Physics I : Physics Lists

Geant4 PHENIICS & ANF IN2P3 Tutorial,
22 – 26 May 2023,
Orsay

Marc Verderi
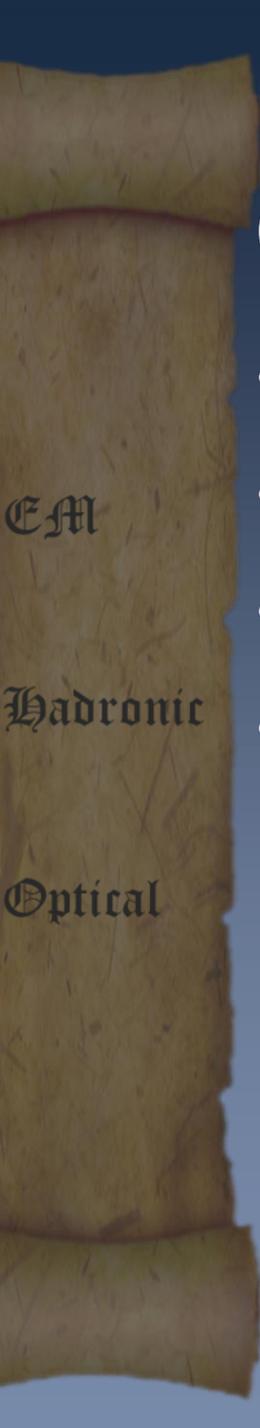LLR, Ecole polytechnique

EM

Hadronic

Optical

Credits...

› Daniel Brandt, Makoto Asai, Dennis Wright (SLAC),

› Gunter Folger (CERN), etc.

EM

Hadronic

Optical

Outline

- Introduction

- The G4VUserPhysicsList class

- Modular physics lists

- Pre-packaged or reference physics lists

EM

Hadronic

Optical

EM

Hadronic

Optical

Introduction

What is a physics list and why do we need one?

What is a Physics List?

› A class which collects all
  - the particles,
  - physics processes,
  - and production thresholds,

  needed for your application.

What is a Physics List?

› A class which collects all

  – the particles,

  – physics processes,

  – and production thresholds,

 needed for your application.

› It is passed to the run manager as the "physics configuration" of your application

EM

Hadronic

Optical

What is a Physics List?

› A class which collects all

– the particles,

– physics processes,

– and production thresholds,

needed for your application.

› It is passed to the run manager as the "physics configuration" of your application

› It is one of the three mandatory classes that must exist in your simulation:

– Remember, the two other ones are

› detector construction and

› primary generation action.

In many practical cases...

› You will not need the details in this presentation !

EM

Hadronic

Optical

In many practical cases…

› You will not need the details in this presentation !

EM

Hadronic

Optical

In many practical cases...

› You will not need the details in this presentation !

› Because Geant4 provides "pre-packaged physics lists"
  – Also called "reference physics lists"
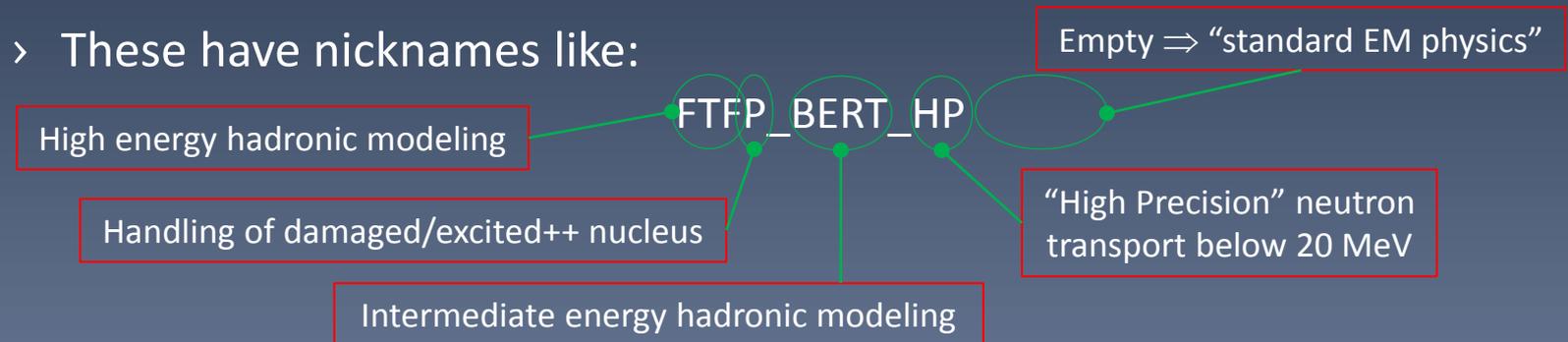
› These have nicknames like:

FTFP_BERT_HP

EM

Hadronic

Optical

In many practical cases…

› You will not need the details in this presentation !

› Because Geant4 provides "pre-packaged physics lists"
  – Also called "reference physics lists"

› These have nicknames like:

Empty $\Rightarrow$ "standard EM physics"

FTFP_BERT_HP

High energy hadronic modeling

Handling of damaged/excited++ nucleus

"High Precision" neutron transport below 20 MeV
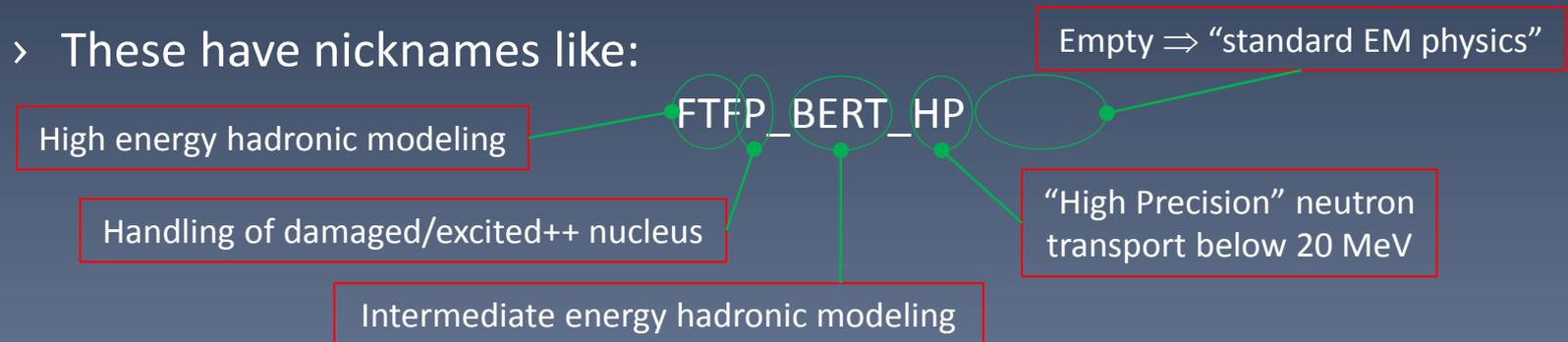
Intermediate energy hadronic modeling

EM

Hadronic

Optical

In many practical cases...

› You will not need the details in this presentation !

› Because Geant4 provides "pre-packaged physics lists"
  – Also called "reference physics lists"

› These have nicknames like:

FTFP_BERT_HP

Empty ⇒ "standard EM physics"

High energy hadronic modeling

Handling of damaged/excited++ nucleus

"High Precision" neutron transport below 20 MeV

Intermediate energy hadronic modeling

› They cover most of the "use cases", and are part of the routine testing and validation of Geant4
  – Reference physics lists are the most tested
  – Several of them are used in (heavy) production by experiments

› But sometimes users (you) want to customize their physics lists
  – This is possible in Geant4
  – At various levels of granularity
  – And for this, you need to know about "physics list".

EM

Hadronic

Optical

The G4VUserPhysicsList class

The fundamental class

EM

Hadronic

Optical

G4VUserPhysicsList

› **G4VUserPhysicsList** is the base class for physics list

› It defines three mandatory (pure virtual) methods:
- **ConstructParticle() :**
  › choose all the particles you need in your simulation
- **ConstructProcess() :**
  › for each particle, assign all the physics processes needed in your simulation
  › *What's a process ?*
    – a class that implements how a particle interacts with matter
    – more on this later
- **SetCuts() :**
  › set the range cuts for secondary production
  › *What's a range cut ?*
    – a production threshold for secondary particles
    – more on this later

› These methods are called by the "run manager".

G4VUserPhysicsList : an example

- Your physics list header hence looks like:

```cpp
#include "G4VUserPhysicsList.hh"

class MyPhysicsList: public G4VUserPhysicsList
 {
     public:
             MyPhysicsList();
             ~MyPhysicsList();
             void ConstructParticle();
             void ConstructProcess();
             void SetCuts();
 };
```

G4VUserPhysicsList : an example

- Your physics list header hence looks like:

```cpp
#include "G4VUserPhysicsList.hh"

class MyPhysicsList: public G4VUserPhysicsList
 {
     public:
             MyPhysicsList();
             ~MyPhysicsList();
             void ConstructParticle();
             void ConstructProcess();
             void SetCuts();
   };
```

- Let's look at these method implementations
  - and possible variations on these.

ConstructParticle() (1/2)

› You have several ways to implement this method.

› The most granular approach:

```
void MyPhysicsList::ConstructParticle()
 {

        G4Electron::ElectronDefinition();
        G4Proton::ProtonDefinition();
        G4Neutron::NeutronDefinition();
        G4Gamma::GammaDefinition();
        …

        …

 }
```

Eg : mandatory call to make the « gamma » particle type existing in memory

ConstructParticle() (2/2)

› A more global approach, using "constructors"
  – Utility classes that gather the proper G4XXX::XXXDefinition() calls

```cpp
void MyPhysicsList::ConstructParticle()
{
        G4BaryonConstructor* baryonConstructor =
                                new G4BaryonConstructor();
        baryonConstructor->ConstructParticle();
        delete baryonConstructor;

        G4BosonConstructor* bosonConstructor =
                                new G4BosonConstructor();
        bosonConstructor->ConstructParticle();
        delete bosonConstructor;
        …
        …
}
```

ConstructParticle() (2/2)

› A more global approach, using "constructors"
  – Utility classes that gather the proper G4XXX::XXXDefinition() calls

```cpp
void MyPhysicsList::ConstructParticle()
{
        G4BaryonConstructor* baryonConstructor =
                                new G4BaryonConstructor();
        baryonConstructor->ConstructParticle();
        delete baryonConstructor;

        G4BosonConstructor* bosonConstructor =
                                new G4BosonConstructor();
        bosonConstructor->ConstructParticle();
        delete bosonConstructor;
        …
        …
}
```

G4XXX::XXXDefinition() calls happen here

ConstructProcess()

› Process construction can also be made in several ways
  – Not showing everything here

› For convenience here, we split ConstructProcess() as:

```cpp
void MyPhysicsList::ConstructProcess()
{
        AddTransportation();
        // method provided by G4VUserPhysicsList : assigns transportation
        // process to all particles defined in ConstructParticle()

        ConstructEM();
        // method may be defined by user (for convenience)
        // put electromagnetic physics here

        ConstructGeneral();
        // method may be defined by user to hold all other processes
}
```

Transportation "process" cares about geometry and fields, and updates coordinates of particles

ConstructProcess() : ConstructEM()

```cpp
void MyPhysicsList::ConstructEM()
{
  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
  theParticleIterator->reset();
  while( (*theParticleIterator)() )
   {
     G4ParticleDefinition* particle = theParticleIterator->value();
     if  (particle == G4Gamma::Gamma() )
       {
         ph->RegisterProcess(new G4GammaConversion(),  particle);
         ….  // add more processes
       }
     …  // do electrons, positrons, etc.
   }
}
```

ConstructProcess() : ConstructEM()

```cpp
void MyPhysicsList::ConstructEM()
{
  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
  theParticleIterator->reset();
  while( (*theParticleIterator)() )
   {

     G4ParticleDefinition* particle = theParticleIterator->value();
     if  (particle == G4Gamma::Gamma() )
       {

        ph->RegisterProcess(new G4GammaConversion(),  particle);
         ….   // add more processes
       }
     …  // do electrons, positrons, etc.
   }
}
```

This is an helper tool : you can go even more granular, specifying the ordering execution of process methods.

EM

Hadronic

Optical

ConstructProcess() : ConstructGeneral()

```cpp
void MyPhysicsList::ConstructGeneral()
{

  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
  // Add decay process
  G4Decay* theDecayProcess = new G4Decay();
  theParticleIterator->reset();
  while( (*theParticleIterator)() )
    {
      G4ParticleDefinition* particle = theParticleIterator->value();
      if  (theDecayProcess->IsApplicable(*particle) )
        {
          ph->RegisterProcess(theDecayProcess, particle);
        }
    }
  // Add other physics
  ...
}
```

ConstructProcess() : SetCuts()

```cpp
void MyPhysicsList::SetCuts()
{
  defaultCutValue = 0.7*mm;
  SetCutValue(defaultCutValue, "gamma");
  SetCutValue(defaultCutValue, "e-");
  SetCutValue(defaultCutValue, "e+");
  SetCutValue(defaultCutValue, "proton");
  //
  // These are the production cuts you need to set
  // - not required for any other particle
}
```

EM

Hadronic

Optical

Modular physics lists

Building physics lists by "physics modules"

From flat list to "physics blocks"

- A realistic physics list has many particles and physics processes

  - Writing such a physics list as a "flat list" makes it long,

  - Complicated, hard to read,

  - And hard to maintain !

- Geant4 adopted a "modular physics list" approach:

  - Physics is organized by "physics modules"

    - EM physics, hadronic physics, optical physics, etc.

  - And you register the physics modules your are interested in in your "modular physics list"

- Still :

  - Starting from the base class G4VUserPhysicsList remains possible

  - You are free to use or not these functionalities

  - Again: this is a "toolkit" approach

G4VModularPhysicsList

› Derived from G4VUserPhysicsList
  – It is a G4VUserPhysicsList in the C++ sense
  – Ie, in header file of this class you find:
    › class G4VModularPhysicsList : public G4VUserPhysicsList {...};

› A G4VModularPhysicsList object registers "physics modules", eg:
  someModularPhysicsList->Register(new DecayPhysics());

G4VModularPhysicsList

› Derived from G4VUserPhysicsList
  – It is a G4VUserPhysicsList in the C++ sense
  – Ie, in header file of this class you find:
    › class G4VModularPhysicsList : public G4VUserPhysicsList {…};

› A G4VModularPhysicsList object registers "physics modules", eg:
  someModularPhysicsList->Register(new DecayPhysics());

› The "physics module" class itself is:
  ### G4VPhysicsConstructor

› Which defines two methods:
  – **virtual void ConstructParticle();**
    › As with G4VUserPhysicsList : it is to declare the particle types used
  – **virtual void ConstructProcess();**
    › As with G4VUserPhysicsList : it is to associate processes to above particles

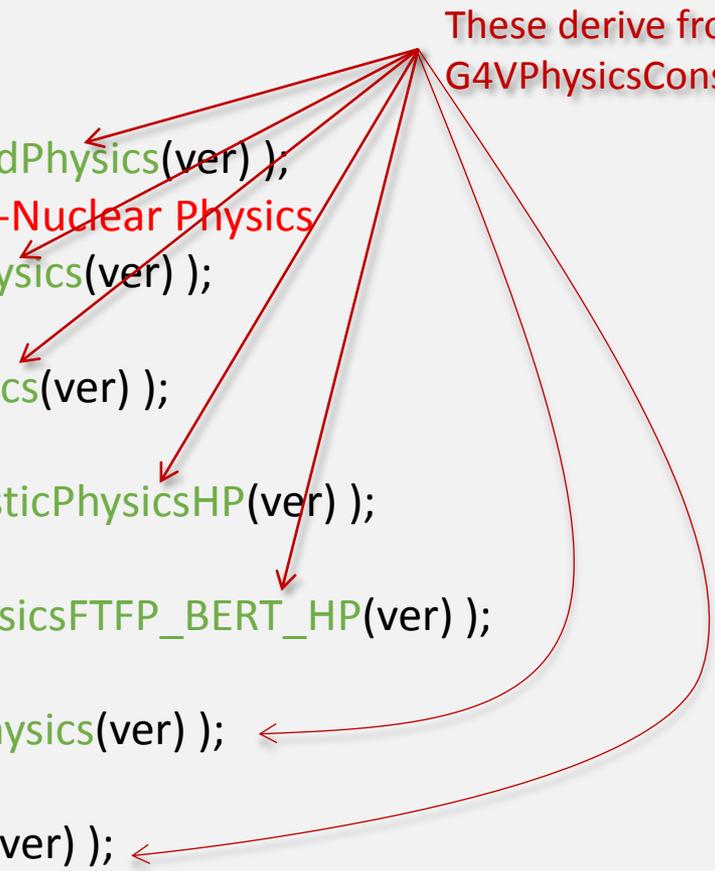› AddTransportation() automatically called for all registered particles

Example : FTFP_BERT_HP constructor

```cpp
G4DataQuestionaire it(photon, neutron);
G4cout << "<<< Geant4 Physics List simulation engine: FTFP_BERT_HP 2.0"<<G4endl;
G4cout <<G4endl;
this->defaultCutValue = 0.7*CLHEP::mm;
this->SetVerboseLevel(ver);
// EM Physics
this->RegisterPhysics( new G4EmStandardPhysics(ver) );
// Synchroton Radiation & Gamma/lepto-Nuclear Physics
this->RegisterPhysics( new G4EmExtraPhysics(ver) );
// Decays
this->RegisterPhysics( new G4DecayPhysics(ver) );
 // Hadron Elastic scattering
this->RegisterPhysics( new G4HadronElasticPhysicsHP(ver) );
 // Hadron Physics
this->RegisterPhysics( new G4HadronPhysicsFTFP_BERT_HP(ver) );
// Stopping Physics
this->RegisterPhysics( new G4StoppingPhysics(ver) );
// Ion Physics
this->RegisterPhysics( new G4IonPhysics(ver) );
```

Example : FTFP_BERT_HP constructor

```cpp
G4DataQuestionaire it(photon, neutron);
G4cout << "<<< Geant4 Physics List simulation engine: FTFP_BERT_HP 2.0"<<G4endl;
G4cout <<G4endl;
this->defaultCutValue = 0.7*CLHEP::mm;
this->SetVerboseLevel(ver);
// EM Physics
this->RegisterPhysics( new G4EmStandardPhysics(ver) );
// Synchroton Radiation & Gamma/lepto-Nuclear Physics
this->RegisterPhysics( new G4EmExtraPhysics(ver) );
// Decays
this->RegisterPhysics( new G4DecayPhysics(ver) );
 // Hadron Elastic scattering
this->RegisterPhysics( new G4HadronElasticPhysicsHP(ver) );
 // Hadron Physics
this->RegisterPhysics( new G4HadronPhysicsFTFP_BERT_HP(ver) );
// Stopping Physics
this->RegisterPhysics( new G4StoppingPhysics(ver) );
// Ion Physics
this->RegisterPhysics( new G4IonPhysics(ver) );
```

These derive from
G4VPhysicsConstructor

EM

Hadron

Optical

Example : FTFP_BERT_HP constructor

```cpp
G4DataQuestionaire it(photon, neutron);
G4cout << "<<< Geant4 Physics List simulation engine: FTFP_BERT_HP 2.0"<<G4endl;
G4cout <<G4endl;
this->defaultCutValue = 0.7*CLHEP::mm;
this->SetVerboseLevel(ver);
// EM Physics
this->RegisterPhysics( new G4EmStandardPhysics(ver) );
// Synchroton Radiation & Gamma/lepto-Nuclear Physics
this->RegisterPhysics( new G4EmExtraPhysics(ver) );
// Decays
this->RegisterPhysics( new G4DecayPhysics(ver) );
// Hadron Elastic scattering
this->RegisterPhysics( new G4HadronElasticPhysicsHP(ver) );
// Hadron Physics
this->RegisterPhysics( new G4HadronPhysicsFTFP_BERT_HP(ver) );
// Stopping Physics
this->RegisterPhysics( new G4StoppingPhysics(ver) );
// Ion Physics
this->RegisterPhysics( new G4IonPhysics(ver) );
```

Example: FTFP_BERT_HP physics lists

```cpp
void G4EmExtraPhysics::ConstructParticle()
{
  G4Gamma::Gamma();
  G4Electron::Electron();
  G4Positron::Positron();
  G4MuonPlus::MuonPlus();
  G4MuonMinus::MuonMinus();
}
```

```cpp
void G4EmExtraPhysics::ConstructProcess()
{
 ...
 ...
  if (synchOn)     BuildSynch();
  if (gammNucOn) BuildGammaNuclear();
  if (muNucOn)    BuildMuonNuclear();
}
```

```cpp
this->RegisterPhysics( new G4EmStandardPhysics(ver) );
// Synchroton Radiation & Gamma/lepto-Nuclear Physics
this->RegisterPhysics( new G4EmExtraPhysics(ver) );
// Decays
this->RegisterPhysics( new G4DecayPhysics(ver) );
 // Hadron Elastic scattering
this->RegisterPhysics( new G4HadronElasticPhysicsHP(ver) );
 // Hadron Physics
this->RegisterPhysics( new G4HadronPhysicsFTFP_BERT_HP(ver) );
// Stopping Physics
this->RegisterPhysics( new G4StoppingPhysics(ver) );
// Ion Physics
this->RegisterPhysics( new G4IonPhysics(ver) );
```

Example: FTFP_BERT_HP ... physics lists

```cpp
void G4EmExtraPhysics::ConstructParticle()
{
  G4Gamma::Gamma();
  G4Electron::Electron();
  G4Positron::Positron();
  G4MuonPlus::MuonPlus();
  G4MuonMinus::MuonMinus();
}
```

```cpp
void G4EmExtraPhysics::ConstructProcess()
{
 …
 …
  if (synchOn)      BuildSynch();
  if (gammNucOn) BuildGammaNuclear();
  if (muNucOn)    BuildMuonNuclear();
}
```

```cpp
this->RegisterPhysics( new G4EmStandardPhysics(ver) );

// Synchroton Radiation & Gamma/lepto-Nuclear Physics
this->RegisterPhysics( new G4EmExtraPhysics(ver) );
// Decays
```

```cpp
void G4EmExtraPhysics::BuildSynch()
{
…
…
  pManager = G4Electron::Electron()->GetProcessManager();
  G4SynchrotronRadiation* theElectronSynch = new G4SynchrotronRadiation();
  pManager->AddDiscreteProcess(theElectronSynch);
…
…
}
```

Physics Constructors Catalogue Overview

Defined in geant4/source/physics_lists/constructors

› **Physics constructors directories:**

- decay
  - › Decay physics, radioactive decay
- electromagnetic
  - › « standard » & low energy EM, optical, DNA
- gamma_lepto_nuclear
  - › Gamma and lepto-nuclear + synchrotron
- hadron_elastic
  - › Elastic hadronic physics, includes ions
- hadron_inelastic
  - › Inelastics hadronic physics options
- ions
  - › Inelastic hadronic physics for ions
- stopping
  - › At rest absorption physics

› **« Technical » constructors directory:**

- limiters
  - › Constructors to add special functionalities to physics lists
  - › Special cuts:
    - Add process to kill tracks by max time, min E kin, with a dedicated constructor for neutrons
  - › Biasing:
    - Options to add biasing processes or to bias physics processes
  - › Parallel world:
    - Activate simultaneous navigation in different & parallel geometries
  - › Fast simulation:
    - Activate shortcut to standard tracking for fast simulation

G4VUserPhysicsList

Mandatory methods (you must provide)
Methods provided (not to implement)
Optional methods

ConstructParticles()

Direct calls to particle construction methods:
eg: G4Electron::ElectronDefinition();
Or using particle constructors:
eg: G4BaryonConstructor

ConstructProcesses()

Direct assignement of processes to particles:
eg: ph->RegisterProcess(new
G4GammaConversion(), particle);

SetCuts()

Set cuts to electron, gamma, positron and proton.

G4VUserPhysicsList

Inherit from

G4VModularPhysicsList

RegisterPhysics(G4VPhysicsConstructor*)

Used to collect the physics constructors

ConstructParticles()

For info: simple loop on collected physics constructors, calling "ConstructParticles()" of each.

ConstructProcess()

For info: simple loop on collected physics constructors, calling "ConstructProcesses()" of each.

SetCuts()

Set cuts to electron, gamma, positron and proton.

Mandatory methods (you must provide)
Methods provided (not to implement)
Optional methods

G4VUserPhysicsList

Methods provided (not to implement)
Optional methods

Inherit from

G4VModularPhysicsList

RegisterPhysics(G4VPhysicsConstructor*)

Used to collect the physics constructors

ConstructParticles()

For info: simple loop on collected physics constructors, calling "ConstructParticles()" of each.

ConstructProcess()

For info: simple loop on collected physics constructors, calling "ConstructProcesses()" of each.

SetCuts()

Set cuts to electron, gamma, positron and proton.

G4VPhysicsConstructor

ConstructParticles()

Direct calls to particle construction methods:
eg: G4Electron::ElectronDefinition();
Or using particle constructors:
eg: G4BaryonConstructor

ConstructProcesses()

Direct assignement of processes to particles:
eg: ph->RegisterProcess(new G4GammaConversion(), particle);

EM

Hadronic

Optical

Reference Physics Lists

Pre-packaged or Reference Physics Lists

› The pre-packaged physics list are a set of physics lists based on G4VModularPhysicsList and which respond to frequent use-cases.

  – HEP, medical, shielding, etc…

› Each pre-packaged physics list includes different choices of EM and hadronic physics

  – These choices are embodied in "physics constructors"

› These physics lists can be found on the Geant4 web page at

  – https://geant4-userdoc.web.cern.ch/UsersGuides/PhysicsListGuide/html/index.html

  – And subsequent "Reference Physics Lists" link.

› Please be critical:

  – When choosing a physics list : does it cover your needs ?

  – Of course you are invited anyway to perform relevant validations.

Pre-packaged Physics Lists

› Hadronic parts:

- FTFP_BERT, FTFP_BERT_HP, FTFP_INCLXX
- FTFP_BERT_ATL, FTFP_BERT_TRV, FTFQGSP_BERT
- QGSP_BERT, QGSP_BERT_HP, QGSP_INCLXX
- QGSP_BIC, QGSP_BIC_AllHP, QGSP_BIC_HP
- QGSP_FTFP_BERT
- FTF_BIC, QGS_BIC
- QBBC
- Shielding, ShieldingLEND
- LBE
- NuBeam

Used++ in production

› EM suffix options:

- "" : Standard (in HEP measure…) EM physics
- _EMV, _EMX : fast options for high-energy physics
- _EMY, _EMZ, _LIV, _PEN : more precise options (medical & space applications)
- _GS : option using Goudsmit-Saunderson multiple scattering model
- _LE more precise MSC for e+ & e-, 5D γ conversion, Lindhard-Sorensent for ion ionisation
- _WVI more precise MSC for e+ and e-
- _SS multiple scattering models replaced by single scattering ones (CPU expensive !)

EM

Hadronic

Optical

Physics list factory

Shielding QGSP_INCLXX QGSP_BIC_HP QGSP_BERT FTFP_BERT

Physics lists and the physics list factory

› To make use of existing physics lists, you have two choices

› You can instantiate the physics list, and set it to the run manager:

```cpp
#include "FTFP_BERT.hh"
…
    G4VModularPhysicsList* physicsList = new FTFP_BERT;
    runManager->SetUserInitialization(physicsList);
```

› Or you can use the "physics list factory" utility either:

– Getting the physics list by name:

```cpp
#include "G4PhysListFactory.hh"
…
 G4PhysListFactory physListFactory;
 G4VModularPhysicsList* physicsList
    = physListFactory.GetReferencePhysList("FTFP_BERT");
runManager->SetUserInitialization(physicsList);
```

› You can (and should) check before if this physics list name exists, with:

```cpp
physListFactory.IsReferencePhysList("FTFP_BERT");
```

– Or getting the physics from the environment variable PHYSLIST:

```cpp
physicsList = physListFactory.ReferencePhysList();
```

Summary

- All the particles, physics processes and production cuts needed for an application must go into a physics list
  - And you pass this physics list to the run manager

- Two kinds of physics list classes are available for users to derive from
  - G4VUserPhysicsList – for relatively simple physics lists
  - G4VModularPhysicsList – for advanced physics lists

- Pre-packaged physics lists are provided by Geant4
  - Electromagnetic physics lists
  - Electromagnetic + hadronic physics lists
  - … and other options
  - These can be used as starting point in dedicated cases

- You must be critical in choosing the physics to use

EM

Hadronic

Optical