
  }
  G4cout << "BeginOfEventAction fPannelHCID: " << fPannelHCID << G4endl;

}
 void EventAction::EndOfEventAction(const G4Event* event)
 {
    //event ID
    G4int evtNb = event->GetEventID();
    //Hit collection assigned to 1 event
    auto hc = GetHC(event, fPannelHCID);
    if ( !hc ) return;
    //number of Hits by events
    auto nhit = hc->GetSize();
    //Loop over n hit for each event=>interaction
   for ( unsigned long i = 0; i < nhit; i++ ) {
     auto hit = static_cast<PannelHit*>(hc->GetHit(i));
     auto pos = hit->GetPos();
    //Get hit information
   }
```

Retrive all hits collection

Retrive hits collection by index

EventAction.cc

Retrive index: index numbers of a hit collection are unique and don't change for a run

Pointer to the hit collection for each event

Size of the collection

Loop over individual hits and retrieve the data (using the Get methods defined in Hit class)

Store the output in analysis objects (ROOT output)

Exercice 6: Get Hit information

1. Open **EventAction** and go to **EndOfEventAction()**:
2. Go in **the loop** over the **number of hits** :
    1. Extract the x,y,z position of your hit in the detector and the pannel ID that your hit (particle) passes through it

```
G4cout << "X Position for hit : " << pos.x()
       << ", Y Position for hit : " << pos.y()
       << ", Z Position for hit :  " << pos.z()
       << ", Pannel:  " << hit->GetPannelID()
       << G4endl;
```

Save

3. Open cmd.file
    1. change number of beam to 5: /run/beam 5
    2. Save

4. make and compile :

1. make
2. ./exampleB1 cmd.file

What do you see in your terminal?

Exercice 6: Get Hit information

1. Open **EventAction** and go to **EndOfEventAction()**:
2. **Go** in **the loop** over the **number of generated particles**:
   1. Extract the name, PDG and momentum of your generated particle

```
G4cout << "TrackID: " << primary->GetTrackID()
        << ", Particle type: " << primary->GetG4code()->GetParticleName()
        << ", PDG encoding: " << primary->GetG4code()->GetPDGEncoding()
        << ", Momentum " << primary->GetMomentum()
        << G4endl;
```

Save

3. Open cmd.file
   1. change number of beam to 5: `/run/beam 5`
   2. Save

4. make and compile :

1. make
2. ./exampleB1 cmd.file

What do you see in your terminal?