Physics processes in general

CERN Geant4 User's Workshop

November 11th–15th 2002

John Apostolakis

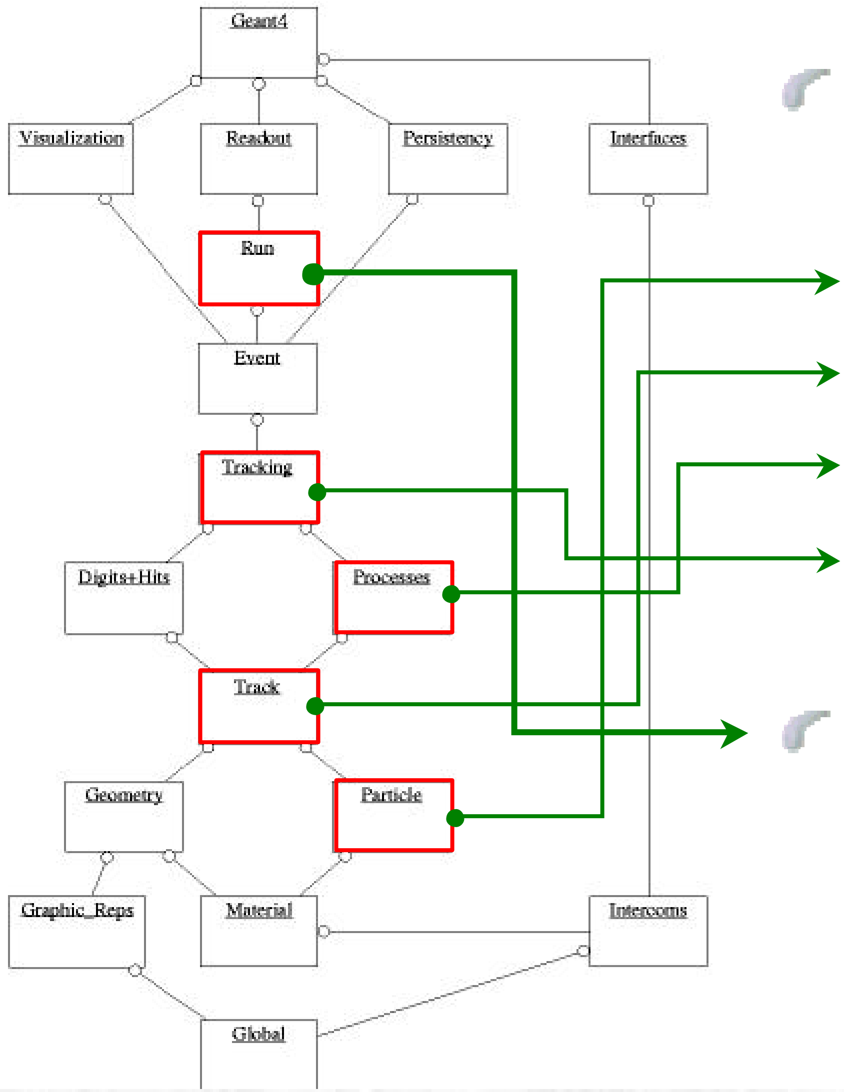
Derived from a talk by Marc Verderi

Ecole Polytechnique - LLR

Introduction

- Present the ingredients needed to understand how to build a « physics list », which is the physics setup:
  - It is the place where the user tells what particles, processes and production cuts will be used in his/her application;
  - This is a ***mandatory*** and ***critical*** user's task;
- We will go through several aspects regarding the « heart » of GEANT4;
- Present only the 'theoretical' aspect of processes:
  - Presentations on the 'concrete' processes follow;

Categories involved



- The ingredients presented sit in these categories:
  - Particles
  - Track
  - Processes
  - We will show how they are handled by the Tracking;
- The « physics list » interface sits in the Run category.

Layout

I. What is tracked;
- Definition of particles;
- G4Track;

II. The process interface;
- G4VProcess;
- How processes are used by the stepping;

III. The production cuts;

IV. Building the « physics lists ».

V. User-defined limits

I. What is tracked in GEANT4;

Speak about:

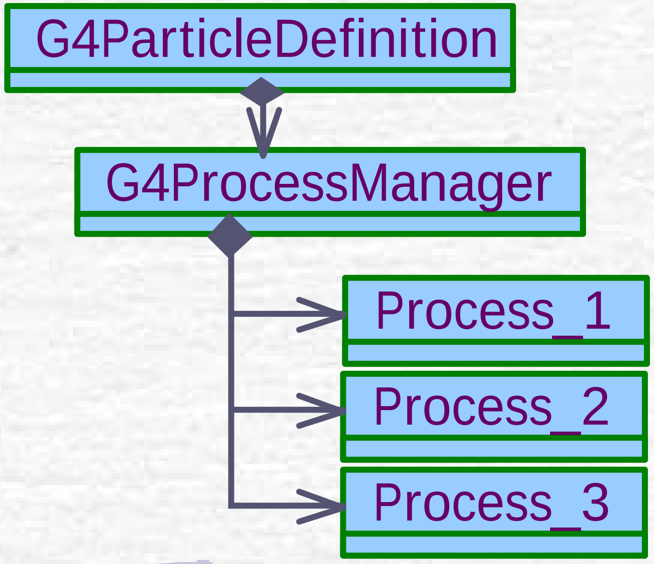
G4ParticleDefinition;

G4DynamicParticle;

G4Track;

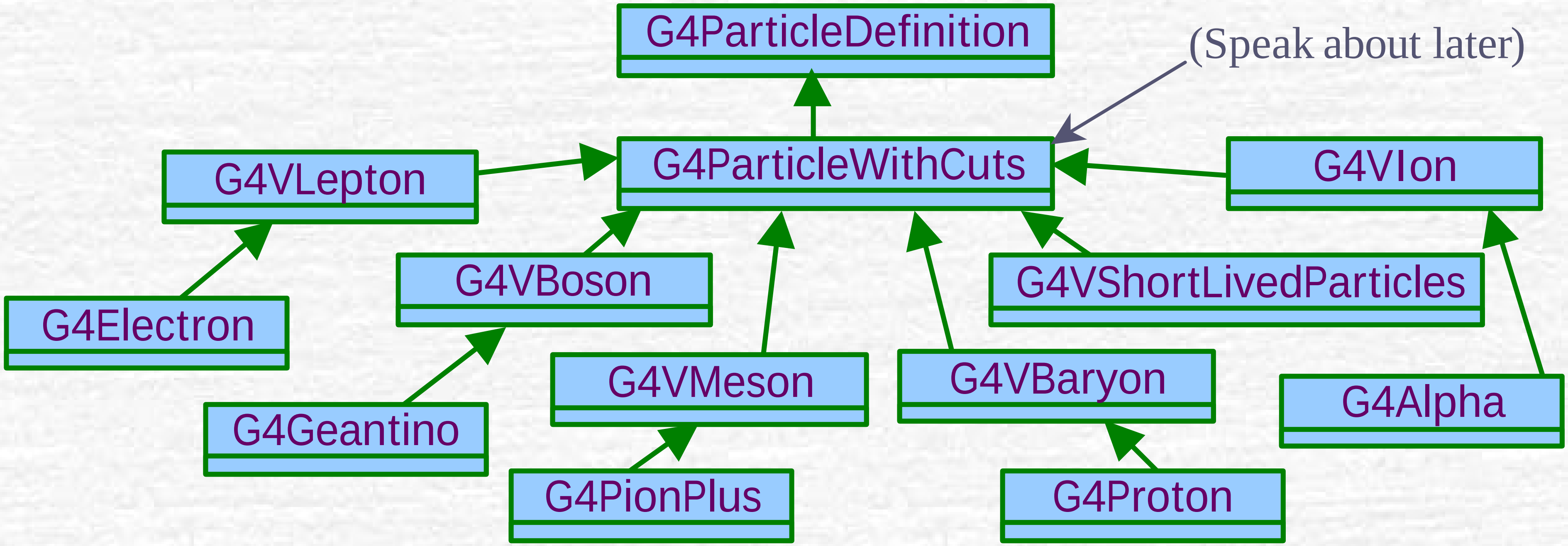
G4ParticleDefinition

- The particle types in GEANT4 are described by the G4ParticleDefinition class;
  - Class defined in source/particles/management;
- Describes the « intrisic » particle properties:
  - Mass, width, spin, lifetime…
- Describes its « sensitivity » to physics:
  - This is realized by a G4ProcessManager;
  - Attached to the G4ParticleDefinition;
  - The G4ProcessManager manages the list of processes the user wants the particle to be sensitive to;
  - Note that G4ParticleDefinition doesn't know by itself its sensitivity to physics.



Concrete G4ParticleDefinition (1)

- G4ParticleDefinition is the base class for defining concrete particles:
  - Several layers are defined:



Concrete G4ParticleDefinition (2)

- Most common particles, with lifetime large enough, are implemented as static classes:
  - Like G4Electron, K^0_S, gamma, pions, but also α...
  - To allow –say- electrons in the simulation, the following call should be made in the « physics list »:
    - G4Electron::ElectronDefinition();
- Heavy ions are created on the fly by processes;
  - Too many ions to have a class per ion !
  - An ion is tracked, and then its « ion type » disapears;
  - Ions are all created from the static class G4GenericIon. To allow heavy ions in the simulation, you should call:
    - G4GenericIon::GenericIonDefinition();
- Resonances (G4VShortLivedParticles) are also created on the fly. Similar calls to the definitions (excited baryons, mesons ...) should be made.

Example: G4Electron class (1)

- Extract from source/particles/leptons/include/G4Electron.hh

```
class G4Electron : public G4VLepton
  {
   public:
        static G4Electron* ElectronDefinition();
   private: //hide constructor as private
        G4Electron(
           const G4String&     aName,        G4double          mass,
           G4double            width,        G4double          charge,
           G4int               iSpin,        G4int             iParity,
           G4int               iConjugation, G4int             iIsospin,
           G4int               iIsospin3,    G4int             gParity,
           const G4String&     pType,        G4int             lepton,
           G4int               baryon,       G4int             encoding,
           G4bool              stable,       G4double          lifetime,
           G4DecayTable       *decaytable);
   private:
        static G4Electron theElectron;

    ...
  };
```

Example: G4Electron class (2)

- Extract from source/particles/leptons/src/G4Electron.cc

```
G4Electron::G4Electron(
        const G4String&    aName,        G4double     mass,
        G4double           width,        G4double     charge,
        G4int              iSpin,        G4int        iParity,
        G4int              iConjugation, G4int        iIsospin,
        G4int              iIsospin3,    G4int        gParity,
        const G4String&    pType,        G4int        lepton,
        G4int              baryon,       G4int        encoding,
        G4bool             stable,       G4double     lifetime,
        G4DecayTable       *decaytable )
   : G4VLepton(    aName,   mass,     width, charge,   iSpin,     IParity,
                iConjugation, iIsospin, iIsospin3, gParity,  pType,
                   lepton,  baryon, encoding,  stable,l  ifetime, decaytable )
   {SetParticleSubType("e");}

G4Electron G4Electron::theElectron(
          "e-",   0.51099906*MeV,   0.0*MeV,    -1.*eplus,
            1,                0,           0,
            0,                0,           0,
     "lepton",                1,           0,          11,
          true,            -1.0,        NULL);

G4Electron* G4Electron::ElectronDefinition(){return &theElectron;}
```

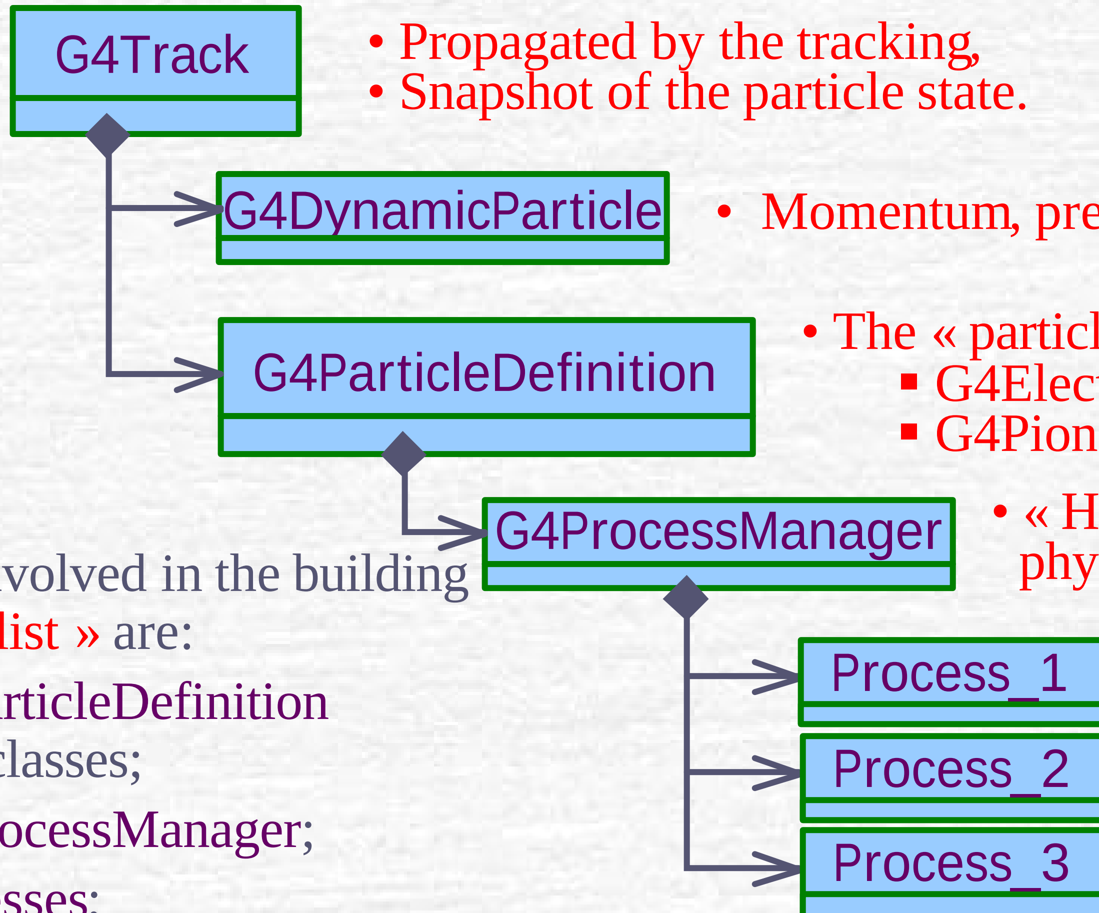
G4DynamicParticle

- G4DynamicParticle describes the purely dynamic part (ie no position, nor geometrical information...) of the particle state:
  - Momentum, energy, polarization;
  - It hangs a G4ParticleDefinition pointer;
  - Retains eventual pre-assigned decay informations:
    - decay products;
    - lifetime;
- Class defined in source/particles/management;

G4Track

- G4Track defines the class of objects propagated by the GEANT4 tracking;
  - Class defined in source/track;
- The G4Track represents a « snapshot » of the particle state;
- A G4Track object agregates:
  - A G4ParticleDefinition;
  - A G4ParticleDynamics;
  - Geometrical informations:
    - Position, current volume …
  - Track ID, parent ID;
  - process which created this G4Track;
  - weigth, used for event biaising technic;
  - …
- A G4Track is tracked from its birth until:
  - It is killed:
    - By an interaction
    - Or because it comes to rest, and is stable;
    - Or, by a user's action (under his responsability !).
  - Or, it exits the world volume;
- Class users need to be familiar with !

Summary view of « What is tracked in GEANT4 »



- Propagated by the tracking,
- Snapshot of the particle state.

- Momentum, pre-assigned decay…

- The « particle type »:
  - G4Electron,
  - G4PionPlus…

- « Hangs » the physics sensitivity;

- The physics processes;

- The classes involved in the building the « physics list » are:
  - The G4ParticleDefinition concrete classes;
  - The G4ProcessManager;
  - The processes;

II. The process interface;

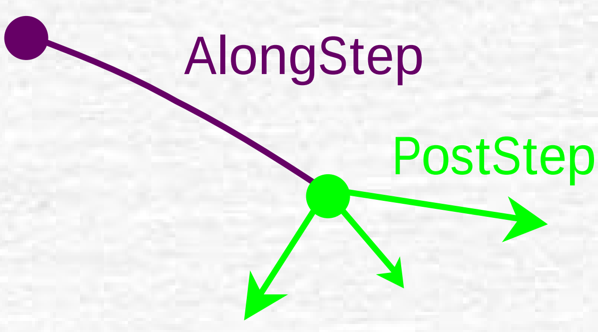
Speak about:

G4VProcess;

The Stepping;

G4VProcess (1)

- Abstract class defining the common interface of all processes in GEANT4:
  - Used by all « physics » processes
  - but is also used by the transportation, etc…
  - Defined in source/processes/management
- Define three kinds of actions:
  - AtRest actions:
    - Decay, e^+ annihilation …
  - AlongStep actions:
    - To describe continuous (inter)actions, occurring along the path of the particle, like ionisation;
  - PostStep actions:
    - For describing point-like (inter)actions, like decay in flight, hard radiation…



G4VProcess (2)

- A process can implement any combination of the three AtRest, AlongStep and PostStep actions:
  - Eg: decay = AtRest + PostStep
- Each action defines two methods:
  - GetPhysicalInteractionLength():
    - Used to ***limit the step size***:
      - either because the process « triggers » an interaction, a decay;
      - Or any other reasons, like fraction of energy loss;
      - geometry boundary;
      - user's limit …
  - DoIt():
    - Implements the ***actual action*** to be applied on the track;
    - And the related production of secondaries.

G4VProcess (3)

- The « action » methods are thus:
  - AtRestGetPhysicalInteractionLength(), AtRestDoIt();
  - AlongStepGetPhysicalInteractionLength(), AlongStepDoIt();
  - PostStepGetPhysicalInteractionLength(), PostStepDoIt();
- A set of processes implementing given combinations of actions exists:
  - G4VDiscreteProcess: only PostStep actions;
  - G4VContinuousDiscreteProcess: AlongStep + PostStep actions;
  - ...
- G4VProcess also defines the method:
  - G4bool IsApplicable(const G4ParticleDefinition &);
  - which returns « true » if the process is applicable to the given particle type;

G4VProcess & G4ProcessManager

- In practice the G4ProcessManager retains three vectors of actions:
  - One for the AtRest methods of the particle;
  - One for the AlongStep ones;
  - And one for the PostStep actions.
- These are those vectors the user sets up in the « physics list » and which are used by the tracking.

How the Stepping handles processes

- The stepping treats processes ***generically***:
  - The stepping does not know what processes it is handling;
- The stepping imposes on the processes to
  - Cooperate in their AlongStep actions;
  - Compete for PostStep and AtRest actions;
- Processes can optionally emit also a «signal» to require particular treatment:
  - notForced: «standard» case;
  - forced: PostStepDoIt action is applied anyway;
  - conditionallyForced: PostStepDoIt applied if AlongStep has limited the step;
  - More on this will be said in the session «Adding a new process»;

Stepping Invocation Sequence of Processes for a particle travelling

1. At the beginning of the step, determine the step length:
   - Consider all processes attached to the current G4Track;
   - Define the step length as the smallest of the lengths among:
     - All AlongStepGetPhysicalInteractionLenght()
     - All PostStepGetPhysicalInteractionLength()
2. Apply ***all*** AlongStepDoIt() actions, « at once »:
   - Changes computed from particle state at the beginning of the step;
   - Accumulated in the G4Step;
   - Then applied to the G4Track, from the G4Step.
3. Apply PostStepDoIt() action(s) « sequentially », as long as the particle is alive:
   - Apply PostStepDoIt() of process which proposed the smallest step length;
   - apply « forced » and « conditionally forced » actions.

Stepping Invokation Sequence of Processes for a Particle at Rest

1. If the particle is at rest, *is stable and can't annihilate*, it is killed by the tracking:
   - To be more accurate: if a particle at rest has no « AtRest » actions defined, it is killed.
2. Otherwise determine the lifetime:
   - Take the smallest time among:
     - All AtRestGetPhysicalInteractionLenght()
       - Called «physical interaction length» but returns a time!
3. Apply the AtRestDoIt() action of the process which returned the smallest time.

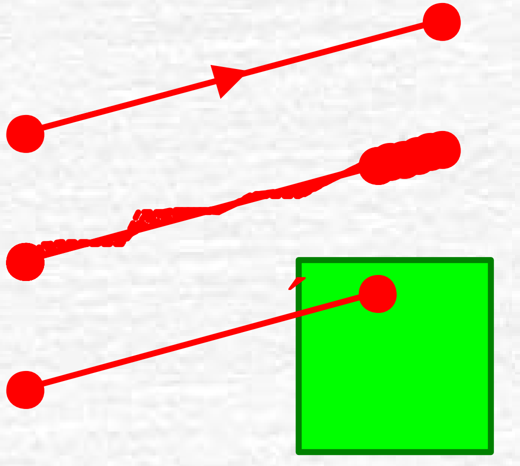
Processes ordering

- **The Ordering of processes matters!**
- Ordering of following processes is critical:
  - Assuming n processes, the ordering of the AlongGetPhysicalInteractionLength of the last processes should be:

    [n-2] ...

    [n-1] multiple scattering

    [n] transportation
- Why ?
  - Processes return a « true path length »;
  - The multiple scattering « virtually folds up » this true path length into a ***shorter*** « geometrical » path length;
  - Based on this new length, the transportation can geometrically limit the step.



- Ordering of most other process does not matter.

III. The production cuts;

Speak about:

Why production cuts are needed;

The cuts scheme in GEANT4

The cuts in GEANT4

- In GEANT4 there is no tracking cut:
  - Particles are tracked down to a zero range/kinetic energy;
- Only production cuts exist;
  - ie cuts allowing a particle to born or not;
- Why are production cuts needed ?
- Some electromagnetic processes involve infrared divergences:
  - This leads to an infinity[huge number] of smaller and smaller energy photons[electrons] (like in bremstrahlung, δ-ray productions);
  - Production cuts limit this production to particles above the threshold;
  - The remaining, divergent part is treated as a « net » continuous effect (ie « AlongStep » action);
- For other processes, production cuts can be an « option » to speed-up the simulation.

Range versus Energy production cuts

- The production of a secondary particle is relevant if it can be « visible » in the detector:
  - Ie produce a signal -say an energy deposition- visible compared to the signal of the primary alone;
- Range cut allows to easily define such visibility:
  - « I want to produce particles able to travel at least 1 mm; »
  - Criteria which can be applied uniformly accross the detector;
- A cut of the same energy would lead to very different ranges:
  - For the same particle type, depending on the material;
  - For the same material, depending on particle type;
- Range cut has been adopted by GEANT4;
- Actual input to cross-section is the energy threshold, but the conversion range-energy is done automatically in GEANT4;

«Violations» of the production threshold

- In some (many) cases, particles are produced, even if they are below the production threshold;
- This is intented to let the processes doing the « best » they can;
- This happens typically for:
  - Decays;
  - Positrons production:
    - In order to simulate the subsequent γ from the annihilation;
  - Hadronic processes:
    - Since no infrared divergences affect the cross-sections;
- Note: these are not « hard-coded » exceptions, but is a sophisticated, generic, mechanism of the tracking;

How GEANT4 produces the production cuts

- The user specifies a range cut for each particle 'type'
  - In the « physics list »;
- This range cut is converted into energy cuts:
  - Each particle -G4ParticleWithCut- converts the range cut into an energy cut, for each material;
- Physics processes can then compute the cross-section based on those energy cuts;
- Done at initialization time;

Relaxing the 'unique' range-cut

- Today each particle type has a unique range-cut
  - One for electrons,
  - one for γ,
  - one for protons,
  - one for other particles
- This ensures consistency
  - but is not optimal for applications in which the accuracy of energy deposition varies greatly between regions of the setup/detector.
- We are developing functionality to relax the restriction of a single cut
  - Allowing each range cut to be set for a region
  - In addition to the 'global'/default range-cut

IV. Building the « physics list »;

Speak about:

G4VUserPhysicsList;

Concrete physics lists;

G4VUserPhysicsList

- It is one of the « mandatory user classes »;
  - Defined in source/run
- Defines the three pure virtual methods:
  - ConstructParticles();
  - ConstructProcesses();
  - SetCuts();
- Need to inherit from G4VUserPhysicsList to implement a physics list;
- (Note that a G4UserPhysicsListMessenger allows to control interactively the physics list.)

ConstructParticles() (1)

- To get particle G4XXX, you need to invoke the static method XXXDefinition() in your ConstructParticles() method:

```
void MyPhysicsList::ConstructParticles()
{
    G4XXX::XXXDefinition();
}
```

- For example, to have electrons, positrons and gammas only:

```
void MyPhysicsList::ConstructParticles()
{
    G4Electron::ElectronDefinition();
    G4Positron::PositronDefinition();
    G4Gamma::GammaDefinition();
}
```

ConstructParticles() (2)

- Alternatively, some helper classes are provided:
  - G4BosonConstructor, G4LeptonConstructor
  - G4MesonConstructor, G4BaryonConstructor
  - G4IonConstructor, G4ShortlivedConstructor
- You can use as:

```
G4BaryonConstructor baryonConstructor;
baryonConstructor.ConstructParticle();
```

- Those helper classes are defined in source/particles/
  - bosons, leptons
  - hadrons/mesons, hadrons/barions
  - hadrons/ions, shortlived

ConstructProcesses() the hard way

- The class heavily used there is the G4ProcessManager:
  - Defined in source/processes/management
  - It is used to attach processes to particles;
  - And set their ordering;
- Several ways to « add » a process:
  - AddProcess
  - AddRestProcess, AddDiscreteProcess, AddContinuousProcess
- And to order AtRest/AlongStep/PostStep actions of processes:
  - SetProcessOrdering
  - SetProcessOrderingToFirst, SetProcessOrderingToLast
  - (This is the ordering for the DoIt() methods, the GetPhysical-InteractionLength() ones have the reverse order.)
- Please review those G4ProcessManager methods !
- Show now various examples.

Example 1: G4VUserPhysicsList::AddTransportation()

- Helper method to add the transportation process:

```
void G4VUserPhysicsList::AddTransportation()
{
    G4Transportation* theTransportationProcess= new G4Transportation();
     // loop over all particles in G4ParticleTable
     theParticleIterator->reset();
     while( (*theParticleIterator)() ){
        G4ParticleDefinition* particle = theParticleIterator->value();
        G4ProcessManager* pmanager = particle->GetProcessManager();
        if (!particle->IsShortLived()) {
           // Add transportation process for all particles other than  "shortlived"
           if ( pmanager == 0) {
              // Error !! no process manager
              G4Exception("G4VUserPhysicsList::AddTransportation : no process manager!");
           } else {
             // add transportation with ordering = ( -1, "first", "first" )
             pmanager->AddProcess(theTransportationProcess);
             pmanager->SetProcessOrderingToFirst(theTransportationProcess, idxAlongStep);
             pmanager->SetProcessOrderingToFirst(theTransportationProcess, idxPostStep);
           }
        } else {
          // shortlived particle case
        }
     }
}
```

Example 2: EM processes for gamma

- Simple example of « discrete » processes: ie only PostStep actions;
  - Show usage of helper function AddDiscreteProcess;
  - **pmanager** is the G4ProcessManager of the gamma;
  - Assume the transportation has been set by AddTransportation;
- Code sample:

```
// Construct processes for gamma:
pmanager->AddDiscreteProcess(new G4GammaConversion());
pmanager->AddDiscreteProcess(new G4ComptonScattering());
pmanager->AddDiscreteProcess(new G4PhotoElectricEffect());
```

- Simple case, where we don’t have to deal with processes ordering (except for the transportation which has been set to « first » elsewhere).
- A more complicated case now…

Example 3: EM processes for positrons

```
// Construct processes for positron
G4VProcess* theeplusMultipleScattering = new G4MultipleScattering();
G4VProcess* theeplusIonisation = new G4eIonisation();
G4VProcess* theeplusBremsstrahlung = new G4eBremsstrahlung();
G4VProcess* theeplusAnnihilation = new G4eplusAnnihilation();
// add processes
pmanager->AddProcess(theeplusMultipleScattering);
pmanager->AddProcess(theeplusIonisation);
pmanager->AddProcess(theeplusBremsstrahlung);
pmanager->AddProcess(theeplusAnnihilation);
// set ordering for AtRestDoIt
pmanager->SetProcessOrderingToFirst(theeplusAnnihilation, idxAtRest);
// set ordering for AlongStepDoIt
pmanager->SetProcessOrdering(theeplusMultipleScattering, idxAlongStep,  1);
pmanager->SetProcessOrdering(theeplusIonisation,         idxAlongStep,  2);
// set ordering for PostStepDoIt
pmanager->SetProcessOrdering(theeplusMultipleScattering, idxPostStep, 1);
pmanager->SetProcessOrdering(theeplusIonisation,         idxPostStep, 2);
pmanager->SetProcessOrdering(theeplusBremsstrahlung,     idxPostStep, 3);
pmanager->SetProcessOrdering(theeplusAnnihilation,       idxPostStep, 4);
```

An alternative way to implement particles and processes

- It exists the G4VModularPhysicsList class:
  - Defined in source/run;
  - Which inherits from G4VUserPhysicsList;
- Which makes use of a set of G4VPhysicsConstructor:
  - Defined in source/run;
- G4VPhysicsConstructor defines the pure virtual methods:
  - ContructParticle();
  - ConstructProcess();
  - It is a kind of « sub-physics list », each of those implementing -say- the EM physics only, the hadronics physics only, etc...
- Allows to avoid all the physics definition in a single class;
- Please see example/novice/N04

For the best and newest way to create the UserPhysicsList

- See the presentation later today on Hadronics Physics Lists,
- It builds on the Modular Physics lists
  - ‘Builders’ modularise further the creation of Physics Lists
    - Accumulating physics models and processes for a particular use case.

SetCuts() (1)

- This pure virtual method is used to define the cut range;
- I will here talk only about the recommended way of setting cuts:
  - Ie: same cut range for all particles;
  - Setting particle dependent cuts is possible but might be reserved to advanced (perverted ? ;->) users.
- The G4VUserPhysicsList base class has the protected member:

```
protected:
    G4double defaultCutValue;
```

  - Which is set to 1.0*mm in the constructor;
- You can eventually change this value in an implementation of SetCuts();
- The helper G4VUserPhysicsList::SetCutsWithDefault method implements the machinery to set the cuts using this defaultCutValue value;

SetCuts() (2)

- A typical implementation of SetCuts() is then simply:

```
void MyPhysicsList::SetCuts()
{
    defaultCutValue = 1.0*mm;
    SetCutsWithDefault();
}
```

V. User-defined limits;

Speak about:

G4UserLimit;

G4UserSpecialCuts process;

G4UserLimit

- This class allows to define the following limits in a given G4LogicalVolume:
  - Maximum step size;
  - Maximum track length;
  - Maximum track time;
  - Minimun kinetic energy;
  - Minimum range;
  - Class defined in source/global/management
- The user can inherit from G4UserLimit, or can instanciate the default implementation;
- The object has then to be set to the G4LogicalVolume;

G4UserSpecialCuts

- How to activate G4UserLimit ?
- The maximum step size is automatically taken into account by the stepping;
  - This is only the case for this G4UserLimit's attribute;
- For the other limits, the G4UserSpecialCuts process (discrete process) can be set in the physics list;
  - Defined in source/processes/transportation
- Or, a simple implementation of discrete process can be done to deal with only the cuts the user is interested in;

Conclusion

- Creating the « physics list »
  - exposes, deliberately, the user to the choice of physics (particles + processes) relevant to his/her application;
  - is a critical & complex task
- To assist users
  - The examples have been used as a starting point;
  - Then a modular structure was created;
  - Now a better, more structured, builder solution has been created; see later talk (J.P. Wellisch).
- Hypernews, email etc..., remain important to exchange experiences and expertise !