What is a Physics List

Setting up the physics environment:

Particles and the associated physics processes

An object of a C++ class, which is responsible of defining following:

Particles used in simulation

Physics process associated with each particle

Is it really required ??

Yes, as it is one the 3 mandatory object that needs to be registered with RunManager

Eventually its just an interface to define physics of your application.

User is supposed to have a good idea of the physics behind the application

Removal of particles and process may lead to incomplete or unpredictable simulation results

Why not all the physics is included by default by Geant4

Will that be a good idea : Yes / No ? NO

There are different model that defines the same interaction.
Some are approximation and some are extremely precise.
Effect on computation time

You actually DON'T need all the particles

Eg. : Study of energy deposition by gamma radiation in NaI

One may not need optical photon and the associated process,
Unless you need to do its photon yield study
or PMT response study.

For these reasons Geant4 does not following integral physics approach rather it follows application based physics approach.

Application based physics approach

Provides :
Independent Particles to be used
Independent physics components : Physics process

These components (processes) may be select in the custom physics list defined by user.

One important process that should always be there : Transportation

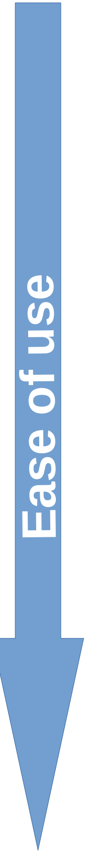
This must be assigned to all the **stable particles**

Depending upon the requirement one needs to chose different components.

Results in efficient simulation run.

Sometimes you may afford less accurate calculation, so you may get faster model for a particular interaction

Creating a Physics List



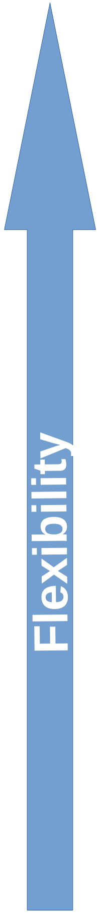
There are three ways

UserPhysicsList : Create from scratch using components (processes) and particles

available in Geant4

ModularPhysicsList : Again going to use existing components and particles but it provides an easy to use interface.

Prepacked (Reference) Physics List : Use the PhysicsList already existing in Geant4 (A good start point)



UserPhysicsList : Inheritance mechanism of C++ : G4VUserPhysicsList

Utilizes the Inheritance mechanism of C++

Base class : G4**V**UserPhysicsList

The most basic interface

User needs to specify all the particles that may be used or generated during the lifetime of a run.

For each of these particles specify all the associated process

Transportation needs to be attached to all the stable particles.

Not suitable for less experience users

But provides great flexibility.

Hence recommended for advance users.

UserPhysicsList : Interface to define Physics List

G4VUserPhysicsList : Defines the interface for Geant4 Physics List.

Recipe to implement you physics list

Inherit the G4VUserPhysicsList in your physics list

Implement the 2 mandatory functions

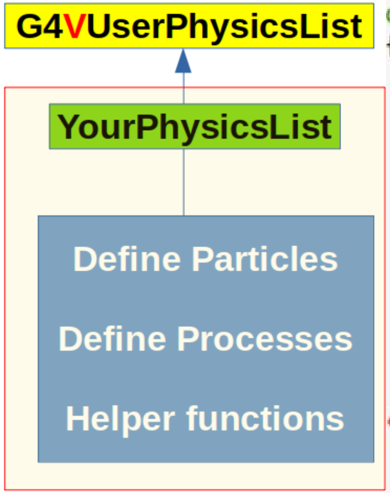
(Pure virtual functions in G4VPhysicsList)

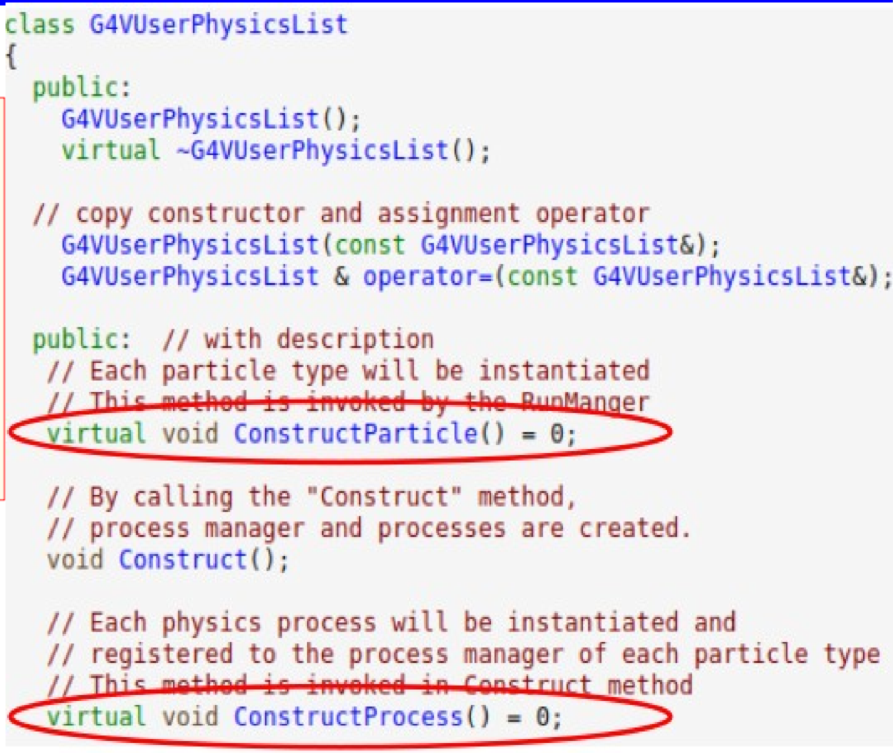
ConstructParticle()

responsible for creating all the particles required during simulation

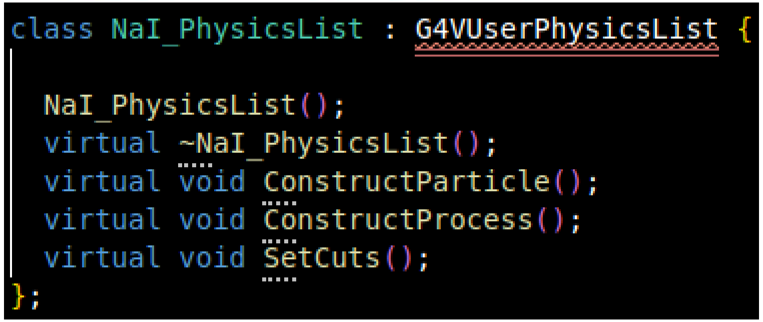
ConstructProcess()

responsible for assigning the required processes associated with each particle type





```
class G4VUserPhysicsList
{
  public:
    G4VUserPhysicsList();
    virtual ~G4VUserPhysicsList();

  // copy constructor and assignment operator
    G4VUserPhysicsList(const G4VUserPhysicsList&);
    G4VUserPhysicsList & operator=(const G4VUserPhysicsList&);

  public:  // with description
   // Each particle type will be instantiated
   // This method is invoked by the RunManger
   virtual void ConstructParticle() = 0;

   // By calling the "Construct" method,
   // process manager and processes are created.
   void Construct();

   // Each physics process will be instantiated and
   // registered to the process manager of each particle type
   // This method is invoked in Construct method
   virtual void ConstructProcess() = 0;
```

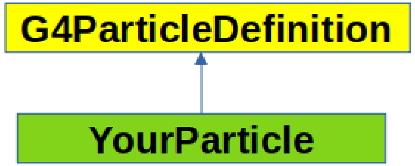


```
class NaI_PhysicsList : G4VUserPhysicsList {

  NaI_PhysicsList();
  virtual ~NaI_PhysicsList();
  virtual void ConstructParticle();
  virtual void ConstructProcess();
  virtual void SetCuts();
};
```

UserDefined Physics List : Particles

In Geant4, everything is a C++ class.

Each of the particle is defined by its class



G4ParticleDefinition

YourParticle

A Particle can be created using following ways
1) Using ParticleTable : G4ParticleTable::GetParticleTable()
2) Using ParticleClass : G4<ParticleName>::Definition()

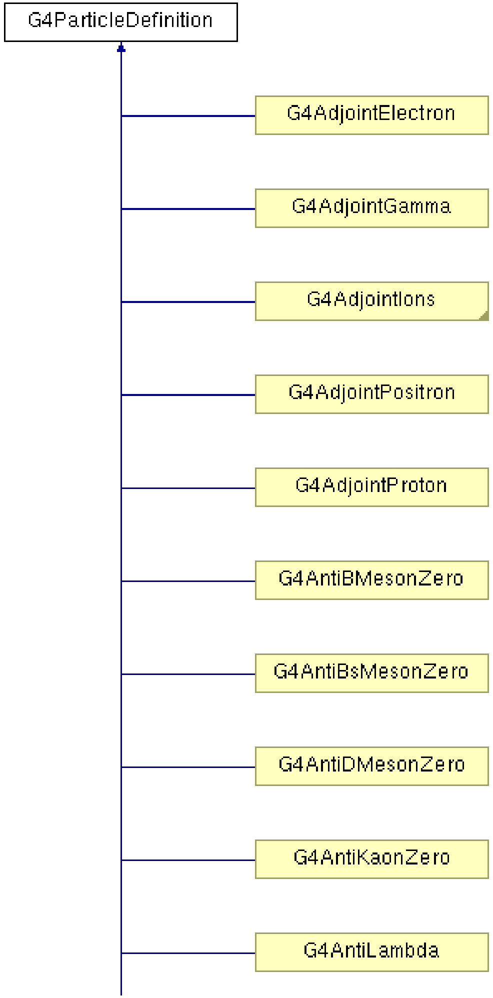
Eg. Getting a muon- and gamma

G4ParticleDefinition *muMinus = G4ParticleTable::GetParticleTable()->FindParticle("mu-")

G4ParticleDefinition *muMinus = G4MuonMinus::Definition();

G4ParticleDefinition *gamma = G4ParticleTable::GetParticleTable()->FindParticle("gamma")

G4ParticleDefinition *gamma = G4Gamma::Definition();



G4ParticleDefinition

G4AdjointElectron

G4AdjointGamma

G4AdjointIons

G4AdjointPositron

G4AdjointProton

G4AntiBMesonZero

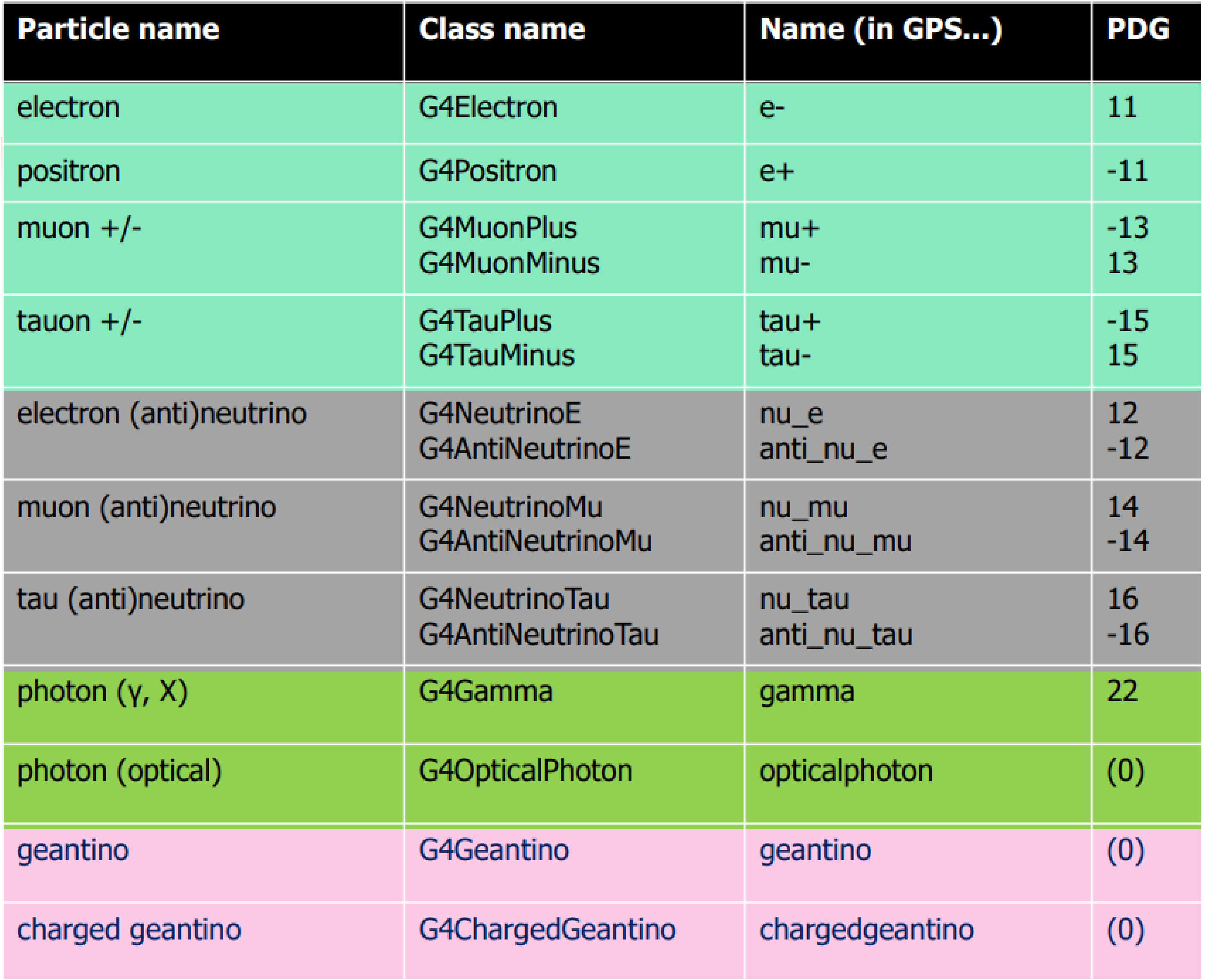
G4AntiBsMesonZero

G4AntiDMesonZero

G4AntiKaonZero

G4AntiLambda

UserDefined Physics List : Particles



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

UserDefined Physics List : Processes

Each process assiociated with each particle is implemented in a separate class.

All these classes are placed at **geant4-v11.3.0/source/processes/**

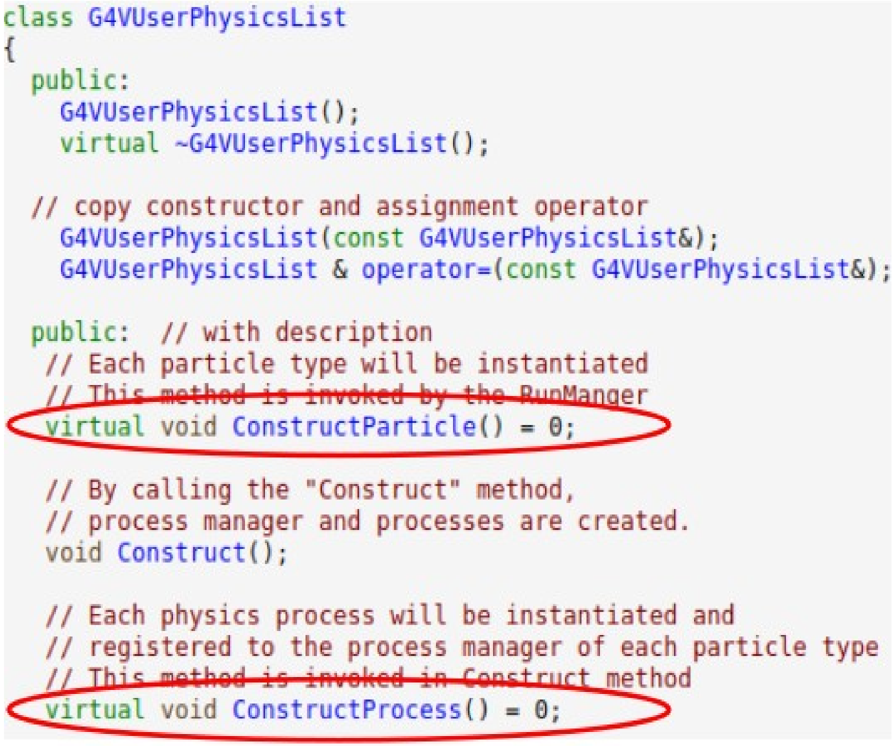
Eg. : Electromagnetic Processes related to gamma can be obtained by creating object of following classes

G4PhotoElectricEffect
G4ComptonScattering
G4GammaConversion
G4RayleighScattering

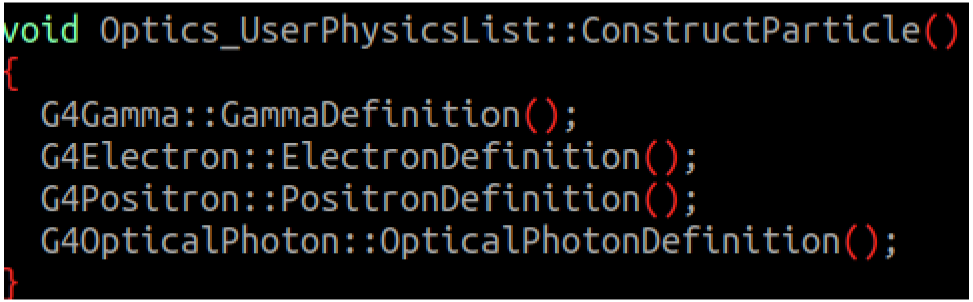
Whatever process you want to create is available but you need to know its class name

UserDefined Physics List : Final Step

Once you have identified particles and process used in the simulation, populated them at the proper place.



```
class G4VUserPhysicsList
{
  public:
    G4VUserPhysicsList();
    virtual ~G4VUserPhysicsList();

  // copy constructor and assignment operator
    G4VUserPhysicsList(const G4VUserPhysicsList&);
    G4VUserPhysicsList & operator=(const G4VUserPhysicsList&);

  public:  // with description
   // Each particle type will be instantiated
   // This method is invoked by the RunManger
   virtual void ConstructParticle() = 0;

   // By calling the "Construct" method,
   // process manager and processes are created.
   void Construct();

   // Each physics process will be instantiated and
   // registered to the process manager of each particle type
   // This method is invoked in Construct method
   virtual void ConstructProcess() = 0;
```



```
void Optics_UserPhysicsList::ConstructParticle()
{
  G4Gamma::GammaDefinition();
  G4Electron::ElectronDefinition();
  G4Positron::PositronDefinition();
  G4OpticalPhoton::OpticalPhotonDefinition();
}
```

UserDefined Physics List : Process Registration

Particles are created.

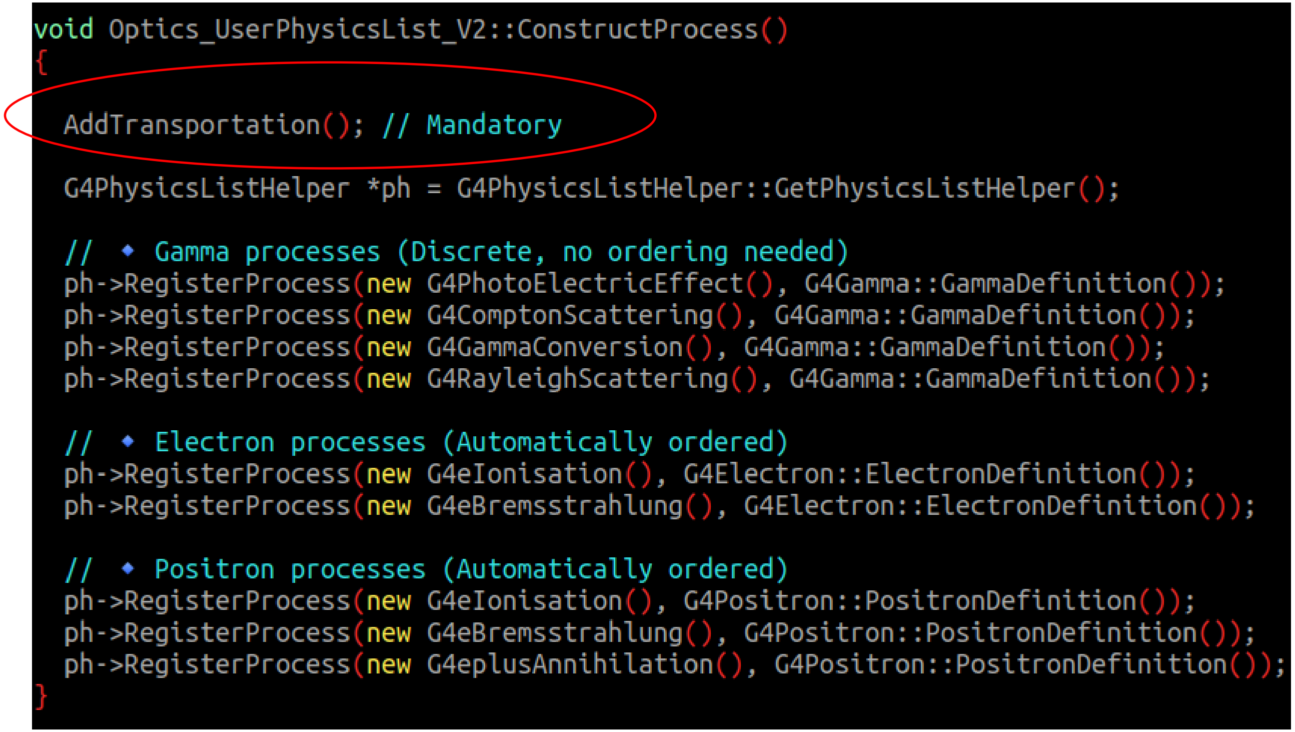
Processes are created.

Finally they need to be linked.

Multiple ways of linking.

The cleanest is by using **PhysicsListHelper** class.

The linking is done by coupling the processes and particle using the RegisterProcess function of **PhysicsListHelper** Class.



```
void Optics_UserPhysicsList_V2::ConstructProcess()
{
  AddTransportation(); // Mandatory

  G4PhysicsListHelper *ph = G4PhysicsListHelper::GetPhysicsListHelper();

  // ◆ Gamma processes (Discrete, no ordering needed)
  ph->RegisterProcess(new G4PhotoElectricEffect(), G4Gamma::GammaDefinition());
  ph->RegisterProcess(new G4ComptonScattering(), G4Gamma::GammaDefinition());
  ph->RegisterProcess(new G4GammaConversion(), G4Gamma::GammaDefinition());
  ph->RegisterProcess(new G4RayleighScattering(), G4Gamma::GammaDefinition());

  // ◆ Electron processes (Automatically ordered)
  ph->RegisterProcess(new G4eIonisation(), G4Electron::ElectronDefinition());
  ph->RegisterProcess(new G4eBremsstrahlung(), G4Electron::ElectronDefinition());

  // ◆ Positron processes (Automatically ordered)
  ph->RegisterProcess(new G4eIonisation(), G4Positron::PositronDefinition());
  ph->RegisterProcess(new G4eBremsstrahlung(), G4Positron::PositronDefinition());
  ph->RegisterProcess(new G4eplusAnnihilation(), G4Positron::PositronDefinition());
}
```

**** Congratulation you had succesfully created a UserDefined Physics list ****

Summary UserDefined Physics List

Determine all the Physics of the problem at hand.

Find all the particles associated with that Physics

Find all the processes associated with each particles for that Physics

Create the required objects of particles and Physics processes.

Register them with PhysicsListHelpers

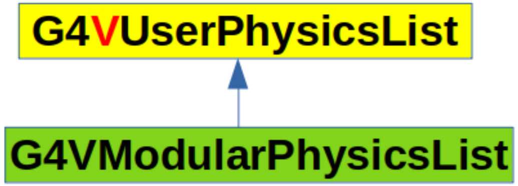
Needs detailed information

Not very trivial to implement

But provide complete control

Modular Physics List is an alternative that provides higher level implementation.

ModularPhysicsList : Inheritance mechanism of C++ : G4VModularPhysicsList



G4VUserPhysicsList

G4VModularPhysicsList

Utilizes the Inheritance mechanism of C++

Base class : G4VModularPhysicsList

G4VModularPhysicsList Class is inherited from G4VUserPhysics List

Different Processes related to particular physics are already attached to respective particles

Electromagnetic Physics: Attached all the particles which undergoes electromagnetic interactions to the corresponding process.

User don’t have to worry for individual particles and the associated processes.

Provide more convenient way to create user define physics lists.

Automatically attached Transportation to all the constructed particles.

User may add more helper functions for debugging or to get additional information about the physics process and the associated particles.

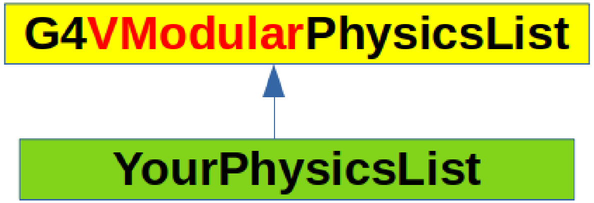
Defining a user defined physics list using G4VModularPhysicsList

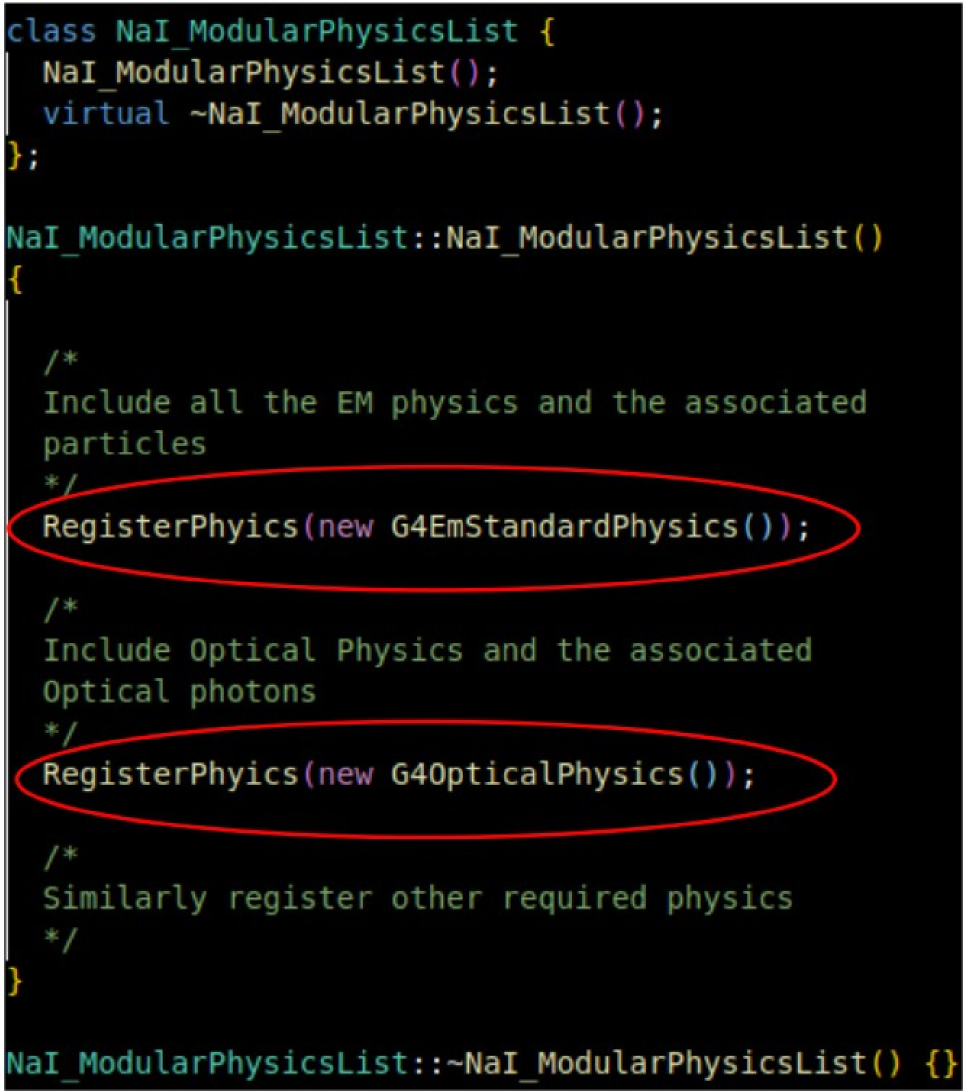
G4VModularPhysicsList : Defines the higher level interface for Geant4 Physics List.

Recipe to implement you physics list

Inherit the G4VModularPhysicsList in your physics list

Register the desired physics in the constructor of your physics list.





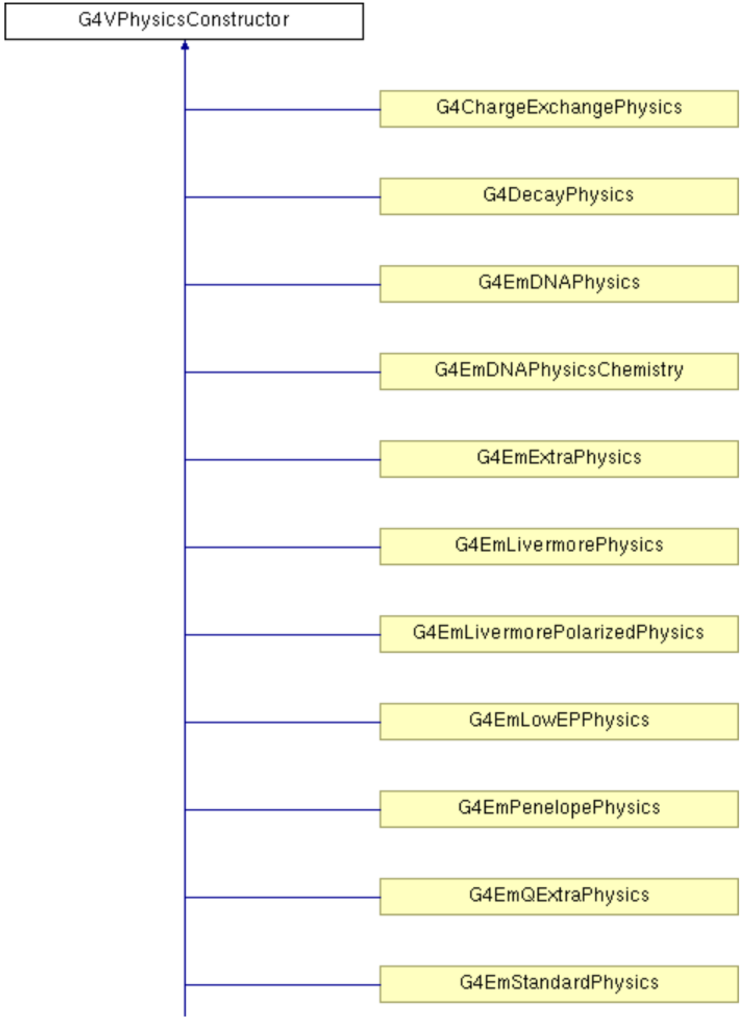
```
class NaI_ModularPhysicsList {
  NaI_ModularPhysicsList();
  virtual ~NaI_ModularPhysicsList();
};

NaI_ModularPhysicsList::NaI_ModularPhysicsList()
{

  /*
  Include all the EM physics and the associated
  particles
  */
  RegisterPhyics(new G4EmStandardPhysics());

  /*
  Include Optical Physics and the associated
  Optical photons
  */
  RegisterPhyics(new G4OpticalPhysics());

  /*
  Similarly register other required physics
  */
}

NaI_ModularPhysicsList::~NaI_ModularPhysicsList() {}
```

Existing Physics constructor required use ModularPhysicsList interface

What all Physics constructor are already defined.

https://apc.u-paris.fr/~franco/g4doxy/html/classG4VPhysicsConstructor.html

- **Some "standard" EM physics constructors:**
  - G4EmStandardPhysics – default
  - G4EmStandardPhysics_option1 - for HEP, fast but not precise settings
  - G4EmStandardPhysics_option2 - for HEP, experimental
  - G4EmStandardPhysics_option3 - for medical and space science applications
  - G4EmStandardPhysics_option4 - most accurate EM models and settings
- **Some hadronic physics constructors**
  - G4HadronElasticPhysics – default for hadron nuclear elastic for all hadrons
  - G4HadronElasticPhysicsHP – as above, but use HP for neutrons below 20 MeV
  - G4HadronPhysicsFTFP_BERT – hadron nucleus inelastic physics for all hadrons
  - G4IonPhysics – interactions of Ions
- **The complete list of constructors can be found in your toolkit:**
  - geant4/source/physics_lists/constructors/...
- **More information at:**
  - README files in geant4/source/physics_lists/constructors/..../README
  - http://cern.ch/geant4-userdoc/UsersGuides/PhysicsListGuide/html/index.html



G4VPhysicsConstructor

G4ChargeExchangePhysics

G4DecayPhysics

G4EmDNAPhysics

G4EmDNAPhysicsChemistry

G4EmExtraPhysics

G4EmLivermorePhysics

G4EmLivermorePolarizedPhysics

G4EmLowEPPhysics

G4EmPenelopePhysics

G4EmQExtraPhysics

G4EmStandardPhysics

Summary Modular Physics List

Need to create your own class inherited from G4VModularPhysicsList class

Most of the work that was manually done in UserDefined physics list is already implemented in terms of Physics constructor.

No need to worry about each particle and associated process.

Just Need to know the correct physics constructor name.

In the constructor of you class just need to register the Physics constructor, rest everything will be taken care by Geant4 itself.

Now instantiate the object of your class in your main program and register that with RunManager.

Pre-Packaged (Reference) PhysicsList : Use them directly (Inheritance NOT required)

Pre-packaged physics list provides several advantages

Ready to use physics list : Just create an object of the existing pre-packaged class and inform the RunManager

Avoids complexity of manually selecting physics processes.

Extremely easy to use, even for beginners.

Created and maintained by experts, chances of error are extremely less.

Warning : Responsibility lies with the user. One has see carefully chose the physics list based on the his application.

Also the user is responsible to validate the chosen physics list for his application.

Examples: FTFP_BERT, QGSP_BERT, QGSP_FTFP_BERT, etc.

Naming Conventions of Reference Physics Lists

QGSP_BERT: Quark-Gluon String Model + Bertini Cascade.

FTFP_BERT: Fritiof String Model + Bertini Cascade.

QGSP_FTFP_BERT: Hybrid model using QGS, FTF, and BERT.

QGSP_BIC: Binary Cascade instead of Bertini.

Shielding: Optimized for radiation shielding applications.

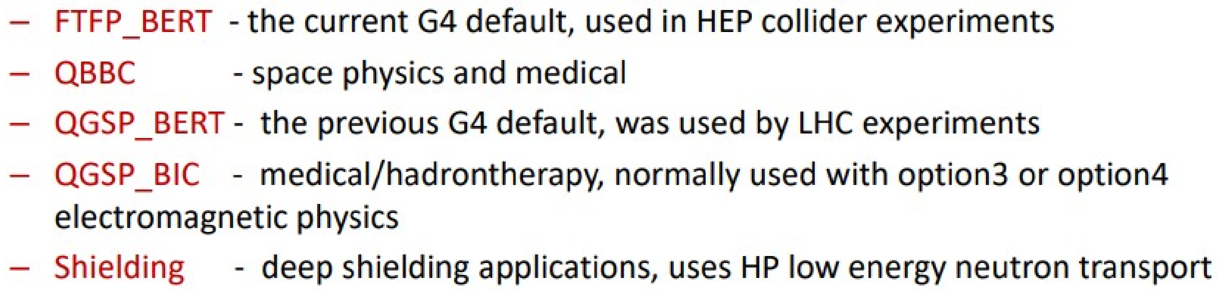
Pre-Packaged PhysicsList cont..

All the pre-packaged physics list are very well documented, and is used by large group of people.

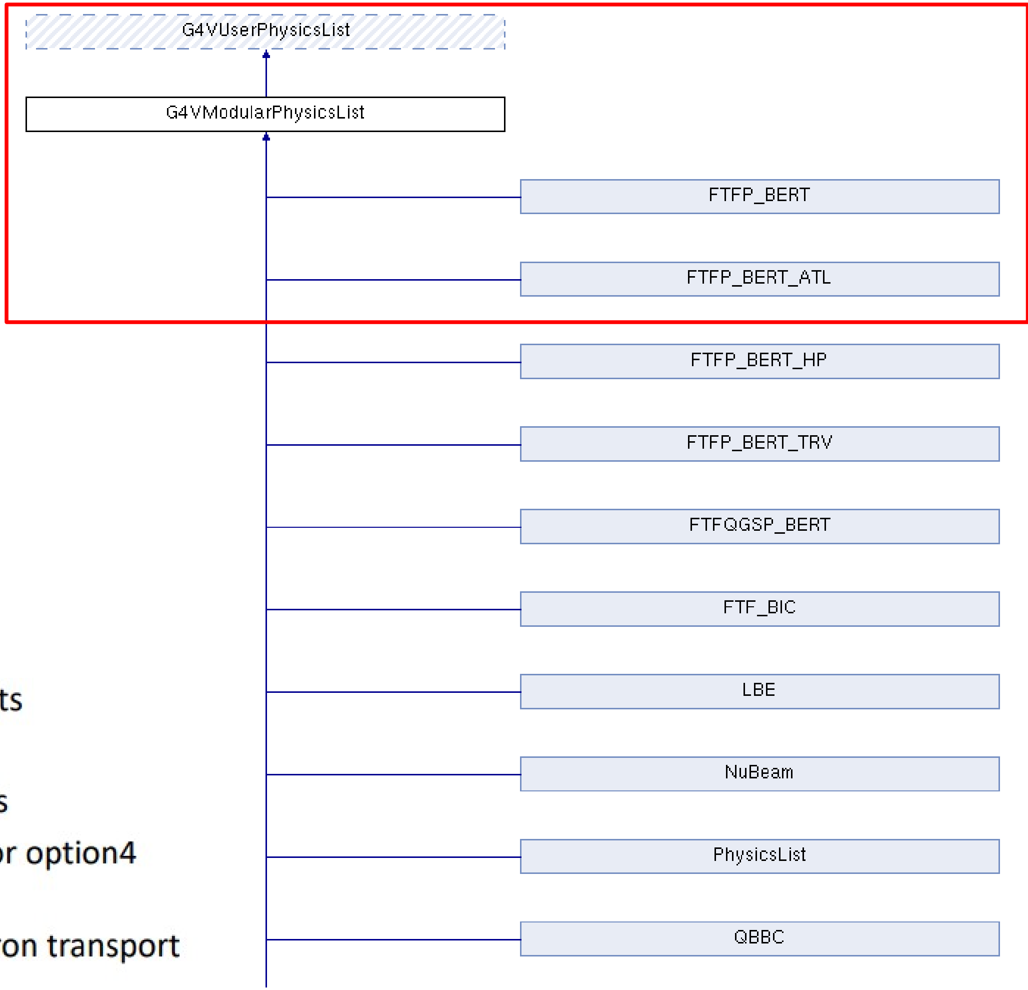
https://apc.u-paris.fr/~franco/g4doxy4.11/html/classG4VModularPhysicsList.html

They need to be used in the same way as user defined ModularPhysicsList.

Details : https://geant4.web.cern.ch/documentation/dev/plg_html/PhysicsListGuide/reference_PL/index.html



- FTFP_BERT - the current G4 default, used in HEP collider experiments
- QBBC - space physics and medical
- QGSP_BERT - the previous G4 default, was used by LHC experiments
- QGSP_BIC - medical/hadrontherapy, normally used with option3 or option4 electromagnetic physics
- Shielding - deep shielding applications, uses HP low energy neutron transport



Pros and Cons of Reference Physics Lists

Pros:

- Prevalidated and well-tested
- Optimized for different applications
- Easy to implement
- Regularly updated with new Geant4 releases

Cons

- Not always optimized for every experiment.
- May require tuning for specific applications.
- Custom physics lists may be necessary for specialized studies.

How to Use a Reference Physics List in Geant4?

- Include the physics list in Main program:

  ```
  #include "FTFP_BERT.hh"
  ```

- Instantiate the physics list in G4RunManager:

  ```
  runManager->SetUserInitialization(new FTFP_BERT);
  ```

- No need to manually define particle interactions.

Summary of Reference Physics List

- Reference physics lists simplify simulation setup.
- Choose based on your application needs.
- Geant4 continuously improves these lists.
- Good starting point for beginners.
- Can be extended depending upon user requirements.

Electromagnetic Models in Reference Lists

- Geant4 includes multiple EM models optimized for different energy ranges.
- Common EM models used in reference lists:
- - G4EmStandardPhysics: Default for general applications.
- - G4EmStandardPhysics_option4: Provides high precision for medical applications.
- - G4EmLivermorePhysics: Optimized for low-energy electromagnetic interactions.
- - G4EmPenelopePhysics: Used for very low-energy simulations.

Extended Reference Physics Lists

- Some reference lists extend standard models for specific applications:
- - Shielding: Optimized for radiation protection studies.
- - QGSP_BIC_HP: High-precision neutron transport.
- - LIV: Uses Livermore EM physics for precise low-energy simulations.
- - PENELOPE: Uses Penelope EM physics for very low-energy interactions.

How to Customize Reference Physics Lists?

- Geant4 allows modifying reference physics lists for specific needs.
- Steps to customize:
- 1. Start with a base physics list (e.g., FTFP_BERT).
- 2. Add or replace components (e.g., use Livermore EM physics instead of standard EM).
- 3. Register additional processes (e.g., optical photon physics).

Customize Reference Physics Lists

Example: Replace the Standard Electromagnetic model with Livermore model

```
G4VModularPhysicsList* physicsList = new FTFP_BERT;

physicsList->ReplacePhysics(new G4EmLivermorePhysics());

runManager->SetUserInitialization(physicsList);
```

Conclusion

Physics list is one of the mandatory class used in Geant4 Simulation

To do a logical simulation all the required particles and process needs to be registered within the physics list.

Various interface exists to define you physics list

G4VUserPhysics : Provides the maximum level of flexibility but needs expertisze, hence \ can safely be used for simple experiment setup

G4VModularPhysicsList : Provides a convenient way to define you physics list, and is generally used for more complex physics problem.

Reference (pre-packaged) physics lists makes the life easier and is a good starting point for beginners.

Physics lists must be selected with extreme care to get the meaningful results from the simulations.

Assignment

You have to write the simulation code to simulate the interaction of 662 keV gammas with NaI crystal.

Specification of NaI Crystal : Cylinder of 2 inch diameter, 2 inch height

Energy resolution : 40 keV

Particle source : gammas of 662 keV

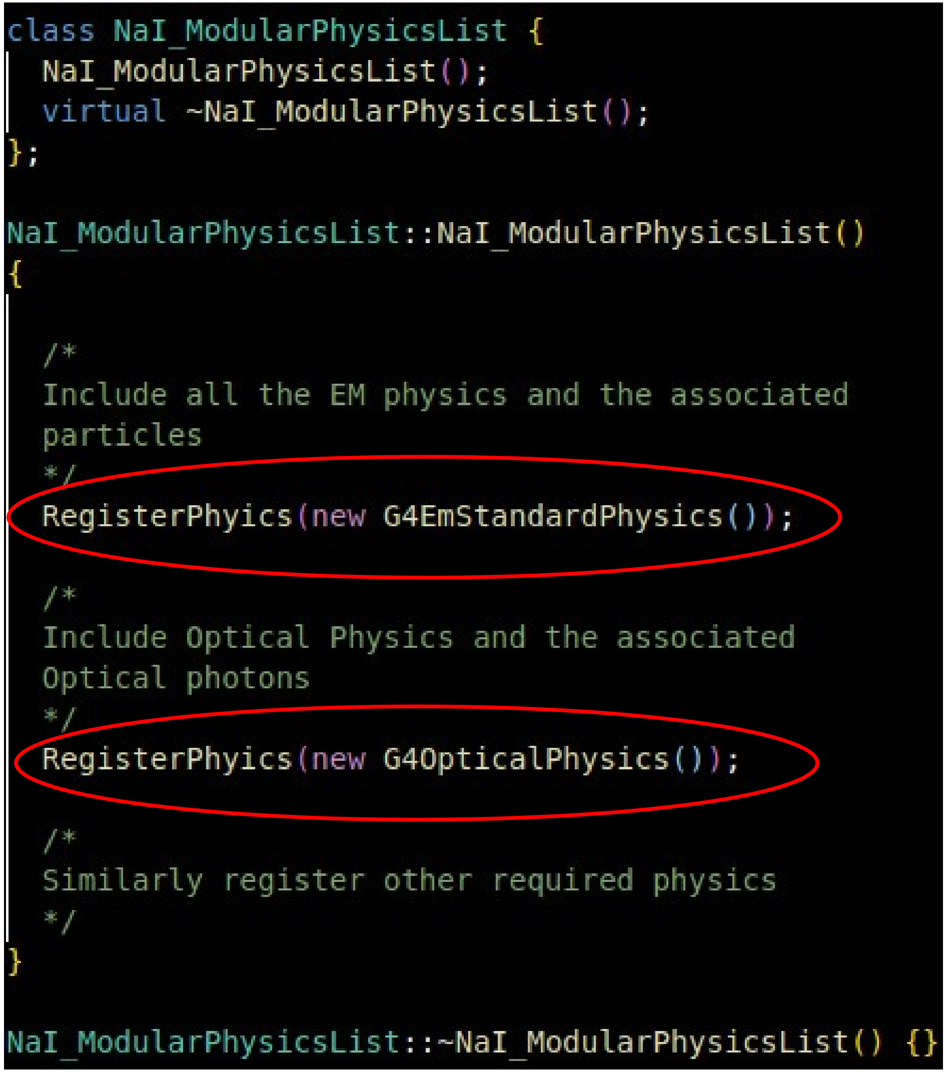
Physics List creation : (a) Pre-packaged
(b) ModularPhysicsList
(c) User Physics List.

Final outcome : Should get a gaussian peak at 662 keV.
spectras from all the three types of physics lists should match

Thank you and All the best for the assignment

Heading

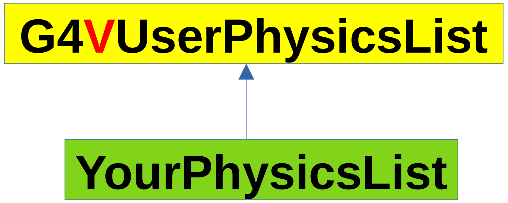
Content



```
class NaI_ModularPhysicsList {
  NaI_ModularPhysicsList();
  virtual ~NaI_ModularPhysicsList();
};

NaI_ModularPhysicsList::NaI_ModularPhysicsList()
{

  /*
  Include all the EM physics and the associated
  particles
  */
  RegisterPhyics(new G4EmStandardPhysics());

  /*
  Include Optical Physics and the associated
  Optical photons
  */
  RegisterPhyics(new G4OpticalPhysics());

  /*
  Similarly register other required physics
  */
}

NaI_ModularPhysicsList::~NaI_ModularPhysicsList() {}
```

Heading

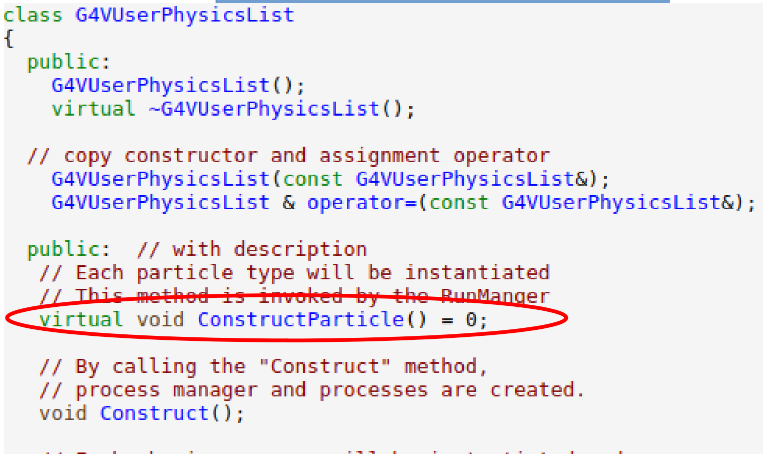
Content



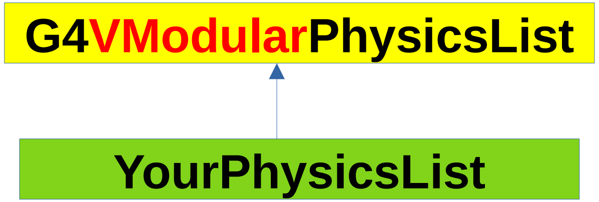
Define Particles

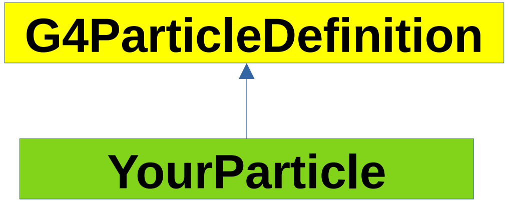
Define Processes

Helper functions



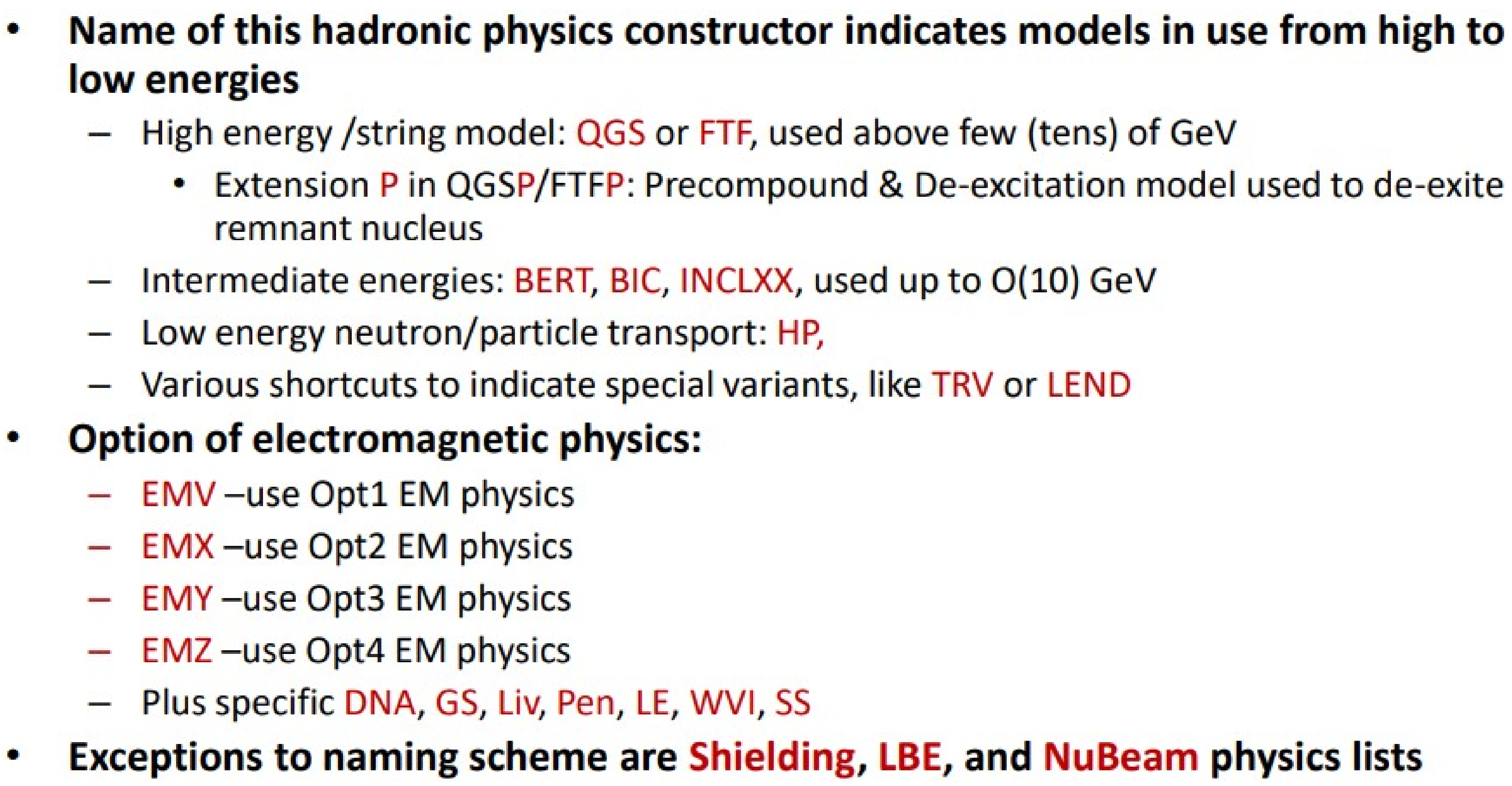
```
class G4VUserPhysicsList
{
  public:
    G4VUserPhysicsList();
    virtual ~G4VUserPhysicsList();

  // copy constructor and assignment operator
    G4VUserPhysicsList(const G4VUserPhysicsList&);
    G4VUserPhysicsList & operator=(const G4VUserPhysicsList&);

  public:  // with description
   // Each particle type will be instantiated
   // This method is invoked by the RunManger
   virtual void ConstructParticle() = 0;

   // By calling the "Construct" method,
   // process manager and processes are created.
   void Construct();
```





Naming conventions of various Pre-packaged Production Physics lists

Most of the physics lists follows name of Physics Constructor



- **Name of this hadronic physics constructor indicates models in use from high to low energies**
  - High energy /string model: QGS or FTF, used above few (tens) of GeV
    - Extension P in QGSP/FTFP: Precompound & De-excitation model used to de-exite remnant nucleus
  - Intermediate energies: BERT, BIC, INCLXX, used up to O(10) GeV
  - Low energy neutron/particle transport: HP,
  - Various shortcuts to indicate special variants, like TRV or LEND
- **Option of electromagnetic physics:**
  - EMV –use Opt1 EM physics
  - EMX –use Opt2 EM physics
  - EMY –use Opt3 EM physics
  - EMZ –use Opt4 EM physics
  - Plus specific DNA, GS, Liv, Pen, LE, WVI, SS
- **Exceptions to naming scheme are Shielding, LBE, and NuBeam physics lists**

Physics List in Geant4

What is a Physics List (Is it really required ??)

Various Physics List Interface (usecase depends on the expertise)

- UserPhysicsList (Level1 : Write from scratch)
- ModularPhysicsList (Level 2 : More convenient to implement)
- Prepacked Physics List (Level 3 : Provided by Geant4 Toolkit)
  - Reference Physics List
  - How to extend them

Heading

Content