Geant4 Beginners Course
TUM University (Munich)
16-20 April 2018

Physics in Geant4:
Particles and processes

Giada Petringa
(LNS-INFN)
User classes (...continued)

At initialization

- G4VUserDetectorConstruction
- G4VUserPhysicsList
- G4VUserActionInitialization

Global: only one instance exists in memory, shared by all threads.

At execution

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

Thread-local: an instance of each action class exists for each thread.
Contents

• Physics in Geant4 – motivation
• Particles
• Processes
• Physics lists

...Part 2:
• Production cuts
• Electromagnetic / hadronic physics
• Crystalline / optical physics
Physics – the challenge

• Huge amount of different processes for various purposes (only a handful relevant)
• Competing descriptions of the same physics phenomena (necessary to choose)
  – fundamentally different approaches
  – balance between speed and precision
  – different parameterizations
• Hypothetical processes & exotic physics

Solution: Atomistic approach with modular physics lists
Part I: Particles
Particles: basic concepts

There are three levels of class to describe particles in Geant4:

**G4ParticleDefinition**

*Particle static properties:* name, mass, spin, PDG number, etc.

**G4DynamicParticle**

*Particle dynamic state:* energy, momentum, polarization, etc.

**G4Track**

Information for tracking in a detector simulation: position, step, current volume, track ID, parent ID, etc.
Definition of a particle

Geant4 provides \texttt{G4ParticleDefinition} daughter classes to represent a large number of elementary particles and nuclei, organized in six major categories:

\textit{leptons, mesons, baryons, bosons, short-lived and ions}

User must define \textbf{all particle} types which might be used in the application: not only \textbf{primary particles} but also all other particles which may appear as \textbf{secondaries} generated by the used physics processes.
Particles in Geant4

- Particle Data Group (PDG) particles
- Optical photons (different from gammas!)
- Special particles: geantino and charged geantino
  - Only transported in the geometry (no interactions)
  - Charged geantino also feels the EM fields
- Short-lived particles ($\tau < 10^{-14}$ s) are not transported by Geant4 (decay applied)
- Light ions (as deuterons, tritons, alphas)
- Heavier ions represented by a single class: G4Ions

# Leptons & bosons table

<table>
<thead>
<tr>
<th>Particle name</th>
<th>Class name</th>
<th>Name (in GPS...)</th>
<th>PDG</th>
</tr>
</thead>
<tbody>
<tr>
<td>electron</td>
<td>G4Electron</td>
<td>e-</td>
<td>11</td>
</tr>
<tr>
<td>positron</td>
<td>G4Positron</td>
<td>e+</td>
<td>-11</td>
</tr>
<tr>
<td>muon +/-</td>
<td>G4MuonPlus G4MuonMinus</td>
<td>mu+ mu-</td>
<td>-13 13</td>
</tr>
<tr>
<td>tauon +/-</td>
<td>G4TauPlus G4TauMinus</td>
<td>tau+ tau-</td>
<td>-15 15</td>
</tr>
<tr>
<td>electron (anti)neutrino</td>
<td>G4NeutrinoE G4AntiNeutrinoE</td>
<td>nu_e anti_nu_e</td>
<td>12 -12</td>
</tr>
<tr>
<td>muon (anti)neutrino</td>
<td>G4NeutrinoMu G4AntiNeutrinoMu</td>
<td>nu_mu anti_nu_mu</td>
<td>14 -14</td>
</tr>
<tr>
<td>tau (anti)neutrino</td>
<td>G4NeutrinoTau G4AntiNeutrinoTau</td>
<td>nu_tau anti_nu_tau</td>
<td>16 -16</td>
</tr>
<tr>
<td>photon (γ, X)</td>
<td>G4Gamma</td>
<td>gamma</td>
<td>22</td>
</tr>
<tr>
<td>photon (optical)</td>
<td>G4OpticalPhoton</td>
<td>opticalphoton</td>
<td>(0)</td>
</tr>
<tr>
<td>geantino</td>
<td>G4Geantino</td>
<td>geantino</td>
<td>(0)</td>
</tr>
<tr>
<td>charged geantino</td>
<td>G4ChargedGeantino</td>
<td>chargedgeantino</td>
<td>(0)</td>
</tr>
</tbody>
</table>
# Common hadrons & ions table

<table>
<thead>
<tr>
<th>Particle name</th>
<th>Class name</th>
<th>Name (in GPS...)</th>
<th>PDG</th>
</tr>
</thead>
<tbody>
<tr>
<td>(anti)proton</td>
<td>G4Proton, G4AntiProton</td>
<td>proton, anti_proton</td>
<td>2212, -2212</td>
</tr>
<tr>
<td>(anti)neutron</td>
<td>G4Neutron, G4AntiNeutron</td>
<td>neutron, anti_neutron</td>
<td>2112, -2112</td>
</tr>
<tr>
<td>(anti)lambda</td>
<td>G4Lambda, G4AntiLambda</td>
<td>lambda, anti_lambda</td>
<td>3122, -3122</td>
</tr>
<tr>
<td>pion</td>
<td>G4PionMinus, G4PionPlus, G4PionZero</td>
<td>pi-, pi+, pi0</td>
<td>-211, 211, 111</td>
</tr>
<tr>
<td>(anti)alpha</td>
<td>G4Alpha, G4AntiAlpha</td>
<td>alpha, anti_alpha</td>
<td>1000020040, -1000020040</td>
</tr>
<tr>
<td>(anti)deuteron</td>
<td>G4Deteuron, G4AntiDeuteron</td>
<td>deuteron, anti_deuteron</td>
<td>1000010020, -1000010020</td>
</tr>
<tr>
<td>Heavier ions</td>
<td>G4Ions</td>
<td>ion</td>
<td>100ZZZAAAAI*</td>
</tr>
</tbody>
</table>

*ZZZ=proton number, AAA=nucleon number, I=excitation level*
Part II: Processes
**From particles to processes**

- Propagated by the tracking
- Snapshot of the particle state

**G4Track**

**G4DynamicParticle**

Momentum, pre-assigned decay…

**G4ParticleDefinition**

The particle type: **G4Electron, ...**

**G4ProcessManager**

Container for all...

...relevant processes

**Process_1**

**Process_2**

**Process_3**

handled by kernel

Configured by the User in the "physics list"
Processes

How do particles interact with materials?

Responsibilities:

1. decide **when** and **where** an interaction occurs
   - **GetPhysicalInteractionLength**...()
   - \( \rightarrow \) limit the step
   - this requires a cross section
   - for the **transportation** process, the distance to the nearest object

2. generate the **final state** of the interaction
   - changes momentum, generates secondaries, etc.)
   - method: **DoIt**...()
   - this requires a model of the physics
G4VProcess class

- Abstract class as a base for all processes in Geant4
  - Used by all physics processes (also by the transportation, etc...)
  - Defined in `source/processes/management`
- Define three kinds of actions:
  - **AtRest** actions:
    - Decay, $e^+$ annihilation ...
  - **AlongStep** actions:
    - To describe continuous (inter)actions, occurring along the path of the particle, like ionisation;
  - **PostStep** actions:
    - For describing point-like (inter)actions, like decay in flight, hadronic interactions ...

A process can implement a combination of them (decay = AtRest + PostStep)
Example processes

- **Discrete process:** Compton Scattering, hadronic inelastic, ...
  - step determined by cross section, interaction at end of step
    - PostStepGPIL(), PostStepDoIt()
- **Continuous process:** Čerenkov effect
  - photons created along step, roughly proportional to step length
    - AlongStepGPIL(), AlongStepDoIt()
- **At rest process:** muon capture at rest
  - interaction at rest
    - AtRestGPIL(), AtRestDoIt()
- **Rest + discrete:** positron annihilation, decay, ...
  - both in flight and at rest
- **Continuous + discrete:** ionization
  - energy loss is continuous
  - knock-on electrons (δ-ray) are discrete
Handling multiple processes

1. A particle is shot and “transported”
2. All processes associated to the particle propose a geometrical step length (depends on process cross-section)
3. The process proposing the shortest step “wins” and the particle is moved to destination (if shorter than “Safety”)
4. All processes along the step are executed (e.g. ionization)
5. Post step phase of the process that limited the step is executed. New tracks are “pushed” to the stack
6. If $E_{\text{kin}} = 0$ all at rest processes are executed; if particle is stable the track is killed. Else:
7. New step starts and sequence repeats...
Part III:
Physics lists
Physics list

• One instance per application
  – registered to run manager in `main()`
  – inheriting from `G4VUserPhysicsList`

• Responsibilities
  – all particle types (electron, proton, gamma, ...)
  – all processes (photoeffect, bremsstrahlung, ...)
  – all process parameters (...)
  – production cuts (e.g. 1 mm for electrons, ...)

3 ways to get a physics list

1) Manual: Specify all particles & processes that may occur in the simulation. (difficult)

2) Physics constructors: Combine your physics from pre-defined sets of particles and processes. Still you define your own class – modular physics list (easier)

3) Reference physics lists: Take one of the pre-defined physics lists. You don't create any class (easy)
G4VUserPhysicsList class

Implement 3 methods:

class MyPhysicsList : public G4VUserPhysicsList {
public:
  // ...
  void ConstructParticle(); // pure virtual
  void ConstructProcess(); // pure virtual
  void SetCuts();
  // ...
}

Advantage: most flexible
Disadvantages:
  – most verbose
  – most difficult to get right
G4VUserPhysicsList: implementation

**ConstructParticle()**: 
– choose the particles you need in your simulation, define all of them here

**ConstructProcess()**: 
– for each particle, assign all the physics processes relevant to your simulation

**SetCuts()**: 
– set the range cuts for secondary production for processes with infrared divergence
i) **ConstructParticle()**

Due to the large number of particles can be necessary to instantiate, this method sometimes can be not so comfortable.

It is possible to define all the particles belonging to a **Geant4 category:**

- G4LeptonConstructor
- G4MesonConstructor
- G4BaryonConstructor
- G4BosonConstructor
- G4ShortlivedConstructor
- G4IonConstructor

```cpp
void MyPhysicsList::ConstructParticle() {
    G4Electron::ElectronDefinition();
    G4Proton::ProtonDefinition();
    G4Neutron::NeutronDefinition();
    G4Gamma::GammaDefinition();
    ....
}
```

```cpp
void MyPhysicsList::ConstructParticle() {
    // Construct all baryons
    G4BaryonConstructor bConstructor;
    bConstructor.ConstructParticle();
    // Construct all leptons
    G4LeptonConstructor lConstructor;
    lConstructor.ConstructParticle();
    // ...
}
```
ii) **ConstructProcess()**

1. For each particle, get its **process**

   ```cpp
   G4ProcessManager *elManager = G4Electron::ElectronDefinition()->GetProcessManager();
   ```

2. Construct all **processes** and register them.

   ```cpp
   elManager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
   elManager->AddProcess(new G4eIonisation, -1, 2, 2);
   elManager->AddProcess(new G4eBremsstrahlung, -1, -1, 3);
   elManager->AddDiscreteProcess(new G4StepLimiter);
   ```

3. Don’t forget **transportation**.
iii) SetCuts()

• Define all *production* cuts for *gamma*, *electrons* and *positrons*
  – Recently also for *protons*
• Notice: this is a *production cut*, not a tracking cut

MORE ON THIS LATER
Example: Put it together
G4VModularPhysicsList

• Similar structure as G4VUserPhysicsList (same methods to override – though not necessary):

```cpp
class MyPhysicsList : public G4VModularPhysicsList {
public:
    MyPhysicsList();           // define physics constructors
    void ConstructParticle();  // optional
    void ConstructProcess();   // optional
    void SetCuts();            // optional
}
```

Differences to “manual” way:
• Particles and processes typically handled by physics constructors (still customizable)
• Transportation automatically included
Physics constructor

= “module” of the modular physics list

• Inherits from G4VPhysicsConstructor
• Defines ConstructParticle() and ConstructProcess()
  – to be fully imported in modular list (behaving in the same way)
• GetPhysicsType()
  – enables switching physics of the same type, if possible (see next slide)
Physics constructors

• Huge set of pre-defined ones
  – **EM**: Standard, Livermore, Penelope
  – **Hadronic inelastic**: QGSP_BIC, FTFP_Bert, ...
  – **Hadronic elastic**: G4HadronElasticPhysics, ...
  – ... (decay, optical physics, EM extras, ...)

• You can implement your own (of course) by inheriting from the G4VPhysicsConstructor class
Using **physics constructors**

Add **physics constructor** in the **constructor**:

```cpp
MyModularList::MyModularList() {
    // Hadronic physics
    RegisterPhysics(new G4HadronElasticPhysics());
    RegisterPhysics(new G4HadronPhysicsFTFP_BERT_TRV());
    // EM physics
    RegisterPhysics(new G4EmStandardPhysics());
}
```

This already works and no further method overriding is necessary ☺️.
Replace physics constructors

You can **add** or **remove** the physics constructors after the list instance is created:
- e.g. in response to **UI command**
- only **before** initialization
- physics of the same type can be **replaced**

```cpp
void MyModularList::SelectAlternativePhysics() {
    AddPhysics(new G4OpticalPhysics);
    RemovePhysics(fDecayPhysics);
    ReplacePhysics(new G4EmLivermorePhysics);
}
```
Reference physics lists

• Pre-defined physics lists
  – already containing a complete set of particles & processes (that work together)
  – targeted at specific area of interest (HEP, medical physics, ...)
  – constructed as **modular physics lists**, built on top of **physics constructors**
  – customizable (by calling appropriate methods before initialization)
Alternatively: **Reference list by name**

- If you want to get reference physics lists by name (e.g. from environment variable), you can use the `G4PhysListFactory` class:

```cpp
#include "G4PhysListFactory.hh"
int main() {
   // Run manager
   G4RunManager* runManager = new G4RunManager();
   // E.g. get the list name from environment variable
   G4String listName{ getenv("PHYSICS_LIST") };
   auto factory = new G4PhysListFactory();
   auto physics = factory->GetReferencePhysList(listName);
   runManager->SetUserInitialization(physics);
   // ...
}
```
Lists of reference physics lists

Source code: $G4INSTALL/source/physics_lists/lists

FTF_BIC.hh
FTFP_BERT.hh
FTFP_BERT_HP.hh
FTFP_BERT_TRV.hh
FTFP_INCLXX.hh
FTFP_INCLXX_HP.hh
G4GenericPhysicsList.hh
G4PhysListFactoryAlt.hh
G4PhysListFactory.hh
G4PhysListRegistry.hh
G4PhysListStamper.hh
INCLXXPhysicsListHelper.hh
LBE.hh
NuBeam.hh
QBBC.hh
QGS_BIC.hh
QGSP_BERT.hh
QGSP_BERT_HP.hh
QGSP_BIC_AllHP.hh
QGSP_BIC.hh
QGSP_BIC_HP.hh
QGSP_FTFP_BERT.hh
QGSP_INCLXX.hh
QGSP_INCLXX_HP.hh
Shielding.hh
Where to find information?

User Support

1. Getting started
2. Training courses and materials
3. Source code
   a. Download page
   b. LXR code browser -or- draft doxygen documentation
4. Frequently Asked Questions (FAQ)
5. Bug reports and fixes
6. User requirements tracker
7. User Forum
8. Documentation
   a. Introduction to Geant4
   b. Installation Guide
   c. Application Developers Guide
   d. Toolkit Developers Guide
   e. Physics Reference Manual
9. Physics lists
   a. Electromagnetic
   b. Hadronic

3 ways to get a physics list (summary)

1) Manual: 
   
   G4VUserPhysicsList

2) Physics constructors:
   
   G4VModularPhysicsList

3) Reference physics lists:
   
   no class 😊
Conclusion

• Geant4 description of physics is very flexible
  – many particles
  – many processes
  – many models
  – many physics lists

...End of process