Primary Particles

Geant4 PHENIICS & IN2P3 Tutorial,

13 – 17 May 2019,

Orsay

Marc Verderi

LLR, Ecole polytechnique

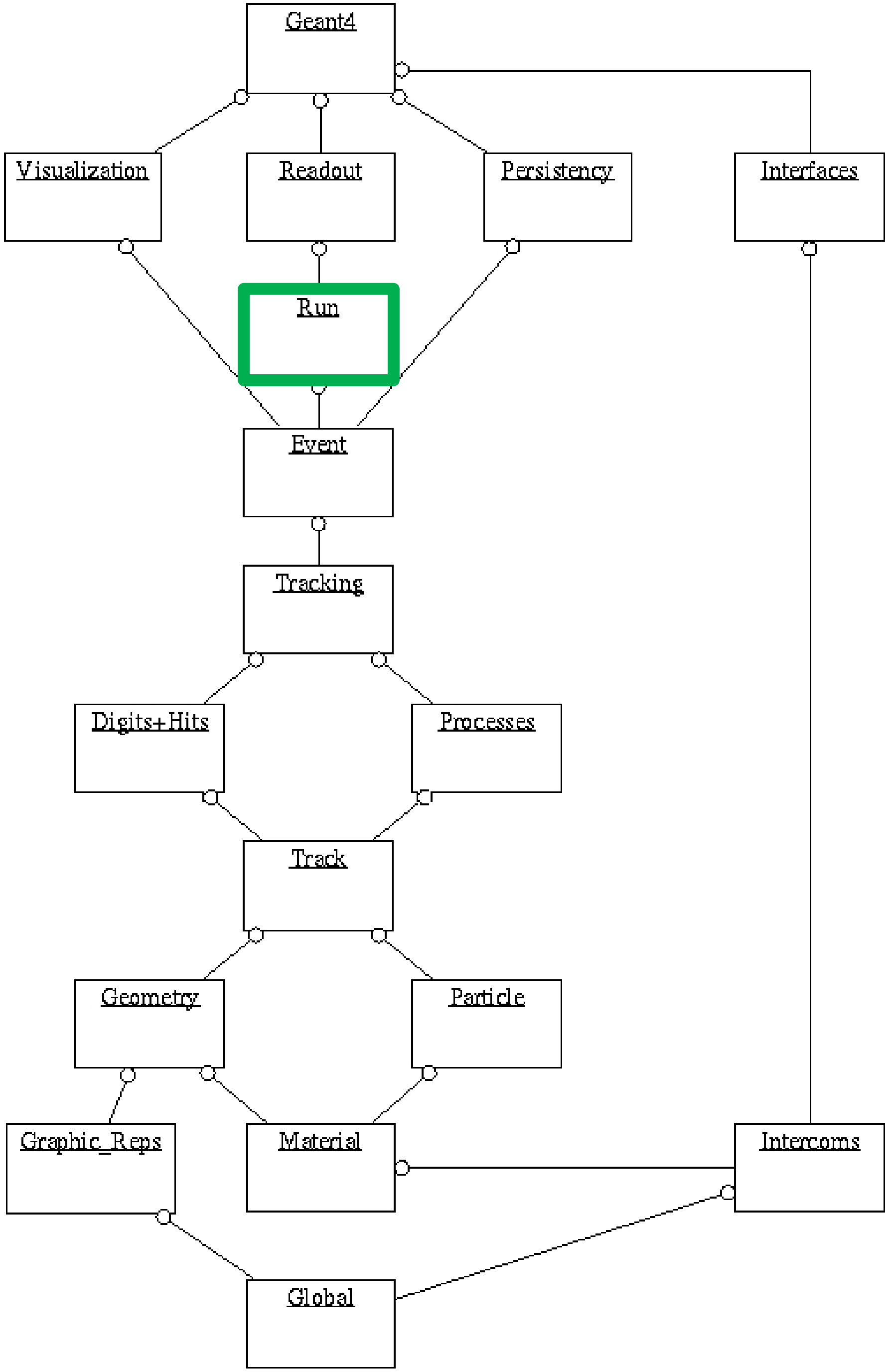
Credits…

- Filiation from at least Sébastien Incerti (CENBG), Makoto Asai, Tatsumi Koi, Dennis Wright (SLAC)
- And certainly other people !

Where will we look in the toolkit ?

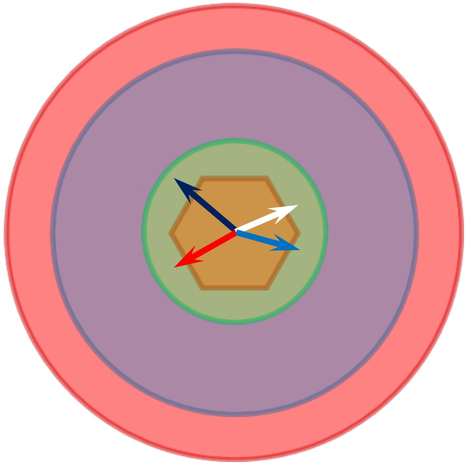
Main category and directory involved:

- Run
  - geant4/source/run



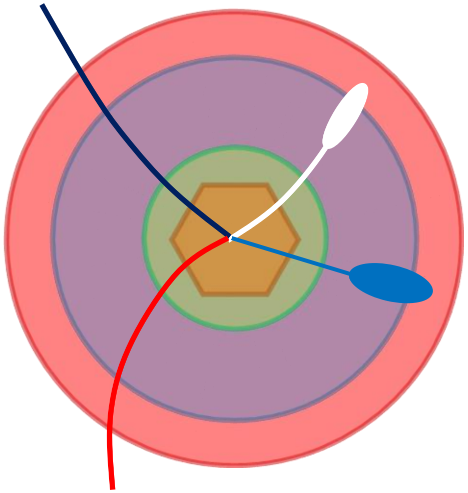
Introduction

- Here, “primary particles” stand for the particles you need to start with in your simulation at the beginning of each event:
  - For example:
    - Positrons in a PET scan imaging system in a medical application
    - Final state products in a proton-proton collision at the LHC



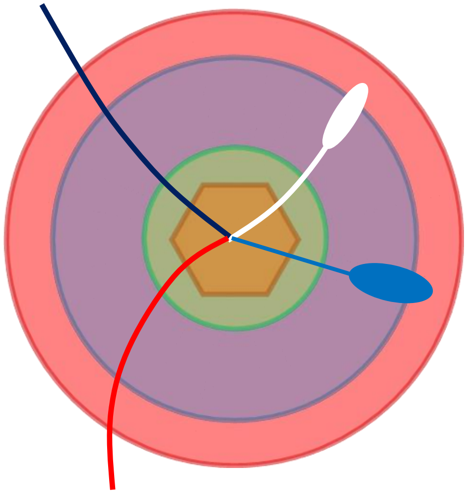
Introduction

- Here, “primary particles” stand for the particles you need to start with in your simulation at the beginning of each event:
  - For example:
    - Positrons in a PET scan imaging system in a medical application
    - Final state products in a proton-proton collision at the LHC



- These particles are then transported in your geometry…
  - … with interactions, creation of secondary particles…
  - … and related detector response.

Introduction



- Here, “primary particles” stand for the particles you need to start with in your simulation at the beginning of each event:
  - For example:
    - Positrons in a PET scan imaging system in a medical application
    - Final state products in a proton-proton collision at the LHC
- These particles are then transported in your geometry…
  - … with interactions, creation of secondary particles…
  - … and related detector response.
- The primary particles must be particles that Geant4 is able to track:
  - Ie : don’t ask Geant4 for tracking a Higgs boson or a SUSY particle !
    - Unless you have extended yourself the physics of Geant4 to do so…
  - But provide instead the decay products of these:
    - Eg : particles resulting from hadronisation of $b\bar{b}$ … or decays of ZZ in case of a Higgs, etc.
- You have to produce these primary particles taking some action
  - It is explained here how.

A mandatory operation

- Defining this action to produce “primary particles” is one of the three **mandatory** operations you have to do to make a working simulation.
  - Remember the two other mandatory operations:
    - detector construction: inheriting from **G4VUserDetectorConstruction**
    - physics list: inheriting from **G4VUserPhysicsList**

A mandatory operation

- Defining this action to produce “primary particles” is one of the three **mandatory** operations you have to do to make a working simulation.
  - Remember the two other mandatory operations:
    - detector construction: inheriting from **G4VUserDetectorConstruction**
    - physics list: inheriting from **G4VUserPhysicsList**

- You define this action inheriting from the **G4VUserPrimaryGeneratorAction** base class:
  - Let’s call your concrete class “MyPrimaryGeneratorAction”

A mandatory operation

- Defining this action to produce “primary particles” is one of the three **mandatory** operations you have to do to make a working simulation.
  - Remember the two other mandatory operations:
    - detector construction: inheriting from **G4VUserDetectorConstruction**
    - physics list: inheriting from **G4VUserPhysicsList**

- You define this action inheriting from the **G4VUserPrimaryGeneratorAction** base class:
  - Let’s call your concrete class “MyPrimaryGeneratorAction”
- To take effect, a MyPrimaryGeneratorAction object must be passed to the runManager:

A mandatory operation

- Defining this action to produce “primary particles” is one of the three **mandatory** operations you have to do to make a working simulation.
  - Remember the two other mandatory operations:
    - detector construction: inheriting from **G4VUserDetectorConstruction**
    - physics list: inheriting from **G4VUserPhysicsList**

- You define this action inheriting from the **G4VUserPrimaryGeneratorAction** base class:
  - Let’s call your concrete class “MyPrimaryGeneratorAction”
- To take effect, a MyPrimaryGeneratorAction object must be passed to the runManager:
    - **In your action initialization class:**

```
void MyActionInitialization::Build() const
{
  SetUserAction(new MyPrimaryGeneratorAction);
}
```

    - **And in your main program:**

```
runManager->SetUserInitialization(new MyActionInitialization);
```

G4RunManager* runManager = new G4RunManager;
or
G4MTRunManager* runManager = new G4MTRunManager;

  - if Geant4 version < Geant4 v10.0 (obsoleting):
    - **In your main program:**

```
runManager->SetUserAction(new MyPrimaryGeneratorAction);
```

[Kept > v10.0 for backward compatibility]

G4RunManager* runManager = new G4RunManager;

A mandatory operation

- Defining this action to produce “primary particles” is one of the three **mandatory** operations you have to do to make a working simulation.
  - Remember the two other mandatory operations:
    - detector construction: inheriting from **G4VUserDetectorConstruction**
    - physics list: inheriting from **G4VUserPhysicsList**

- You define this action inheriting from the **G4VUserPrimaryGeneratorAction** base class:
  - Let’s call your concrete class “MyPrimaryGeneratorAction”
- To take effect, a MyPrimaryGeneratorAction object must be passed to the runManager:

  - **In your action initialization class:**

```
void MyActionInitialization::Build() const
{
  SetUserAction(new MyPrimaryGeneratorAction);
}
```

  - **And in your main program:**

```
runManager->SetUserInitialization(new MyActionInitialization);
```

```
G4RunManager* runManager = new G4RunManager;
or
G4MTRunManager* runManager = new G4MTRunManager;
```

- if Geant4 version < Geant4 v10.0 (obsoleting):
  - **In your main program:**

```
runManager->SetUserAction(new MyPrimaryGeneratorAction);
```

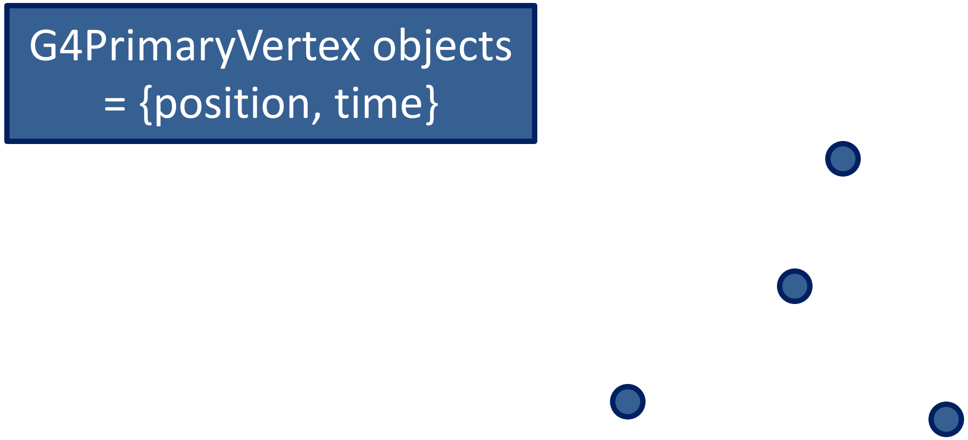
[Kept > v10.0 for backward compatibility]

```
G4RunManager* runManager = new G4RunManager;
```

- During the event loop, this action will be invoked at the ***beginning of each event***.
  - This invocation defines the start of the event.

Primary particle generation, in principle

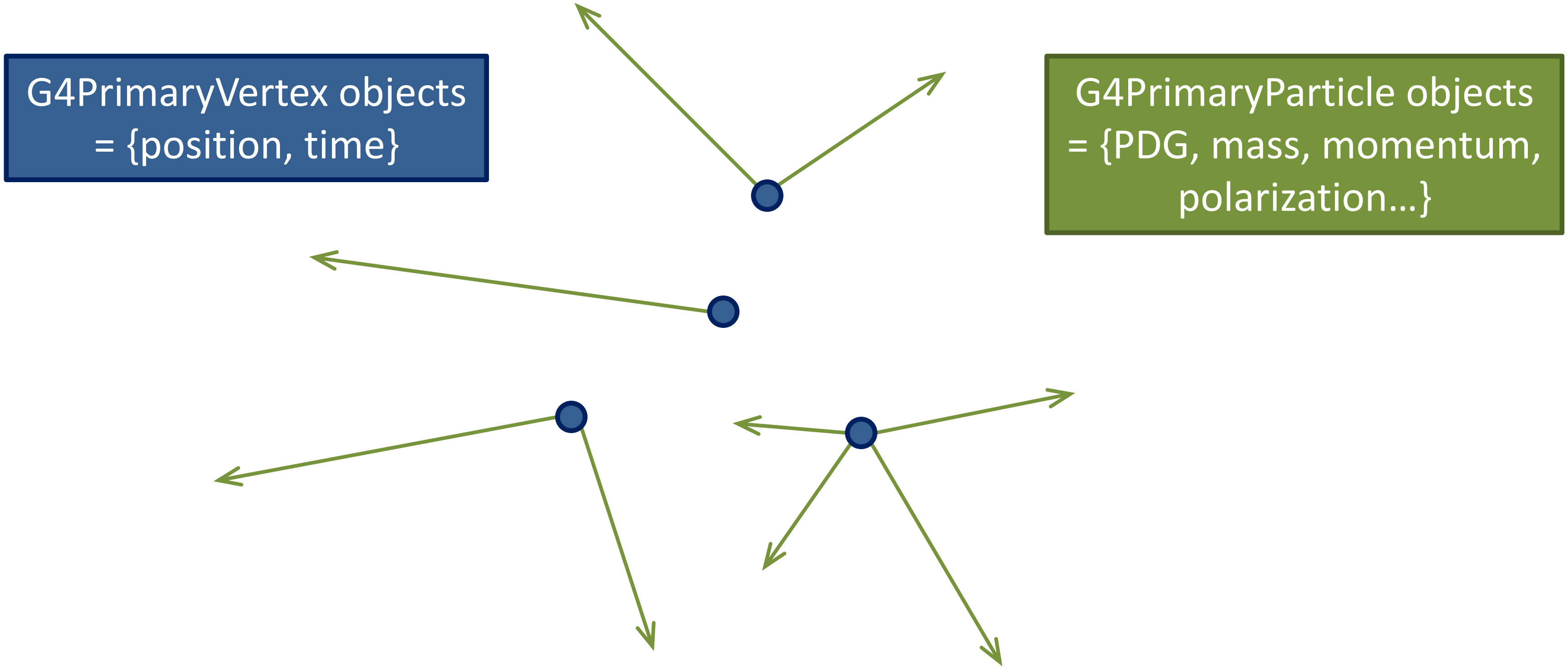
- For each event, you will define:



G4PrimaryVertex objects
= {position, time}

Primary particle generation, in principle

- For each event, you will define:

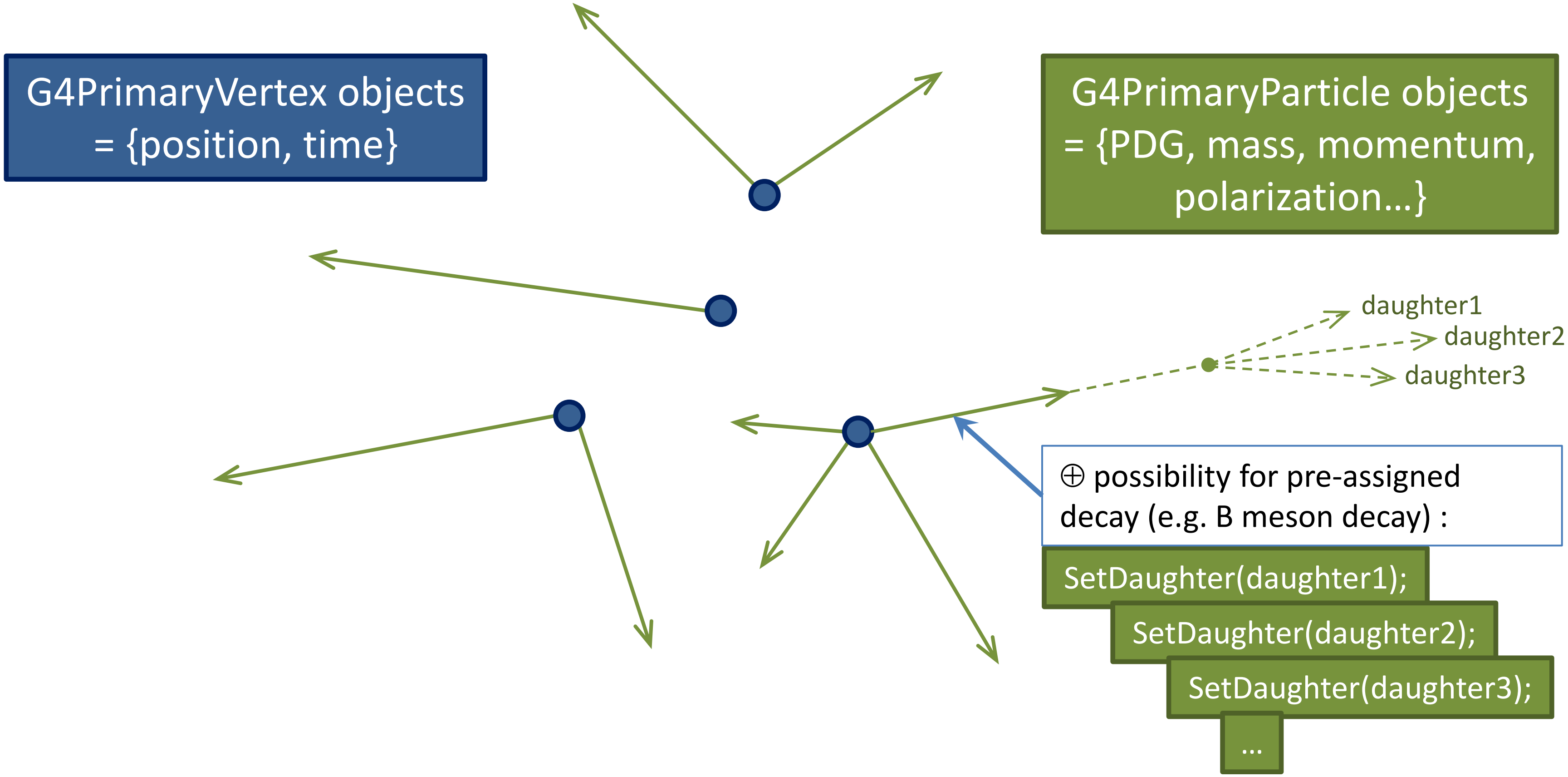


G4PrimaryVertex objects = {position, time}

G4PrimaryParticle objects = {PDG, mass, momentum, polarization…}

Primary particle generation, in principle

- For each event, you will define:



G4PrimaryVertex objects
= {position, time}

G4PrimaryParticle objects
= {PDG, mass, momentum, polarization…}

⊕ possibility for pre-assigned decay (e.g. B meson decay) :

SetDaughter(daughter1);

SetDaughter(daughter2);

SetDaughter(daughter3);

…

Primary particle generation, in practice (1/2)

- Primary particle generation is made by your concrete class, inheriting from base class:
  G4VUserPrimaryGeneratorAction

- The (pure virtual) method you must implement is
  void GeneratePrimaries(G4Event* event);
- In this method, you pass to "event" the G4PrimaryVertex objects you created,
  - to which you have attached the related G4PrimaryParticle objects.
- This is the method called at the beginning of each event.

First example of a G4VUserPrimaryGeneratorAction, with today’s sample code

```
void EDPrimaryGeneratorAction::GeneratePrimaries(G4Event* event)
{
   // Define particle properties
   G4String particleName = "proton";
   G4ThreeVector position(0, 0, -9.*m);
   G4ThreeVector momentum(0, 0, 1.*GeV);
   G4double time = 0;

   // Get particle definition from G4ParticleTable
   G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
   G4ParticleDefinition* particleDefinition = particleTable->FindParticle(particleName);
   if ( ! particleDefinition ) {
      G4cerr << "Error: " << particleName << " not found in G4ParticleTable" << G4endl;
      exit(1);
   }

   // Create primary particle
   G4PrimaryParticle* primaryParticle = new G4PrimaryParticle(particleDefinition);
   primaryParticle->SetMomentum(momentum.x(), momentum.y(), momentum.z());
   primaryParticle->SetMass(particleDefinition->GetPDGMass());
   primaryParticle->SetCharge( particleDefinition->GetPDGCharge());

   // Create vertex
   G4PrimaryVertex* vertex = new G4PrimaryVertex(position, time);
   vertex->SetPrimary(primaryParticle);
   event->AddPrimaryVertex(vertex);
}
```

Primary particle generation, in practice (2/2)

- Primary particle generation is made by your concrete class, inheriting from base class:
  G4VUserPrimaryGeneratorAction

- The (pure virtual) method you must implement is
  void GeneratePrimaries(G4Event* event);
- In this method, you pass to "event" the G4PrimaryVertex objects you created,
  - to which you have attached the related G4PrimaryParticle objects.
- This is the method called at the beginning of each event.

- In practice, actual vertices and particles creation is delegated to an other class
  G4VPrimaryGenerator
- Very recommended, as this makes easy re-use of code for generating primary particles
  - And several concrete implementations of these exist in Geant4 (see after)

- From G4VPrimaryGenerator, you may either
  - Inherit to implement your own, implementing the method
    void GeneratePrimaryVertex(G4Event* event);
  - Or use some of the existing concrete helper implementations (details later):
    - G4ParticleGun, G4GeneralParticleSource, G4SingleParticleSource, G4HEPEvtInterface

2nd example of a G4VUserPrimaryGeneratorAction, using a G4VPrimaryGenerator : G4ParticleGun (1/2)

Sample code of G4ParticleGun class.
It is defined in geant4 : you don’t have to provide it ! But just use it (see after).

```
void G4ParticleGun::GeneratePrimaryVertex(G4Event* evt)
{
   if(particle_definition==0) return;

   // create a new vertex
   G4PrimaryVertex* vertex = new G4PrimaryVertex(particle_position,particle_time);

   // create new primaries and set them to the vertex
  G4double mass =  particle_definition->GetPDGMass();
  for( G4int i=0; i<NumberOfParticlesToBeGenerated; i++ ){
     G4PrimaryParticle* particle = new G4PrimaryParticle(particle_definition);
     particle->SetKineticEnergy( particle_energy );
     particle->SetMass( mass );
     particle->SetMomentumDirection( particle_momentum_direction );
     particle->SetCharge( particle_charge );
     particle->SetPolarization(particle_polarization.x(), particle_polarization.y(), particle_polarization.z());
     vertex->SetPrimary( particle );
   }

   evt->AddPrimaryVertex( vertex );
}
```

2nd example of a G4VUserPrimaryGeneratorAction, using a G4VPrimaryGenerator : G4ParticleGun (2/2)

Constructor (ie, called once)

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
    G4int n_p
    fparticleGun  = new G4ParticleGun(n_particle);

    // default particle kinematic
    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
    fparticleGun->SetParticleDefinition(particle);
    fparticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
    fparticleGun->SetParticleEnergy(100.*MeV);
    fparticleGun->SetParticlePosition(G4ThreeVector(0.,0.,-50*cm));
}
```

class MyPrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    fparticleGun->GeneratePrimaryVertex(anEvent);
}
```

2nd example of a G4VUserPrimaryGeneratorAction, using a G4VPrimaryGenerator : G4ParticleGun (2/2)

Constructor (ie, called once)

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
    G4int n_particle = 1;
    fparticleGun  = new G4ParticleGun(n_particle);

    // default particle kinematic
    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
    fparticleGun->SetParticleDefinition(particle);
    fparticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
    fparticleGun->SetParticleEnergy(100.*MeV);
    fparticleGun->SetParticlePosition(G4ThreeVector(0.,0.,-50*cm));
}
```

G4ParticleGun: public G4VPrimaryGenerator

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    fparticleGun->GeneratePrimaryVertex(anEvent);
}
```

2nd example of a G4VUserPrimaryGeneratorAction, using a G4VPrimaryGenerator : G4ParticleGun (2/2)

Constructor (ie, called once)

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
     G4int n_particle = 1;
     fparticleGun  = new G4ParticleGun(n_particle);

     // default particle kinematic
     G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
     G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
     fparticleGun->SetParticleDefinition(particle);
     fparticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
     fparticleGun->SetParticleEnergy(100.*MeV);
     fparticleGun->SetParticlePosition(G4ThreeVector(0.,0.,-50*cm));
}
```

Initialization of this G4ParticleGun for shooting a same initial gamma (same E, from same $\vec{x}, \vec{p}$ …)

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
     fparticleGun->GeneratePrimaryVertex(anEvent);
}
```

2nd example of a G4VUserPrimaryGeneratorAction, using a G4VPrimaryGenerator : G4ParticleGun (2/2)

Constructor (ie, called once)

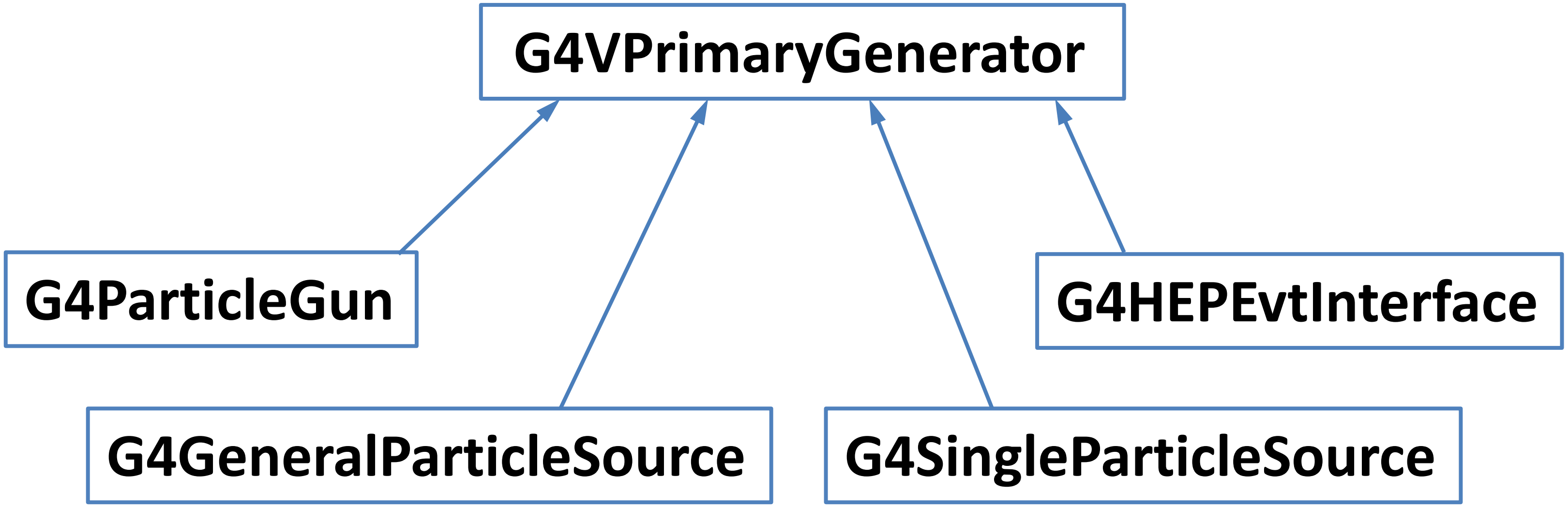
```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
    G4int n_particle = 1;
    fparticleGun  = new G4ParticleGun(n_particle);

    // default particle kinematic
    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
    fparticleGun->SetParticleDefinition(particle);
    fparticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
    fparticleGun->SetParticleEnergy(100.*MeV);
    fparticleGun->SetParticlePosition(G4ThreeVector(0.,0.,-50*cm));
}
```

Called at each event start

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    fparticleGun->GeneratePrimaryVertex(anEvent);
}
```

Built-in G4VPrimaryGenerator classes

- Geant4 provides concrete implementations for G4VPrimaryGenerator:



(used by G4GeneralParticleSource)

G4ParticleGun

- The simplest G4VPrimaryGenerator implementation:
  - Shoot one or several particle(s) at a time,
  - All of same fixed type, energy, momentum direction, position, time, etc.
- Particle gun configured with methods:

  SetNumberOfParticles(G4int)

  SetParticleDefinition(G4ParticleDefinition*)

  SetParticleMomentum(G4ParticleMomentum)

  SetParticleMomentumDirection(G4ThreeVector)

  SetParticleEnergy(G4double)

  SetParticleTime(G4double)

  SetParticlePosition(G4ThreeVector)

  SetParticlePolarization(G4ThreeVector)

- Simple, and a convenient tool to start with, and that can be used for more advanced and randomized generation (and example after).

- G4ParticleGun comes together with a messenger (it creates it):
  - Meaning that once you have created a G4ParticleGun object in memory, its messenger is also created, and you have access interactively to the menu:

    Idle > /gun/
  - With commands like: /gun/energy 10 MeV ; /gun/direction 0 0 1 ; etc…
  - You then just need to have the simple GeneratePrimaryVertex(anEvent) call in your MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent) , all the configuration of the particle gun will be done interactively.

Example with randomizing a direction

- Our first simple example: shooting a gamma, with particleGun fully configured in constructor of MyPrimaryGeneratorAction:

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    fparticleGun->GeneratePrimaryVertex(anEvent);
}                                  Note: case you can all configure your particle gun interactively
```

- An example of e^+e^- generation, with random direction (assumes the rest is configured in MyPrimaryGeneratorAction constructor or interactively) :

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    // shoot one electron (ie: add one electron to anEvent):
    fparticleGun->SetParticleDefinition(G4Electron::Definition());
    fparticleGun->SetParticleMomentum(G4RandomDirection());
    fparticleGun->GeneratePrimaryVertex(anEvent);
    // shoot one positron (ie: add one positron to anEvent):
    fparticleGun->SetParticleDefinition(G4Positron::Definition());
    fparticleGun->SetParticleMomentum(G4RandomDirection());
    fparticleGun->GeneratePrimaryVertex(anEvent);
}
```

An other example, more granular

- Previous example was generating particles uniformly in full angular space.
- If you need to focus the production in some angular space (not uniform here), you may do something like:

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    // shoot one electron (ie: add one electron to anEvent):
    fparticleGun->SetParticleDefinition(G4Electron::Definition());
    G4double dtheta = 10.*deg;
    G4double dphi   = 25.*deg;
    G4double theta  = G4UniformRand()*dtheta;
    G4double phi    = G4UniformRand()*dphi;
    G4ThreeVector randomDirection(sin(theta)*sin(phi),
                                  sin(theta)*cos(phi),
                                  cos(theta)));
    fParticleGun->SetParticleMomentumDirection(randomDirection);
    fparticleGun->GeneratePrimaryVertex(anEvent);
}
```

G4GeneralParticleSource (GPS)

- A more advanced implementation of G4VPrimaryGenerator
- It uses G4SingleParticleSource
  - Itself a G4VPrimaryGenerator
  - And which is an extended version of G4ParticleGun, allowing particles to be shoot according to distributions
- GPS Relies on the concept of "source"
  - The source emits the primary particles;
    - Of a given particle type
  - Sources can be combined with relative intensities to form a more advanced source.
    - Eg: built an Am/Be neutron + gamma source
- A source emits primary particles randomly according to
  - Position distribution
    - Ie the "source" distribution (point-like, surface, 3D…)
  - Energy, angular spectra
    - Built-in (uniform, exponential, gaussian, etc.)
    - Or user defined (providing an histogram-like data)
- Sources can be biased to enhance some phase space regions
  - And related statistical weight is provided

G4GeneralParticleSource (GPS)

- Using the GPS in your primary generator action:

```
MyPrimaryGeneratorAction::PrimaryGeneratorAction()
{
    fgps  = new G4GeneralParticleSource();
}

void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    fgps->GeneratePrimaryVertex(anEvent);
}
```

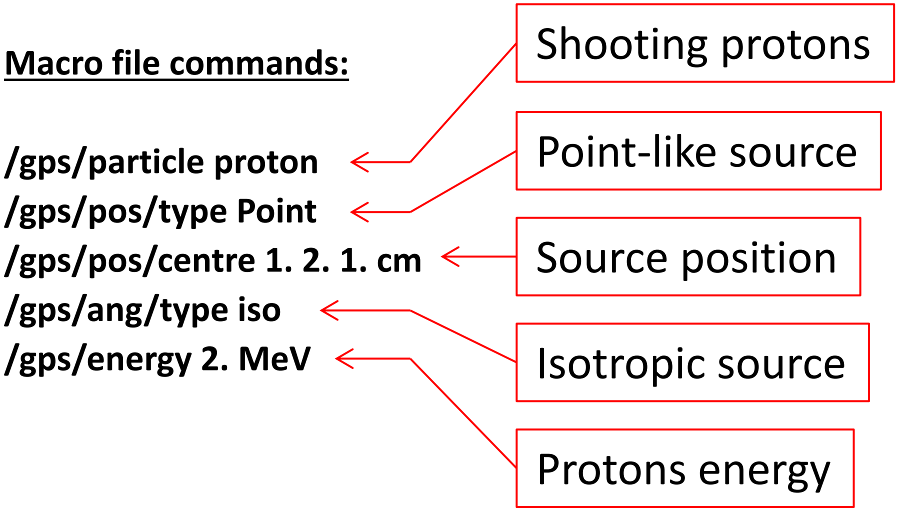
- As for the G4ParticleGun, GPS comes together with a messenger, which commands are under:

  Idle > /gps/

  - Which has a *rich* set of commands
  - All details can be found at:

    http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch02s07.html

geant4/examples/extended/eventgenerator/exgps/macros/ test1.g4mac : GPS Command Example 1



Macro file commands:

```
/gps/particle proton
/gps/pos/type Point
/gps/pos/centre 1. 2. 1. cm
/gps/ang/type iso
/gps/energy 2. MeV
```

Shooting protons

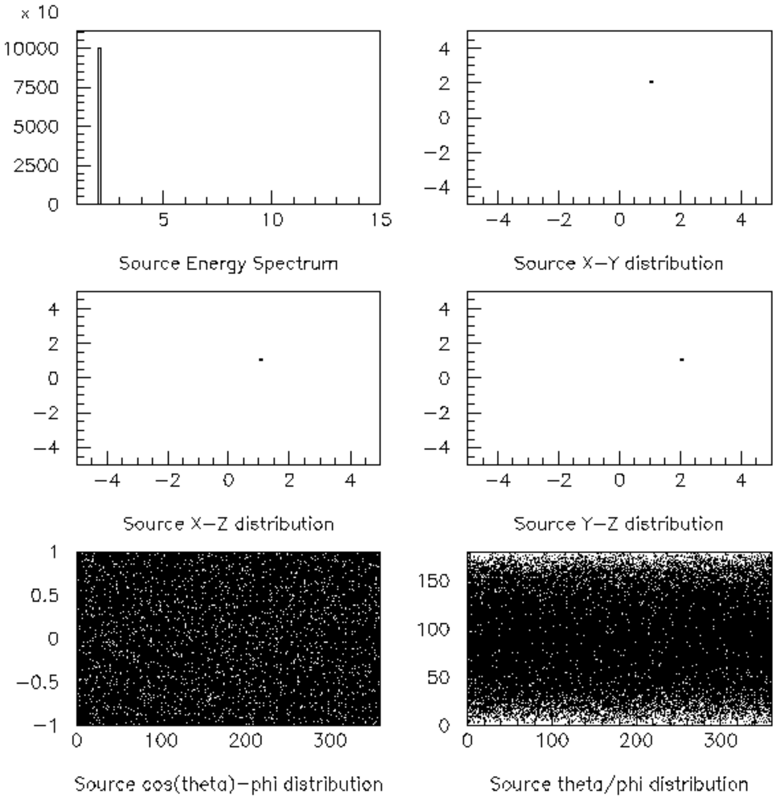
Point-like source

Source position

Isotropic source

Protons energy

Resulting distributions



Source Energy Spectrum

Source X−Y distribution

Source X−Z distribution

Source Y−Z distribution

Source cos(theta)−phi distribution

Source theta/phi distribution

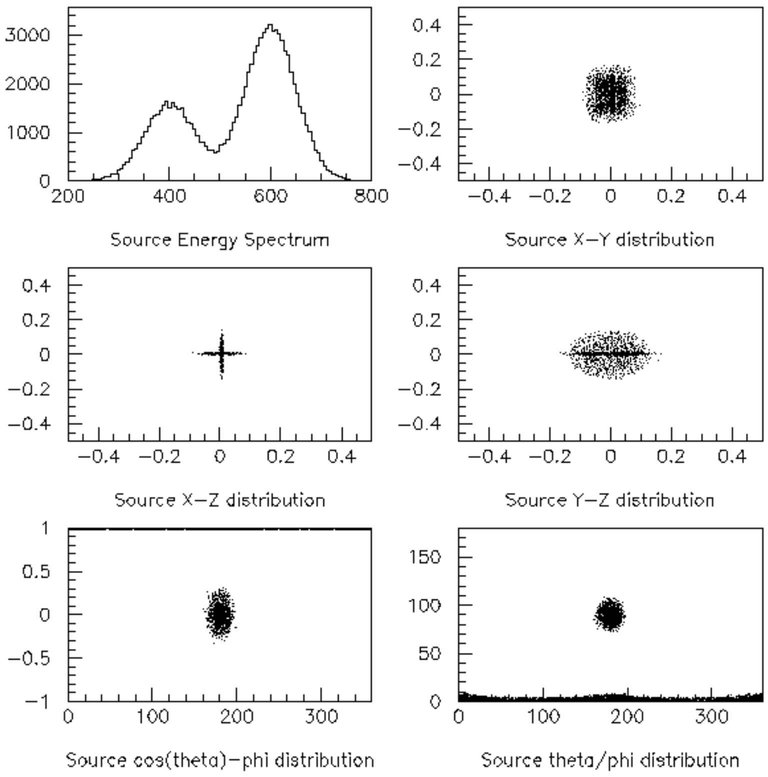
geant4/examples/extended/eventgenerator/exgps/macros/test31.g4mac : GPS Command Example 31

```
# two beams in a generator
#
# beam #1
# default intensity is 1 now change to 5.
/gps/source/intensity 5.
#
/gps/particle proton
/gps/pos/type Beam
#
# the incident surface is in the y-z plane
/gps/pos/rot1 0 1 0
/gps/pos/rot2 0 0 1
#
# the beam spot is centered at the origin and is of
# 1d gaussian shape with a 1 mm central plateau
/gps/pos/shape Circle
/gps/pos/centre  0. 0. 0. mm
/gps/pos/radius 1. mm
/gps/pos/sigma_r .2 mm
#
# the beam is travelling along the X_axis with
# 5 degrees dispersion
/gps/ang/rot1 0 0 1
/gps/ang/rot2 0 1 0
/gps/ang/type beam1d
/gps/ang/sigma_r 5. deg
#
# the beam energy is in gaussian profile
# centered at 400 MeV
/gps/ene/type Gauss
/gps/ene/mono 400 MeV
/gps/ene/sigma 50. MeV
```

(macro continuation…)

```
# beam #2
 # 2x the instensity of beam #1
 /gps/source/add 10.
 #
 # this is a electron beam
 /gps/particle e-
 /gps/pos/type Beam
 # it beam spot is of 2d gaussian profile
 # with a 1x2 mm2 central plateau
 # it is in the x-y plane centred at the orgin
 /gps/pos/centre  0. 0. 0. mm
 /gps/pos/halfx 0.5 mm
 /gps/pos/halfy 1. mm
 /gps/pos/sigma_x 0.1 mm
 # the spread in y direction is stronger
 /gps/pos/sigma_y 0.2 mm
 #
 #the beam is travelling along -Z_axis
 /gps/ang/type beam2d
 /gps/ang/sigma_x 2. deg
 /gps/ang/sigma_y 1. deg
 # gaussian energy profile
 /gps/ene/type Gauss
 /gps/ene/mono 600 MeV
 /gps/ene/sigma 50. MeV
```

Resulting distributions



Source Energy Spectrum

Source X−Y distribution

Source X−Z distribution

Source Y−Z distribution

Source cos(theta)−phi distribution

Source theta/phi distribution

Interfaces to HEPEvt and HepMC

- Interface implementations of G4VPrimaryGenerator to standard formats in HEP:
  - useful for experiment-specific primary generator implementation

- G4HEPEvtInterface:
  - Suitable to /HEPEVT/ common block, which many of (FORTRAN) HEP physics generators are compliant to
  - ASCII file input (4-vectors from HEP generator code)

- More can be found in geant4/examples/extended/eventgenerator:
  - Showing an interface to HepMC
    - which a few new (C++) HEP physics generators are compliant to
      - Eg : Pythia
  - ASCII file input or direct linking to a generator through HepMC

Summary

- User must derive from G4VUserPrimaryGeneratorAction and
  - Implement GeneratePrimaries(G4Event* anEvent)
  - Register it to the run manager
  - Very recommended : use internally a G4VPrimaryGenerator for actual particle generation

- Generators must be derived from G4VPrimaryGenerator
  - Implementing GeneratePrimaryVertex(G4Event* event);
  - G4PrimaryVertex objects will be generated
  - To which G4PrimaryParticle objects will be associated

- Some built-in generators are provided:
  - G4ParticleGun, for simple cases
  - G4GeneralParticleSource for more complex ones
  - Interface G4HEPEvtInterface

For information : what happens then to your “primary particles” ?

- After MyPrimaryGeneratorAction:: GeneratePrimaries(G4Event* anEvent) call :
  - Geant4 makes the conversion :
    - G4PrimaryVextex + G4PrimaryParticle objects → G4Tracks objects
    - Remember, G4Track has:
      - particle type information : mass, charge, PDG, etc.
      - dynamic information : position, time, energy, momentum, polarization, etc.
  - And puts these tracks on the urgent (= normal) stack
    - More on stacks later : for now, it is a stack of particle waiting for being tracked
- Then, the event simulation starts :
  - the G4Track object on top of the stack is popped up and tracked in your detector representation

- Why G4PrimaryVertex and G4PrimaryParticle, and not directly G4Tracks in GeneratePrimaries(G4Event* anEvent) ?
  - G4Track is (too) specific to Geant4, with other information of no meaning for the generation
    - “G4TouchableHistory” geometrical information
    - Or pointer to a G4Step, etc.
  - G4PrimaryVertex and G4PrimaryParticle are free from this G4-specific stuff, and hence offer easier interfacing to standard particle and vertex representations
    - Like HEPEvt, HEPMC, etc.