Physics in Geant4:
Particles and processes

Giada Petringa

Laboratori Nazionali del Sud (LNS)
Istituto Nazionale di Fisica Nucleare (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
VIII International Geant4 School
Belgrade-Serbia
18-22 November 2019

Part 1 : Particles
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 **G4ParticleDefinition** daughter classes to represent a large number of elementary particles and nuclei, organized in six major categories:

*leptons, mesons, baryons, bosons, short-lived and ions*

User must define **all particle** types which might be used in the application: not only **primary particles** but also all other particles which may appear as **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

## Methods to get particles properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G4String GetParticleName()</td>
<td>particle name</td>
</tr>
<tr>
<td>G4double GetPDGMass()</td>
<td>mass</td>
</tr>
<tr>
<td>G4double GetPDGWidth()</td>
<td>decay width</td>
</tr>
<tr>
<td>G4double GetPDGCharge()</td>
<td>electric charge</td>
</tr>
<tr>
<td>G4double GetPDGSpin()</td>
<td>spin</td>
</tr>
<tr>
<td>G4double GetPDGMagneticMoment()</td>
<td>magnetic moment (0: not defined or no magnetic moment)</td>
</tr>
<tr>
<td>G4int GetPDGiParity()</td>
<td>parity (0: not defined)</td>
</tr>
<tr>
<td>G4int GetPDGiConjugation()</td>
<td>charge conjugation (0: not defined)</td>
</tr>
<tr>
<td>G4double GetPDGIosospin()</td>
<td>iso-spin</td>
</tr>
<tr>
<td>G4double GetPDGIosospin3()</td>
<td>3rd-component of iso-spin</td>
</tr>
<tr>
<td>G4int GetPDGiGParity()</td>
<td>G-parity (0: not defined)</td>
</tr>
<tr>
<td>G4String GetParticleType()</td>
<td>particle type</td>
</tr>
<tr>
<td>G4String GetParticleSubType()</td>
<td>particle sub-type</td>
</tr>
<tr>
<td>G4int GetLeptonNumber()</td>
<td>lepton number</td>
</tr>
<tr>
<td>G4int GetBaryonNumber()</td>
<td>baryon number</td>
</tr>
<tr>
<td>G4int GetPDGEncoding()</td>
<td>particle encoding number by PDG</td>
</tr>
<tr>
<td>G4int GetAntiPDGEncoding()</td>
<td>encoding for anti-particle of this particle</td>
</tr>
</tbody>
</table>
### 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</td>
<td>mu+</td>
<td>-13</td>
</tr>
<tr>
<td></td>
<td>G4MuonMinus</td>
<td>mu-</td>
<td>13</td>
</tr>
<tr>
<td>tauon +/-</td>
<td>G4TauPlus</td>
<td>tau+</td>
<td>-15</td>
</tr>
<tr>
<td></td>
<td>G4TauMinus</td>
<td>tau-</td>
<td>15</td>
</tr>
<tr>
<td>electron (anti)neutrino</td>
<td>G4NeutrinoE</td>
<td>nu_e</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>G4AntiNeutrinoE</td>
<td>anti_nu_e</td>
<td>-12</td>
</tr>
<tr>
<td>muon (anti)neutrino</td>
<td>G4NeutrinoMu</td>
<td>nu_mu</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>G4AntiNeutrinoMu</td>
<td>anti_nu_mu</td>
<td>-14</td>
</tr>
<tr>
<td>tau (anti)neutrino</td>
<td>G4NeutrinoTau</td>
<td>nu_tau</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>G4AntiNeutrinoTau</td>
<td>anti_nu_tau</td>
<td>-16</td>
</tr>
<tr>
<td>photon ((\gamma, X))</td>
<td>G4Gamma</td>
<td>gamma</td>
<td>22</td>
</tr>
<tr>
<td>photon (optical)</td>
<td>G4 OpticalPhoton</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>G4 ChargedGeantino</td>
<td>chargedgeantino</td>
<td>(0)</td>
</tr>
<tr>
<td>Particle name</td>
<td>Class name</td>
<td>Name (in GPS...)</td>
<td>PDG</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>------------------</td>
<td>------------</td>
</tr>
<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>kaon</td>
<td>G4KaonMinus, G4KaonPlus, G4KaonZero, G4KaonZeroLong, G4KaonZeroShort</td>
<td>kaon-, kaon+, kaon0, kaon0L, kaon0S</td>
<td>311, 311, 130, 310</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>100ZZZAAAAl*</td>
</tr>
</tbody>
</table>

*ZZZ=proton number, AAA=nucleon number, l=excitation level
Part II: Processes
Physics processes describe how particles interact with a material. **Seven major categories of processes are provided by Geant4:**

- electromagnetic,
- hadronic,
- decay,
- photolepton-hadron,
- optical,
- parameterization,
- transportation.

**All physics processes are treated in the same manner from the tracking point of view.**
How do particles interact with materials?

Responsibilities:

1. *decide when and where an interaction occurs*
   - GetPhysicalInteractionLength...()
   - 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
**G4 VProcess 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**:
  - **G4VParticleChange**· **AtRest** actions:
    - Decay, $e^+$ annihilation …
  - **G4VParticleChange**· **AlongStep** actions:
    - To describe continuous (inter)actions, occurring along the path of the particle, like ionisation;
  - **G4VParticleChange**· **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 Model = final state generator

• Physics Process = cross section + model

• Physics List = list of processes for each particle
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, ...)
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:

```cpp
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
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 manager*.

   ```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);
   ```

   ```cpp
   AddTransportation();
   ```
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
void StandardPhysics::ConstructParticle()
{
    // We are interested in gamma, electrons and possibly positrons
    G4Electron::ElectronDefinition();
    G4Positron::PositronDefinition();
    G4Gamma::GammaDefinition();
}

void StandardPhysics::ConstructProcess()
{
    // Transportation is necessary
    AddTransportation();

    // Electrons
    G4ProcessManager *elManager = G4Electron::ElectronDefinition()->GetProcessManager();
    elManager->AddProcess(new G4MultipleScattering, -1, 1, 1);
    elManager->AddProcess(new G4Ionisation, -1, 2, 2);
    elManager->AddProcess(new G4Bremsstrahlung, -1, -1, 3);
    elManager->AddDiscreteProcess(new G4StepLimiter);

    // Positrons
    G4ProcessManager *posManager = G4Positron::PositronDefinition()->GetProcessManager();
    posManager->AddProcess(new G4MultipleScattering, -1, 1, 1);
    posManager->AddProcess(new G4Ionisation, -1, 2, 2);
    posManager->AddProcess(new G4Bremsstrahlung, -1, -1, 3);
    posManager->AddProcess(new G4plusAnnihilation, 0, -1, 4);
    posManager->AddDiscreteProcess(new G4StepLimiter);

    // Gamma
    G4ProcessManager *phManager = G4Gamma::GammaDefinition()->GetProcessManager();
    phManager->AddDiscreteProcess(new G4ComptonScattering);
    phManager->AddDiscreteProcess(new G4PhotoElectricEffect);
    phManager->AddDiscreteProcess(new G4GammaConversion);

    // TODO: Introduce Rayleigh scattering. It has large cross-section than Pair production
}

void StandardPhysics::SetCuts()
{
    // TODO: Create a messenger for this
defaultCutValue = 0.03 * mm;
    SetCutsWithDefault();
}
G4VModularPhysicsList

• Similar structure as G4VUserPhysicsList (same methods to override – though not 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
= “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)
• 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
Add **physics constructor** in the **constructor**:

```cpp
typedef 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 😊.
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)
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);
  // ...
}
```
List of reference physics list

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
G4PhysList Stamper.hh
INCLXXPhysListHelper.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 informations?

User Support
Submitted by Anonymous (not verified) on Wed, 06/28/2017 - 11:23

1. Getting started
2. Training courses and materials
3. Source code
   a. Download page
   b. LXR code browser
   c. doxygen documentation
   d. GitHub
   e. GitLab @ CERN
4. Frequently Asked Questions (FAQ)
5. Bug reports and fixes
6. User requirements tracker
7. User Forum
8. Documentation
   a. Introduction to Geant4 [pdf]
   b. Installation Guide: [pdf]
   c. Application Developers [pdf]
   d. Toolkit Developers Guide [pdf]
9. Examples
10. User Aids
    a. Tips for improving CPU performance
11. Contact Coordinators & Contact Persons

Related Links
- Object Oriented Analysis & Design
- Archive
- Mailing list subscription
- User requirements document (pdf)
- Technical Forum
3 ways to get a physics list (summary)


2) Physics constructors: G4VModularPhysicsList

3) Reference physics lists: no class 😊
Task 3

- Task 3a: Learn about processes and particles
- Task 3b: Physics constructors and physics lists
- Task 3c: Geant4 production cuts

Exercise 3a.1: Particle list and particle properties

Exercise 3a.2: Processes attached to a particle

Exercise 3a.3: Activate/inactivate processes
Task 3

- Task 3a: Learn about processes and particles
- Task 3b: Physics constructors and physics lists
- Task 3c: Geant4 production cuts

Exercise 3b.1: Change EM physics using Geant4 constructors
Exercise 3b.2: Add an additional EM constructor
Exercise 3b.3: Add hadronic physics