

GEANT4
A SIMULATION TOOLKIT

EM

Hadronic

Optical

Physics I : Physics Lists

Geant4 PHENIICS & IN2P3 Tutorial,
13 – 17 May 2019,
Orsay

Marc Verderi
LLR, Ecole polytechnique

Credits...

- › Again a long and incomplete filiation of 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

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.

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

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.

EM

Hadronic

Optical

Why Do We Need a Physics List?

- › *Physics is physics* : shouldn't Geant4 provide, as a default, a complete set of physics processes that everyone can use?
- › No, as we can only deal with models:
 - there are many different physics models and approximations
 - › very much the case for hadronic physics
 - › but also true for electromagnetic physics
 - “finite” computation speed is an issue
 - › You have to balance computation speed against physics details
 - a given application is only interested in some “physics range”:
 - › Most medical applications do not want multi-GeV physics
 - › HEP applications do not care about details of atomic shells
- › Physics lists are meant to “compose” consistent sets of physics models to respond to well defined use cases.

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

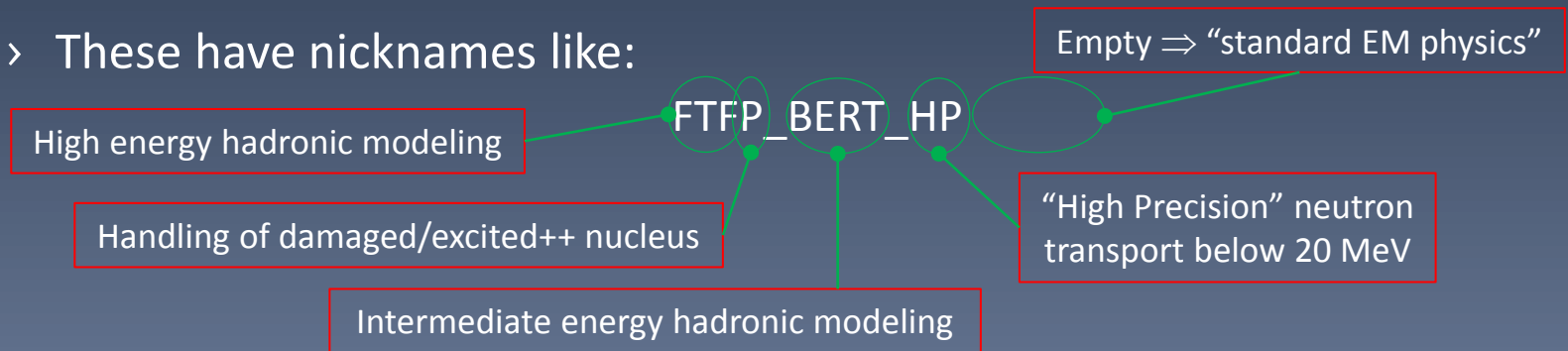
EM

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:



EM

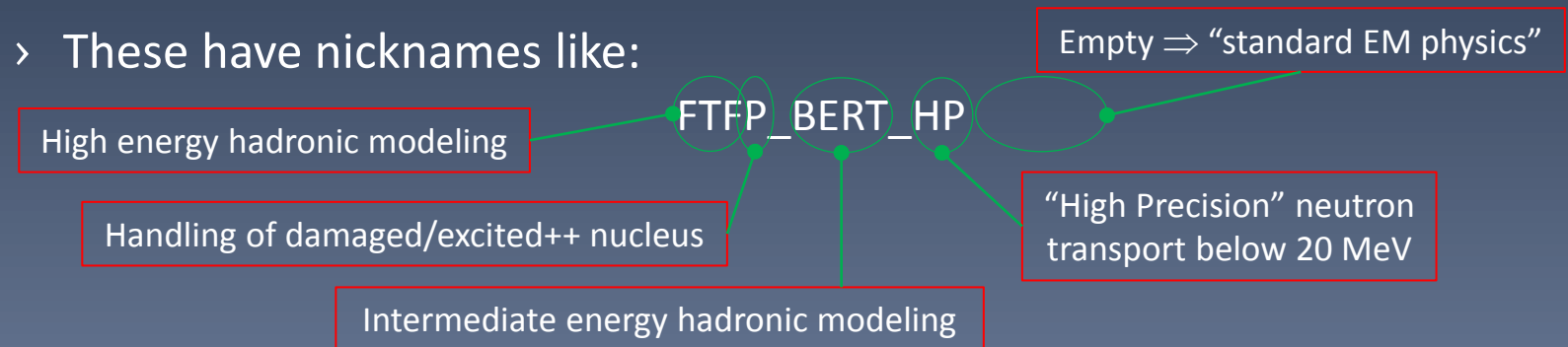
Hadronic

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:



- › They cover most of the “use cases”, and are part of the routine testing and validation of Geant4 (more on Friday, session X)
 - Reference physics lists are the most tested
 - Some of them are used in (heavy) production by experiments
 - More on this Friday, session X
- › But often users (you) want to customize their physics lists
 - And this is possible in Geant4
- › This customization is possible at various levels of granularity
- › And for this, you need to know about “physics list”.



EM

Hadronic

Optical

The G4VUserPhysicsList class

The fundamental class

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”.

EM

Hadronic

Optical

G4VUserPhysicsList : an example

- Your physics list header hence looks like:

```
#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:

```
#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

```
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

```
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:

```
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()

```
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()

```
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() : ConstructEM()

```
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.
    }
}
```



Be careful to do “new G4ProcessXXX” calls in ConstructProcess() or in methods called by it (ie, don’t do “new” calls in physics list constructor, keeping pointers : this will crash in Multi-Threading mode !)

ConstructProcess() : ConstructGeneral()

```
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()

```
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 you 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

EM

Hadronic

Optical

G4VModularPhysicsList

- › Derived from G4VUserPhysicsList
 - It is a G4VUserPhysicsList in the C++ sense
 - I.e., in header file of this class you find:
 - › `class G4VModularPhysicsList : public G4VUserPhysicsList {...};`
- › A G4VModularPhysicsList object registers “physics modules”, eg:
`someModularPhysicsList->Register(new DecayPhysics());`

EM

Hadronic

Optical

G4VModularPhysicsList

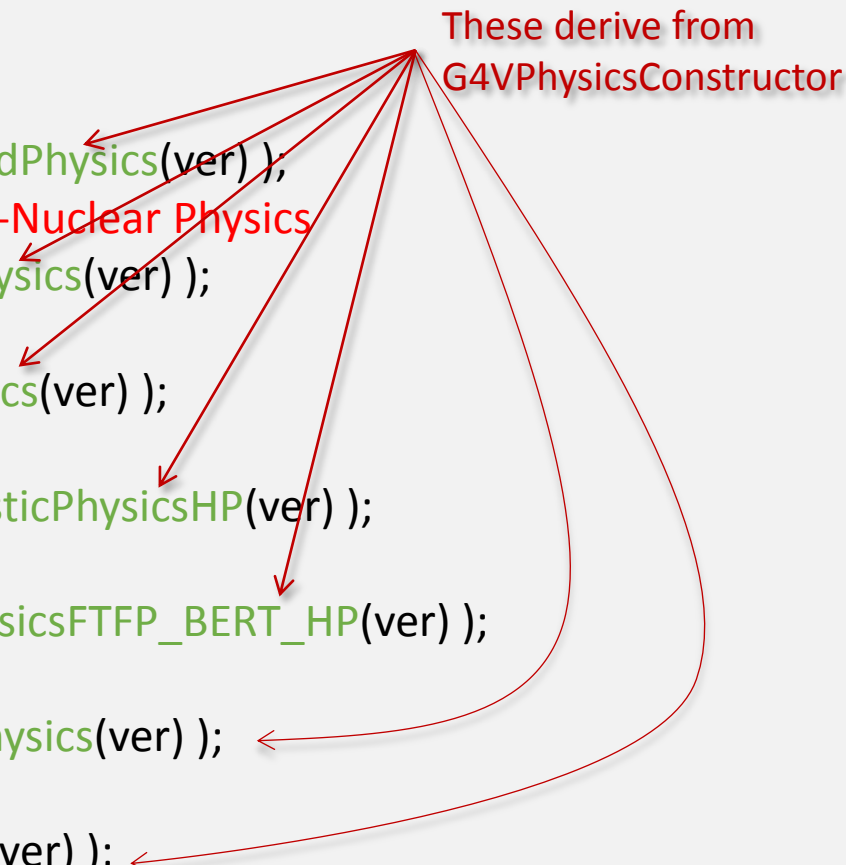
- › Derived from G4VUserPhysicsList
 - It is a G4VUserPhysicsList in the C++ sense
 - I.e., in header file of this class you find:
 - › `class G4VModularPhysicsList : public G4VUserPhysicsList {...};`
- › A G4VModularPhysicsList object registers “physics modules”, eg:
`someModularPhysicsList->Register(new 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

```
G4DataQuestionnaire 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) );
// Synchrotron 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

```
G4DataQuestionnaire 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) );
// Synchrotron 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

The diagram illustrates the inheritance of the `RegisterPhysics` method from `G4VPhysicsConstructor` to various physics list classes. Red arrows point from the text "These derive from G4VPhysicsConstructor" to the following classes: `G4EmStandardPhysics`, `G4EmExtraPhysics`, `G4DecayPhysics`, `G4HadronElasticPhysicsHP`, `G4HadronPhysicsFTFP_BERT_HP`, `G4StoppingPhysics`, and `G4IonPhysics`. The `G4HadronPhysicsFTFP_BERT_HP` class is highlighted in green in the code, indicating it is the specific implementation being used in this example.

Example : FTFP_BERT_HP constructor

```
G4DataQuestionnaire 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) );
// Synchrotron 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) );
```



```

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

```

```

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

```

```

this->RegisterPhysics( new G4EmStandardPhysics(ver) );
// Synchrotron 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) );

```



```

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

```

```

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

```

```

this->RegisterPhysics( new G4EmStandardPhysics(ver) );
// Synchrotron Radiation & Gamma/lepto-Nuclear Physics
this->RegisterPhysics( new G4EmExtraPhysics(ver) );
// Decays

```

```

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
 - › Inelastic 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

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.

Mandatory methods (you must provide)

Methods provided (not to implement)

Optional methods

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

Inherit from

G4VModularPhysicsList

RegisterPhysics(G4VPhysicsConstructor*)

Used to collect the physics constructors

ConstructParticles()

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

ConstructProcesses()

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

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 can be found on the Geant4 web page at
 - geant4.cern.ch/support/proc_mod_catalog/physics_lists/physicsLists.shtml ; and subsequent “Reference Physics Lists” link.

EM

Hadronic

Optical

Program | ED-Geant4
Reference Physics Lists
geant4.cern.ch/support/proc_mod_catalog/physics_lists/referencePL.shtml
Download | User Forum | Gallery | Contact Us
Search Geant4

Home > User Support > Process/model catalog > Physics Lists > Reference Physics Lists

Reference Physics Lists

A web page [recommending physics lists](#) according to the use case is under construction. The previous version of physics list web pages referring to 'are [still available](#).

String model based physics lists

These Physics lists apply a **string model** for the modeling of interactions of high energy hadrons, i.e. for protons, neutrons, pions and kaons above $\sim(5-25)$ GeV depending on the exact physics list. Interactions at lower energies are handled by one of the intranuclear cascade models or the precompound model. Nuclear capture of negative particles and neutrons at rest is handled using either the Chiral Invariant Phase Space (CHIPS) model or the Bertini intranuclear cascade. Hadronic inelastic interactions use:

- a tabulation of the Barashenkov pion cross sections
- the Axen-Wellsch parameterization of the proton and neutron cross sections

The physics lists are:

- QGSP and QGSP_EMV**
 QGSP is the basic physics list applying the quark gluon string model for high energy interactions of protons, neutrons, pions, and Kaons and nuclei. The high energy interaction creates an excited nucleus, which is passed to the precompound model modeling the nuclear de-excitation.
 QGSP_EMV is identical to QGSP, but parameters of electromagnetic processes tuned to yield better cpu performance with only slightly less precision.
- QGSC and QGSC_EMV**
 As QGSP except applying CHIPS modeling for the nuclear de-excitation. In comparison to thin target experiments, this improves simulation of the nuclear de-excitation part of the interaction, resulting in slightly increased production of relatively low energy secondary protons (and neutrons).
 QGSC_EMV is identical to QGSC, but parameters of electromagnetic processes tuned to yield better cpu performance with only slightly less precision.
- QGSP_EFLOW**
 This variant of QGSC uses a different algorithm setting up the excited nucleus created by the high energy interaction resulting in a good description of target fragmentation products; comparisons to thin target data well reproduce the proton production rate in the nuclear fragmentation region.
- QGSP_BERT and QGSP_BERT_EMV**
 Like QGSP, but using Geant4 Bertini cascade for primary protons, neutrons, pions and Kaons below ~ 10 GeV. In comparison to experimental data we find improved agreement to data compared to QGSP which uses the low energy parameterised (LEP) model for all particles at these energies. The Bertini model produces more secondary neutrons and protons than the LEP model, yielding a better agreement to experimental data.
 QGSP_BERT_EMV is like QGSP_BERT, but parameters of electromagnetic processes tuned to yield better cpu performance with only slightly less precision.

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 can be found on the Geant4 web page at
 - geant4.cern.ch/support/proc_mod_catalog/physics_lists/physicsLists.shtml ; and subsequent “Reference Physics Lists” link.
- **Be critical:**
 - These lists are provided as a “best guess” of the physics needed in a given use case
 - They receive quite feed-back from users helping in improving them
 - Stay critical however: you are responsible for validating the physics for your own application !
 - You may have to come to modify/adapt such a reference physics list for your dedicated application

EM

Hadronic

Optical

Pre-packaged Physics Lists (as v10.2)

› Hadronic parts:

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

› 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 new Goudsmit-Saunderson multiple scattering model

› More will be discussed on these on Friday

EM

Hadronic

Optical

Pre-packaged Physics Lists (as v10.2)

› Hadronic parts:

- FTFP_BERT, FTFP_BERT_HP, FTFP_BERT_TRV, **FTFP_BERT_ATL**
- **FTFP_INCLXX**, **FTFP_INCLXX_HP**
- FTF_BIC, LBE, QBBC
- QGSP_BERT, QGSP_BERT_HP
- QGSP_BIC, QGSP_BIC_HP, **QGSP_BIC_AIHHP**
- QGSP_FTFP_BERT
- **QGSP_INCLXX**, **QGSP_INCLXX_HP**
- QGS_BIC
- Shielding, **ShieldingLEND**, **ShieldingM**
- **NuBeam**

Since :

- 10.0

- 10.1

- 10.2

› 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 new Goudsmit-Saunderson multiple scattering model

› More will be discussed on these on Friday

Very promising !

Pre-packaged Physics Lists (as v10.2)

> Hadronic parts:

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

Since :

- 10.0

- 10.1

- 10.2

- Used++ in production

- "Experimental"

> 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 new Goudsmit-Saunderson multiple scattering model

> More will be discussed on these on Friday

Very promising !

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:

```
#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:

```
#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:

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

- Or getting the physics from the environment variable PHYSLIST:

```
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