Generation of a primary event

Annagrazia Varisano

INFN-LNS

Geant4 simulation code: theory and practical session

X Seminar on Software for Nuclear, Subnuclear and Applied Physics

Goal

- Learn how to use the G4ParticleGun and the **G**eneral **P**article **S**ource interfaces to generate primary particles in an event (particle type, energy, position, direction…)

  - The relevant class/method to look at are:
  - the class constructor
    G4VUserPrimaryGeneratorAction::G4VUserPrimaryGeneratorAction
  - the method

    G4VUserPrimaryGeneratorAction::GeneratePrimaries(G4Event)*

- Learn G4ParticleGun and **GPS** macro commands

Geant4 User Classes

- Geant4 does not provide the main().
- In our main, we have to:
  - Construct **G4RunManager**
  - Register User mandatory classes to **RunManager**

Initialisation classes

Invoked at the initialization via
G4RunManager::SetUserInitialization()

- G4VUserDetectorConstruction
- G4VUserPhysicsList

Action classes

Invoked during the execution loop via
G4RunManager::SetUserAction()

- G4VUserPrimaryGeneratorAction
- G4UserRunAction
- G4UserEventAction
- G4UserTrackingAction
- G4UserStackingAction
- G4UserSteppingAction

Three mandatory classes

The **PrimaryGeneratorAction.cc** class file is an 'Action' that must be defined.

G4VUserPrimaryGeneratorAction

- It Is one of the **mandatory** user classes, available for deriving your own concrete class:
- it controls the generation of primary particles
  - This class does not directly generate primaries but invokes the **GeneratePrimaryVertex()** method of a **generator** to create the primary
  - It registers the primary particles to the *G4Event* container

- **Constructor**
  - Instantiate primary generator (i.e. **G4ParticleGun()**)

    particleGun = new G4ParticleGun(n_particle);

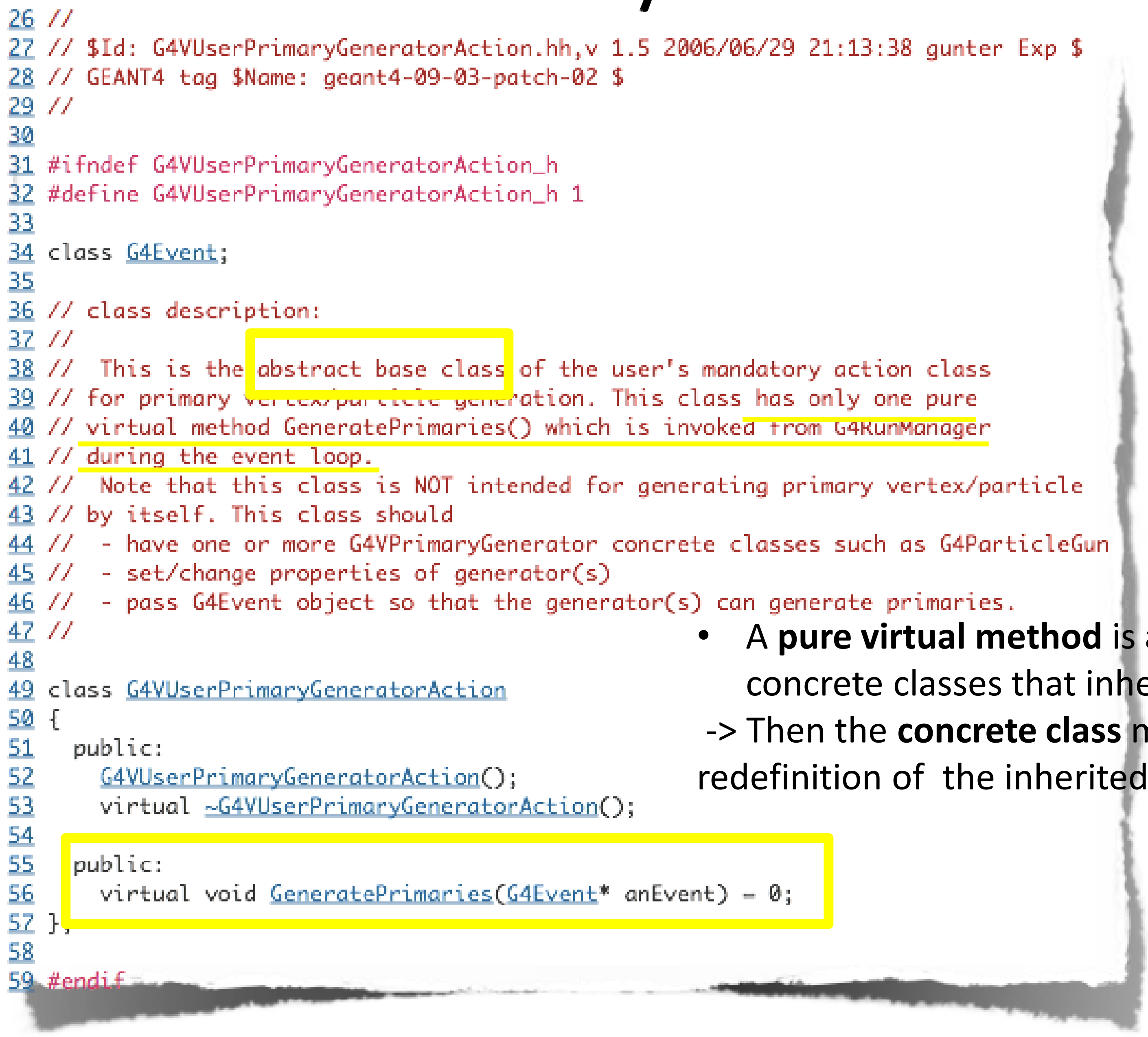
  - Set the default values (optional but advisable)

    particleGun -> SetParticleEnergy(1.0*GeV);

- It has ***GeneratePrimaries(G4Event* anEvent)*** method which is purely virtual, so it must be implemented in the user class
  - Randomise particle-by-particle value, if required
  - Set these values to primary generator
  - Invoke **GeneratePrimaryVertex()** method of primary generator

    particleGun -> GeneratePrimaryVertex(anEvent);

G4VUserPrimaryGeneratorAction (base class)

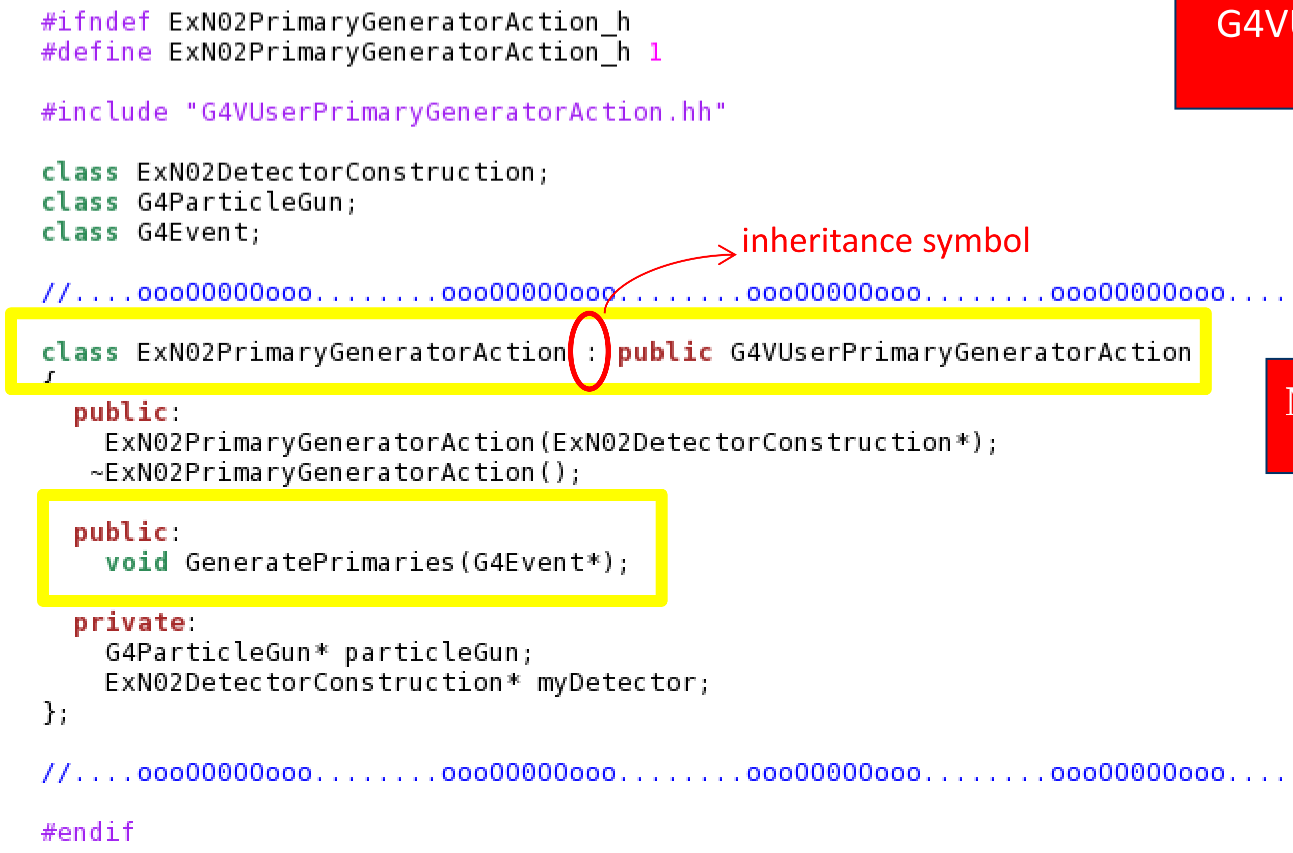


```
//
// $Id: G4VUserPrimaryGeneratorAction.hh,v 1.5 2006/06/29 21:13:38 gunter Exp $
// GEANT4 tag $Name: geant4-09-03-patch-02 $
//

#ifndef G4VUserPrimaryGeneratorAction_h
#define G4VUserPrimaryGeneratorAction_h 1

class G4Event;

// class description:
//
//  This is the abstract base class of the user's mandatory action class
// for primary vertex/particle generation. This class has only one pure
// virtual method GeneratePrimaries() which is invoked from G4RunManager
// during the event loop.
//  Note that this class is NOT intended for generating primary vertex/particle
// by itself. This class should
//  - have one or more G4VPrimaryGenerator concrete classes such as G4ParticleGun
//  - set/change properties of generator(s)
//  - pass G4Event object so that the generator(s) can generate primaries.
//

class G4VUserPrimaryGeneratorAction
{
  public:
    G4VUserPrimaryGeneratorAction();
    virtual ~G4VUserPrimaryGeneratorAction();

  public:
    virtual void GeneratePrimaries(G4Event* anEvent) = 0;
};

#endif
```

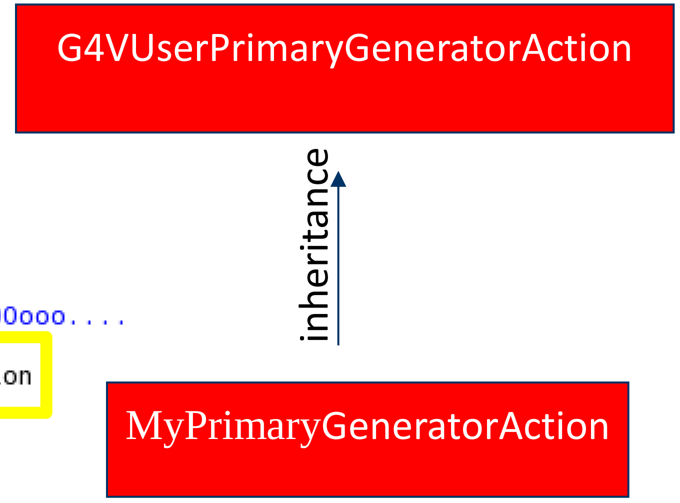
- A **pure virtual method** is an **interface** for concrete classes that inherit the base class.

-> Then the **concrete class** must make the redefinition of the inherited methods(**overriding**)

.... its concrete implementation



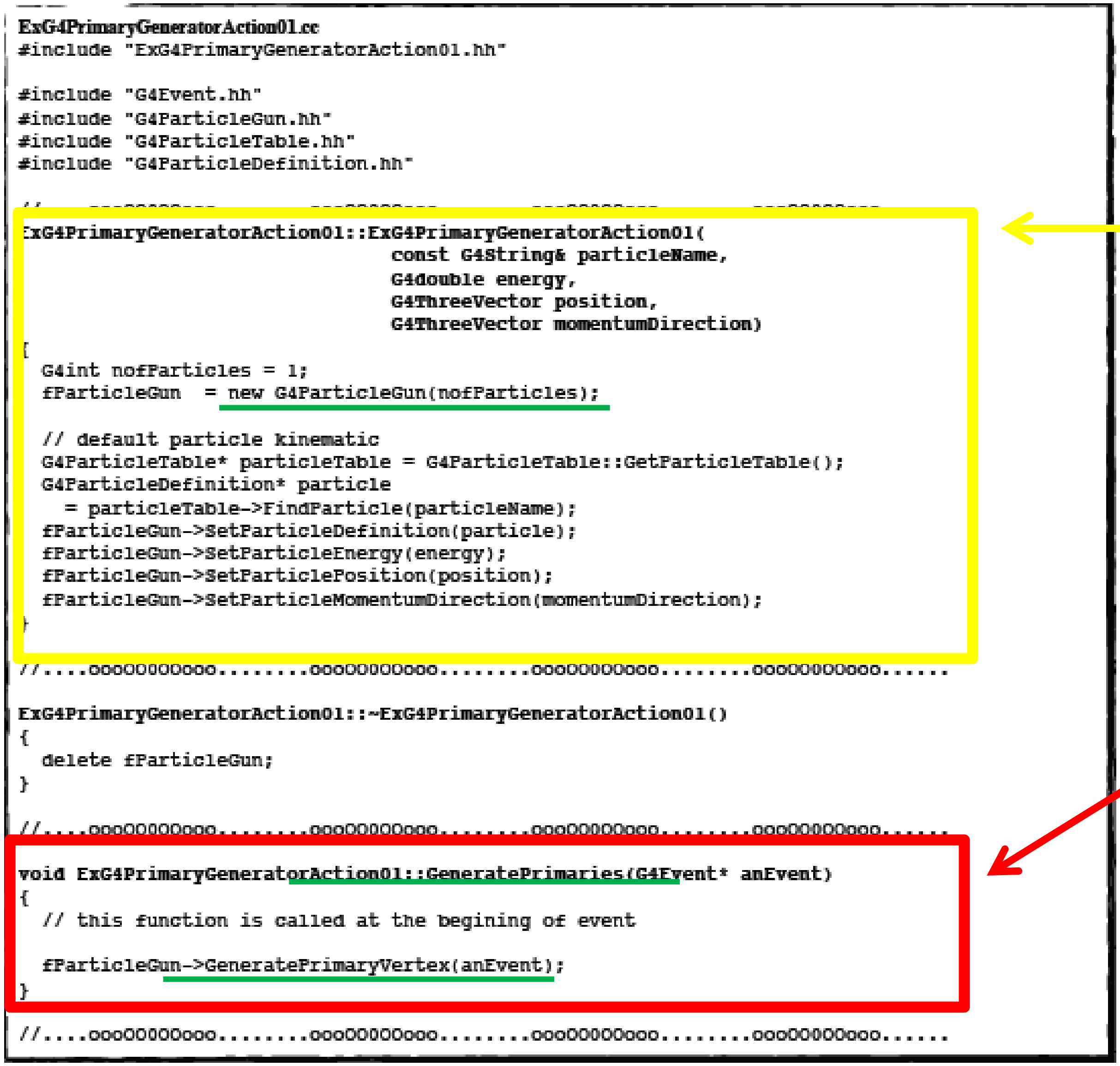
```
#ifndef ExN02PrimaryGeneratorAction_h
#define ExN02PrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"

class ExN02DetectorConstruction;
class G4ParticleGun;
class G4Event;

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

class ExN02PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
  public:
    ExN02PrimaryGeneratorAction(ExN02DetectorConstruction*);
   ~ExN02PrimaryGeneratorAction();

  public:
    void GeneratePrimaries(G4Event*);

  private:
    G4ParticleGun* particleGun;
    ExN02DetectorConstruction* myDetector;
};

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

#endif
```



If ***G4VUserPrimaryGeneratorAction*** class is abstract and ***MyPrimaryGeneratorAction*** class inherits from it, then the MyPrimaryGeneratorAction class must do the overriding of the virtual methods not implemented in G4VUserPrimaryGeneratorAction

MyPrimaryGeneratorAction

$$\begin{tikzcd}
\text{G4VUserPrimaryGeneratorAction} \\
\text{MyPrimaryGeneratorAction} \arrow[u, "\text{inheritance}"]
\end{tikzcd}$$



```
ExG4PrimaryGeneratorAction01.cc
#include "ExG4PrimaryGeneratorAction01.hh"

#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"

//....oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo......
ExG4PrimaryGeneratorAction01::ExG4PrimaryGeneratorAction01(
                                 const G4String& particleName,
                                 G4double energy,
                                 G4ThreeVector position,
                                 G4ThreeVector momentumDirection)
{
  G4int nofParticles = 1;
  fParticleGun  = new G4ParticleGun(nofParticles);

  // default particle kinematic
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle
    = particleTable->FindParticle(particleName);
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticleEnergy(energy);
  fParticleGun->SetParticlePosition(position);
  fParticleGun->SetParticleMomentumDirection(momentumDirection);
}

//....oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo......

ExG4PrimaryGeneratorAction01::~ExG4PrimaryGeneratorAction01()
{
  delete fParticleGun;
}

//....oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo......

void ExG4PrimaryGeneratorAction01::GeneratePrimaries(G4Event* anEvent)
{
  // this function is called at the begining of event

  fParticleGun->GeneratePrimaryVertex(anEvent);
}

//....oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo........oooOOOOOOOooo......
```

Constructor

- **•Constructor**
  - Instantiate primary generator
  - Set the default values

GeneratePrimaries()

The **G4VPrimaryGenerator** concrete class is instantiated via the **GeneratePrimaryVertex**() method

G4VPrimaryGenerator instantiated via the **GeneratePrimaryVertex()**

- *G4VPrimaryGenerator* is the base class for particle generators, that are invoked via the method **GeneratePrimaries(G4Event* aEvent)** to produce an initial state.

- We can instantiate more than one generator and/or invoke one generator more than once

- the logical step are: In ***G4VUserPrimaryGeneratorAction*** the **GeneratePrimaryVertex()** (pubblic method of G4ParticleGun) is invoked inside the **GeneratePrimaries(G4Event* aEvent)**

- Derived class from **G4VPrimaryGenerator** must implement the purely virtual method **GeneratePrimaryVertex()**

- Geant4 provides two concrete class derived by *G4VPrimaryGenerators*

  - G4ParticleGun
  - G4**G**eneral**P**article**S**ource

G4ParticleGun()

- Concrete implementation of G4VPrimaryGenerator, it is used to simulate a particles beam

  class G4ParticleGun:public G4VPrimaryGenerator
- It is provided by Geant4
- It does not provide any sort of randomisation
- Such randomisation can be achieved by the user, by invoking the 'Set' methods provided by **G4ParticleGun**
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction

  - Various "Set" methods are available (see../source/event/include/G4ParticleGun.hh)
- The methods must be invoked **inside GeneratePrimaries()** of G4VUserPrimarygeneratorActions before invoking GeneratePrimaryVertex()

Pubblic methods of G4ParticleGun

- void SetParticleDefinition(G4ParticleDefinition*)
- void SetParticleMomentum(G4ParticleMomentum)
- void SetParticleMomentumDirection(G4ThreeVector)
- void SetParticleEnergy(G4double)
- void SetParticleTime(G4double)
- void SetParticlePosition(G4ThreeVector)
- void SetParticlePolarization(G4ThreeVector)
- void SetNumberOfParticles(G4int)

```
void T01PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{ G4ParticleDefinition* particle;
  G4int i = (int)(5.*G4UniformRand());
  switch(i)
  { case 0: particle = positron; break; ... }
  particleGun->SetParticleDefinition(particle);
  G4double pp = momentum+(G4UniformRand()-0.5)*sigmaMomentum;
  G4double mass = particle->GetPDGMass();
  G4double Ekin = sqrt(pp*pp+mass*mass)-mass;
  particleGun->SetParticleEnergy(Ekin);
  G4double angle = (G4UniformRand()-0.5)*sigmaAngle;
  particleGun->SetParticleMomentumDirection
           (G4ThreeVector(sin(angle),0.,cos(angle)));
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

You can repeat this for generating more than one primary particles

G4**G**eneral**P**article**S**ource()

- Concrete implementation of **`G4VPrimaryGenerator`**
  `class G4GeneralParticleSource : public G4VPrimaryGenerator`
- It is designed to replace the G4ParticleGun class
- It is designed to allow specification of multiple particle sources each with independent definition of particle type, position, direction and energy distribution
  - Primary vertex can be randomly chosen on the surface of a certain volume
  - Momentum direction and kinetic energy of the primary particle can also be randomised
- Distribution defined by **UI commands**

/gps main command

/gps/pos/type (Sets the source positional distribution type: planar, point, etc.)

/gps/ang/type (Sets the angular distribution type to either isotropic, cosine-law or user-defined)

/gps/ene/type (Sets the energy distribution type: monoenergetic, linear, User defined)

....

On line manual: http://reat.space.qinetiq.com/gps/new_gps_sum_files/gps_sum.htm

•**Source:** point-like source, 100 MeV proton, along z

– **/gps/pos/type** point

– **/gps/particle** proton

– **/gps/energy** 100 MeV

– **/gps/direction** 0 0 1

ParticleGun vs. GPS

- G4ParticleGun
  - Simple and native
  - Shoots one track at a time
  - Easy to handle
- G4GeneralParticleSource
  - Powerful
  - Controlled by UI commands (G4GeneralParticleSourceMessenger.hh)
    - Almost impossible to control with set method
  - Capability of shooting particles from a surface of a volume
  - Capability of randomizing kinetic energy, position, direction following a user-specified distribution (histogram)

GPS is the choice if:

- If you need to shot primary particles from a surface of a complicated volume (outward or inward)
- If you need a complicated distribution

Examples

- examples/novice/N02 for G4ParticleGun
- examples/extended/analysis/A01/src/A01PrimaryGeneratorAction.cc is a good example to start
- Examples also exist for GPS examples/extended/eventgenerator/ exgps

A summary: what to do and where to do

- In the constructor of our UserPrimaryGeneratorAction
  - Instantiate **G4ParticleGun**
  - Set default values by Set methods of G4ParticleGun:
    - Particle type, kinetic energy, position and direction
- In your macro file or from your interactive terminal session
  - Set values for a run
- In the **GeneratePrimaries()** method
  - Shoot random numbers and prepare the values of
    - kinetic energy, position, direction ....
  - Use set methods of G4ParticleGun to set such values
  - Then invoke **GeneratePrimaryVertex()** method of G4ParticleGun
  - If you need more than one primary track per event, loop over randomisation and **GeneratePrimaryVertex()**