

Physics List: in a nutshell

Mihaly Novak (CERN, EP-SFT)

Getting Started with Geant4 at CERN, Geneva (Switzerland)

- What is a Physics List?
- The Geant4 Physics List interface
- Modular and reference Physics Lists

Physics List

WHAT IS A PHYSICS LIST?

- **Physics List** is an object that **is responsible to**:
 - **specify** all the **particles** that will be used in the simulation application
 - together with the list of **physics processes** assigned to them
- One out of the 3 **mandatory** objects that the user needs to provide to the `G4RunManager` in case of all Geant4 applications
- Provides a very **flexible** way to set up the **physics environment**
 - the user can chose, specify the particles that they want to be used
 - the user can chose the physics to assign to each particle
- **BUT**, the user must have a good understanding of the physics required to describe properly the given problem:
 - omission of relevant particles and/or physics interactions could lead to poor modelling results !!!

- **Why** do we need **Physics List**:
 - physics is physics - Shouldn't Geant4 provide, as default, a complete set of physics that everyone can use?
 - there are many **different approximations** and models to describe the same interaction: **computation time**
 - there is no any simulation application that would require all the particles, all their possible interactions that Geant4 can provide (also **alternative theories** to describe a given interaction)
- For this reason, Geant4 provides an **atomistic**, rather than an integral approach to physics::
 - provides many independent (for the most part) physics components i.e. physics **processes**
 - users select these components in their custom-designed physics lists

Physics List

THE GEANT4 PHYSICS LIST INTERFACE

- **G4VUserPhysicsList** is the Geant4 physics list interface:
 - user must implement the 2 pure virtual methods:
ConstructParticle and **ConstructProcess**
 - user can implement the **SetCuts** method (optional)

```
4  class YourPhysicsList: public G4VUserPhysicsList {
5      public:
6          // CTR
7          YourPhysicsList();
8          // DTR
9          virtual ~YourPhysicsList();
10
11         // pure virtual => needs to be implemented
12         virtual void ConstructParticle();
13         // pure virtual => needs to be implemented
14         virtual void ConstructProcess();
15
16         // virtual method
17         virtual void SetCuts();
18         ...
19         ...
20     };
```

- **ConstructParticle**: is the interface method to define the particles to be used in the simulation:
 - constructing them either individually

```
23 void YourPhysicsList::ConstructParticle() {
24     G4Electron::Definition();
25     G4Gamma::Definition();
26     G4Proton::Definition();
27     G4Neutron::Definition();
28     // other particle definitions
29     ...
30     ...
31 }
```

- or by using helpers

```
35 void YourPhysicsList::ConstructParticle() {
36     // construct baryons
37     G4BaryonConstructor baryonConstructor;
38     baryonConstructor.ConstructParticle();
39     // construct bosons
40     G4BosonConstructor bosonConstructor;
41     bosonConstructor.ConstructParticle();
42     // more particle definitions
43     ...
44     ...
45 }
```


- **ConstructProcess**: is the interface method to define the list of **physics processes** to be used in the simulation **for a given particle**:
 - **process**: an object that defines the way in which a given particle interacts with matter through a given type of interaction (e.g. *electron ionisation process*)

```
48 void YourPhysicsList::ConstructProcess() {
49     // method (provided by the G4VUserPhysicsList base class)
50     // that assigns transportation process to all particles
51     // defined in ConstructParticle()
52     AddTransportation();
53     // helper method might be defined by the user (for convenience)
54     // to add electromagnetic physics processes
55     ConstructEM();
56     // helper method might be defined by the user
57     // to add all other physics processes
58     ConstructGeneral();
59 }
```

```
62 void YourPhysicsList::ConstructEM() {
63     // get the physics list helper
64     // it will be used to assign processes to particles
65     G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
66     auto particleIterator = GetParticleIterator();
67     particleIterator->reset();
68     // iterate over the list of particles constructed in ConstructParticle()
69     while( (*particleIterator)() ) {
70         // get the current particle definition
71         G4ParticleDefinition* particleDef = particleIterator->value();
72         // if the current particle is the appropriate one => add EM processes
73         if ( particleDef == G4Gamma::Definition() ) {
74             // add physics processes to gamma particle here
75             ph->RegisterProcess(new G4GammaConversion(), particleDef);
76             ...
77             ...
78         } else if ( particleDef == G4Electron::Definition() ) {
79             // add physics processes to electron here
80             ph->RegisterProcess(new G4eBremsstrahlung(), particleDef);
81             ...
82             ...
83         } else if (...) {
84             // do the same for all other particles like e+, mu+, mu-, etc.
85             ...
86         }
87     }
88 }
```

```
62 void YourPhysicsList::ConstructEM() {
63     // get the physics list helper
64     // it will be used to assign processes to particles
65     G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
66     auto particleIterator = GetParticleIterator();
67     particleIterator->reset();
68     // iterate over the list of particles constructed in ConstructParticle()
69     while( (*particleIterator)() ) {
70         // get the current particle definition
71         G4ParticleDefinition* particleDef = particleIterator->value();
```

Too complicated for us!

```
76     ...
77     ...
78     } else if ( particleDef == G4Electron::Definition() ) {
79         // add physics processes to electron here
80         ph->RegisterProcess(new G4eBremsstrahlung(), particleDef);
81         ...
82         ...
83     } else if (...) {
84         // do the same for all other particles like e+, mu+, mu-, etc.
85         ...
86     }
87 }
88 }
```

Physics List

MODULAR AND REFERENCE PHYSICS LISTS

- **Modular physics list:**
 - significantly easier way to obtain/compose a physics list?
 - allows to use “physics modules”: a given physics module handles a well defined **category of physics** (e.g. EM, hadronic physics, decay, etc.)
 - transportation is automatically added to all constructed particles
- **Reference physics list:**
 - even easier way to obtain a complete physics list
 - these are “ready-to-use”, complete physics lists provided by the toolkit and constructed by expert developers
 - each pre-packaged physics list includes different combinations of EM and hadronic physics
 - note, these are physics lists used by some larger user groups like ATLAS, CMS, etc
 - given as it is: the user is responsible for validating them
 - see more details can be found in the [Guide for Physics Lists](#)

- **Example:** to use the `QGSP_BIC_HP` physics list with **EMZ** EM option

```
212 // IM YOUR MAIN APPLICATION
213 //
214 // create your run manager
215 #ifdef G4MULTITHREADED
216   G4MTRunManager* runManager = new G4MTRunManager;
217   // number of threads can be defined via macro command
218   runManager->SetNumberOfThreads(4);
219 #else
220   G4RunManager* runManager = new G4RunManager;
221 #endif
222 //
223 // create a physics list factory object that knows
224 // everything about the available reference physics lists
225 // and can replace their default EM option
226 G4PhysListFactory physListFactory;
227 // obtain the QGSP_BIC_HP_EMZ reference physics lists
228 // which is the QGSP_BIC_HP reference list with opt4 EM
229 const G4String plName = "QGSP_BIC_HP_EMZ";
230 G4VModularPhysicsList* pList = physListFactory.GetReferencePhysList(plName);
231 // (check that pList is not nullptr, that I skip now)
232 // register your physics list in the run manager
233 runManager->SetUserInitialization(pList);
234 // register further mandatory objects i.e. Detector and Primary-generator
235 ...
```

- **Example:** to use the `QGSP_BIC_HP` physics list with `EMZ` EM option

```
212 // IM YOUR MAIN APPLICATION
213 //
214 // create your run manager
215 #ifdef G4MULTITHREADED
216   G4MTRunManager* runManager = new G4MTRunManager;
217   // number of threads can be defined via macro command
```

This is what we will do!

```
222 //
223 // create a physics list factory object that knows
224 // everything about the available reference physics lists
225 // and can replace their default EM option
226 G4PhysListFactory physListFactory;
227 // obtain the QGSP_BIC_HP_EMZ reference physics lists
228 // which is the QGSP_BIC_HP reference list with opt4 EM
229 const G4String plName = "QGSP_BIC_HP_EMZ";
230 G4VModularPhysicsList* pList = physListFactory.GetReferencePhysList(plName);
231 // (check that pList is not nullptr, that I skip now)
232 // register your physics list in the run manager
233 runManager->SetUserInitialization(pList);
234 // register further mandatory objects i.e. Detector and Primary-generator
235 ...
```