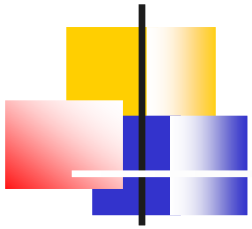


Physics I: Physics Lists

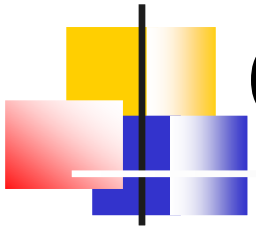
Paris Geant4 Tutorial

4 June 2007

Marc Verderi

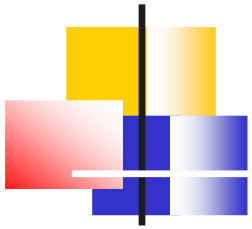
Laboratoire Leprince-Ringuet

(Heavily copied from D. Wright)

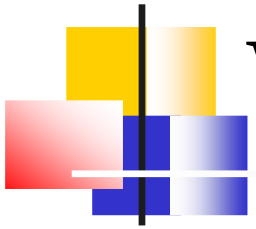


Outline

- Introduction
  - What is a physics list and why do we need one?
- The **`G4VUserPhysicsList`** class
  - What you need to begin
- Modular physics lists
  - A more sophisticated way to go
- Reference physics lists
  - A word about…



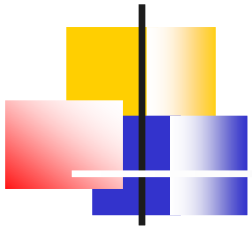
Introduction



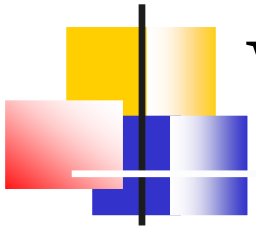
What is a Physics List?

- A class which implements the configuration of the physics (modelling) of your application:
  - Declare all the particles used
  - The physics processes they undergo
  - The production thresholds (some of these processes need one)
- This physics environment is built by the user in a flexible way:
  - picking up the particles he wants
  - picking up the physics to assign to each particle
- User must have a good understanding of the physics required
  - omission of particles or physics could cause errors or poor simulation

Why Do We Need a Physics List?



- "Physics is physics – shouldn't Geant4 provide, as a default, a complete set of physics that everyone can use?"
- No:
  - Softwares can only capture Physics through a modelling
    - No unique Physics modelling
      - Very much the case for hadronic physics
      - But also the electromagnetic physics
      - Existing models still evolve and new models are created
    - Some modellings are more suited to some energy ranges
      - Medical applications not interested in multi-GeV physics in general
      - HEP experiments not interested in effects due to atomic shell structure for example
  - computation speed is an issue
    - a user may want a less-detailed, but faster approximation

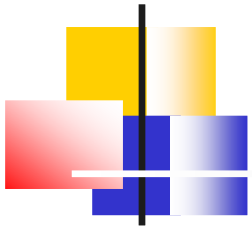


Why Do We Need a Physics List?

- For this reason Geant4 takes an atomistic, rather than an integral approach to physics
  - provide many physics components (processes) which are de-coupled from one another
  - user selects these components in custom-designed physics lists
- ***Exceptions :***
  - a few electromagnetic processes must be used together
  - future processes involving interference of electromagnetic and strong interactions may require coupling as well

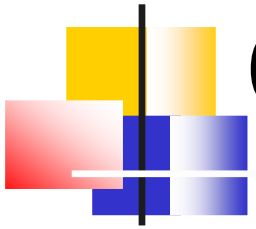
Physics Processes Provided by Geant4

- EM physics
  - "standard" processes valid from ~ 1 keV → ~ PeV
  - "low-energy" valid from 250 eV → ~ PeV
  - optical photons
- Weak physics
  - decay of subatomic particles
  - radioactive decay of nuclei
- Hadronic physics
  - pure hadronic processes valid from 0 → ~100 TeV
  - Muon and gamma nuclear valid from 10 MeV → ~TeV
- Parameterized or "fast simulation" physics
  - "glfash" model for EM showers



The **`G4VUserPhysicsList`** class

- The class which holds the physics configuration of your application:
  - Particles
  - Processes
  - Cuts (ie production thresholds)



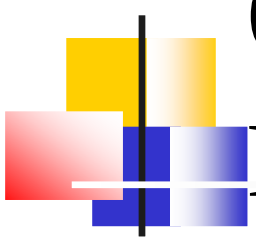
G4VUserPhysicsList

- All physics lists must derive from this class
  - and then be registered to the run manager
- For example:

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

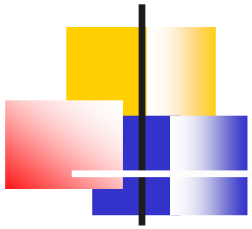
Base class methods overridden

- User must implement the methods **`ConstructParticle()`**, **`ConstructProcess()`** and **`SetCuts()`**.



`G4VUserPhysicsList`: Required Methods

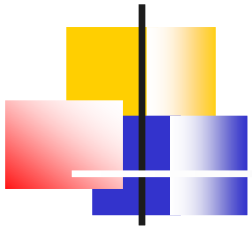
- **`ConstructParticle()`:**
  - choose the particles you need in your simulation, define all of them here
- **`ConstructProcess()`:**
  - for each particle, assign all the physics processes relevant to your simulation
    - What's a process ?
      - a class that defines how a particle should interact with matter, or decays
        - it's where the physics is!
      - more on this later
- **`SetCuts()`:**
  - set the range cuts for secondary production
    - What's a range cut ?
      - a threshold on particle production
        - Particle unable to travel at least the range cut value are not produced
      - more on this later



ConstructParticle() [1]

- Basic construction method:
  - By manually invoking the particle definition methods

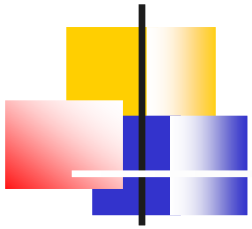
```
#include “G4Electron.hh”
#include “G4Proton.hh”
…
void MyPhysicsList::ConstructParticle()
    {
        G4Electron::ElectronDefinition();
        G4Proton::ProtonDefinition();
        G4Neutron::NeutronDefinition();
        G4Gamma::GammaDefinition();
        …
    }
```



ConstructParticle() [2]

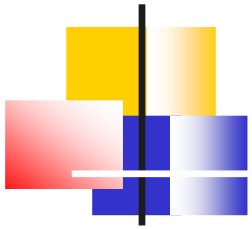
- By using utility classes:
  - That make the individual calls for you:

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

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



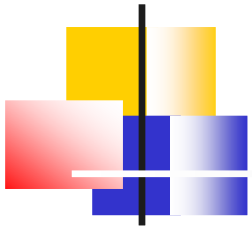
ConstructProcess()

```
void MyPhysicsList::ConstructProcess()
    {
        AddTransportation();
        // Method provided by G4VUserPhysicsList
        // It assignes the transportation process to all
        // particles, with non-zero lifetime, defined
        // in ConstructParticle()

        ConstructEM();
        // Method may be defined by user (for convenience)
        // Instantiate electromagnetic processes here

        ConstructGeneral();
        // Method may be defined by user (for convenience)
    }
```



ConstructEM()

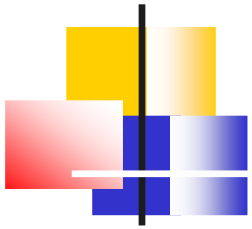
```
void MyPhysicsList::ConstructEM()
    {
        theParticleIterator->reset();
        while( (*theParticleIterator)() ) {
            G4ParticleDefinition*
                particle = theParticleIterator->value();
            G4ProcessManager*
                pmanager = particle->GetProcessManager();
            G4String
                particleName = particle->GetParticleName();
            if (particleName == “gamma”){
                pmanager->AddDiscreteProcess(new G4GammaConversion());
                …
            }
          …
        }
    }
```



ConstructGeneral()

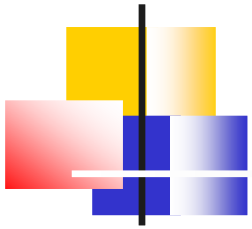
```
void MyPhysicsList::ConstructGeneral()
{
    // Add decay process
    G4Decay* theDecayProcess = new G4Decay();
    theParticleIterator->reset();
    while( (*theParticleIterator)() ) {
        G4ParticleDefinition*
            particle = theParticleIterator->value();
        G4ProcessManager*
            pmanager = particle->GetProcessManager();
        if (theDecayProcess->IsApplicable(*particle) ) {
            pmanager->AddProcess(theDecayProcess);
            pmanager->SetProcessOrdering(theDecayProcess,
                                         idxPostStep);
            pmanager->SetProcessOrdering(theDecayProcess,
                                         idxAtRest);
        }
    }
}
```

Tell more about process ordering later



SetCuts()

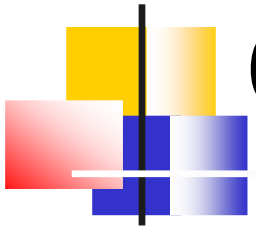
```
void MyPhysicsList::SetCuts()
    {
        defaultCutValue = 1.0*mm;
        SetCutValue(defaultCutValue, “gamma”);
        SetCutValue(defaultCutValue, “e-”);
        SetCutValue(defaultCutValue, “e+”);
        // These are all the production cut
        // values you need to set
        // Not required for any other particle
    }
```



G4VModularPhysicsList

Together with

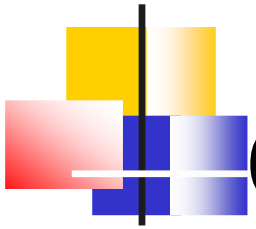
G4VPhysicsConstructor



G4VModularPhysicsList

- Previous physics list was relatively simple

- A realistic physics list is likely to have many more physics processes
  - Such a list can become quite long, complicated and hard to maintain
  - Try a modular physics list instead
- Features of **G4VModularPhysicsList**
  - derived from **G4VUserPhysicsList**
  - **AddTransportation()** automatically called for all registered particles
  - Allows you to define "physics modules":
    - EM physics,
    - hadronic physics,
    - optical physics, etc.



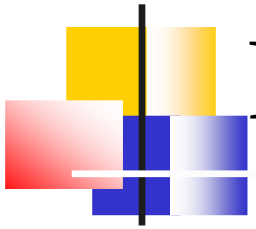
A simple G4VModularPhysicsList

- Constructor:

```
MyModPhysList::MyModPhysList():
    G4VModularPhysicsList()
    {
        defaultCutValue = 1.0*mm;
        RegisterPhysics( new ProtonPhysics() );
        // all physics processes having to do with protons
        RegisterPhysics( new ElectronPhysics() );
        // all physics processes having to do with electrons
        RegisterPhysics( new DecayPhysics() );
        // physics of unstable particles
    }
```

- Set Cuts:

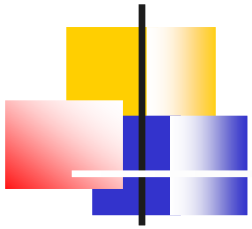
```
void MyModPhysList::SetCuts()
    {
        SetCutsWithDefault();
    }
```



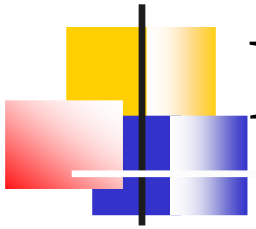
Physics Constructors

- Allow you to group particle and process construction according to physics domains

```
class ProtonPhysics : public G4VPhysicsConstructor
    {
        public:
        ProtonPhysics(const G4String& name = “proton”);
        virtual ~ProtonPhysics();
        virtual void ConstructParticle();
        // easy – only one particle to build in this case
        virtual void ConstructProcess();
        // put here all the processes a proton can have
    }
```



Reference physics lists



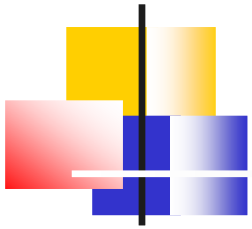
Reference Physics Lists

- Geant4 provides a set of "physics constructors" in:

  `$G4INSTALL/source/physics_lists`

  - **`G4EmStandardPhysics`**
  - **`G4Decay`**
  - **`G4EmExtraPhysics`**
    - Eg: gamma-nuclear
  - **`G4HadronElasticPhysics`**
  - **`G4HadronInelasticPhysics`**
  - **`G4StoppingPhysics`**
    - For particle at rest
  - **`G4IonPhysics`**

- That are combined to build "reference physics lists"



Summary

- All the particles, physics processes, and production cuts needed for an application must go into a physics list

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

- A set of reference physics lists is provided by Geant4
  - Users are encouraged to use / start from these lists
  - … and bring their expertise back from experience they get with these physics lists !