Introduction

- GEANT4 has been designed to allow users to implement new processes the kernel will treat as any other processes;
- Processes are designed in a generic/general way:
  - Good, but makes the user to « hit » the meaning of many things;
  - Try to highlight the basics allowing to implement a new process;
    - Indicated by « ☺ » marks;
- Sometimes show some « tricky » details:
  - By sake of completeness;
  - But only a few processes face those tricky issues !
- Simple processes are easy to realize!
Layout

I. G4VProcess interface overview;
   i. The mandatory;
   ii. The optional;

II. Number of interaction length;

III. Method signatures:
   i. GetPhysicalInteractionLength;
   ii. DoIt;
   iii. G4VParticleChange;
G4VProcess interface overview;
The mandatory:

- **G4VProcess** defines 6 *pure virtual* methods:
  - `AtRestGetPhysicalInteractionLength`,
  - `AtRestDoIt`;
  - `AlongStepGetPhysicalInteractionLength`,
  - `AlongStepDoIt`;
  - `PostStepGetPhysicalInteractionLength`,
  - `PostStepDoIt`.

- You **need** to override to implement a new process;
  - Either directly, or by using some intermediate class, like **G4VDiscreteProcess**;
  - The signatures of those methods are described later;
  - **G4VProcess** defined in `source/processes/management`;
The optional:

- **G4VProcess** defines also several virtual (but non pure-virtual) methods. For example:
  - `void BuildPhysicsTable(const G4ParticleDefinition&);`
    - Invoked by the run manager, after cuts are (re)set;
  - `void StartTracking();`
    - Invoked when GEANT4 starts to track a new track;
  - `void EndTracking();`
    - Invoked after the tracking of the track has finished;

- But several other virtual methods exist. Please review **G4VProcess.hh** for details;
« Life cycle » of a process

1. Construction;
   1. BuildPhysicsTable();
   1. StartTracking();
      1. If particle at rest:
         1. AtRestGetPhysicalInteractionLength();
         2. AtRestDoIt();
      2. Else:
         1. PostStepGetPhysicalInteractionLength();
         2. AlongStepGetPhysicalInteractionLength();
         3. AlongStepDoIt();
         4. PostStepDoIt();
   2. EndTracking();
2. Destruction;

• Implementation mandatory
• Implementation optional
II. Number of interaction length

Speak about:

Scheme of the « number of interaction length »
Scheme of « number of interaction length » (2)

- Then the step occurs with the actual step length $L_{\text{step}}$ value;

- At the beginning of the new step:
  - If the process has limited the previous step (ie its interaction occurred), it gets a new $N_{\text{int}}$ value;
  - Otherwise, the process converts back $L_{\text{step}}$ into a number of « consumed » interaction length, which is subtracted to its $N_{\text{int}}$ amount;

- Please review for example G4VDiscreteProcess;
  - Note that all related methods are virtual, allowing to redefine them, if needed.
Scheme of « number of interaction length » (1)

- **G4VProcess** and derived classes implement a scheme based on the « number of interaction length » to define the occurrence of the interaction:
  - useful for discrete processes, dealing with the exponential law;
  - At the beginning of the tracking the process is given a « number of interaction length » $N_{\text{int}}$;
  - According to the exponential law;
  - At the beginning of each step:
    - The (concrete) process evaluates the current « mean free path » $\lambda_{\text{free}}$, given the current material;
    - The « true path length » the process allows to the particle before the interaction occurs is then: $N_{\text{int}} \times \lambda_{\text{free}}$;
    - This value is returned by `GetPhysicalInteractionLength`;
Example: G4VDiscreteProcess

```cpp
inline G4double G4VDiscreteProcess::PostStepGetPhysicalInteractionLength(
    const G4Track& track,
    G4double previousStepSize,
    G4ForceCondition* condition)
{
    if ( (previousStepSize <= 0.0) || (theNumberOfInteractionLengthLeft <= 0.0) ) {
        // beginning of tracking (or just after DoIt of this process)
        ResetNumberOfInteractionLengthLeft();
    } else {
        // subtract NumberOfInteractionLengthLeft
        SubtractNumberOfInteractionLengthLeft(previousStepSize);
        if (theNumberOfInteractionLengthLeft < perMillion)
            theNumberOfInteractionLengthLeft = 0.0;
    }
    //...
    // get mean free path
    currentInteractionLength = GetMeanFreePath(track, previousStepSize, condition);
    G4double value = theNumberOfInteractionLengthLeft * currentInteractionLength;
    return value;
}
```
III. Method signatures;

Speak about:

Method signature of:
GetPhysicalInteractionLength and DoIt;
G4VParticleChange;
GetPhysicalInteractionLength signature

- virtual G4double AtRestGetPhysicalInteractionLength(
  const G4Track& track,
  G4ForceCondition* condition ) = 0;

- virtual G4double AlongStepGetPhysicalInteractionLength(
  const G4Track& track,
  G4double previousStepSize,
  G4double currentMinimumStep,
  G4double& proposedSafety,
  G4GPIILSelection* selection ) = 0;

- virtual G4double PostStepGetPhysicalInteractionLength(
  const G4Track& track,
  G4double previousStepSize,
  G4ForceCondition* condition ) = 0;

- All methods return a G4double;
G4ForceCondition & G4GPILSelection

- GetPhysicalInteractionLength methods involve G4ForceCondition & G4GPILSelection;

- These are two enumerations:
  - They define signals, that processes send to the stepping, to require the treatment they wish from the stepping;
  - Involve ± « delicate/tricky » aspects;
  - Defined in source/track;
G4ForceCondition

- **G4ForceCondition** (AtRest and PostStep) defines requests for treatment of the DoIt methods.

- It can take the values:
  - **NotForced**: Usual case 😊: the DoIt method is invoked if the related GetPhysicalInteractionLength has limited the step;
  - **Forced**: The related DoIt is applied anyway;
  - **Conditionally**: The PostStepDoIt is applied if the AlongStep has limited the step;
  - **ExclusivelyForced**: Only the PostStepDoIt of the process is applied: all other AlongStep and PostStep are ignored;
G4GPILSelection

- More tricky...
- **G4GPILSelection** *(AlongStep)* defines requests for the treatment of the *GetPhysicalInteractionLength* methods.
- It can take the values:
  - **CandidateForSelection**: usual case 😊: the process will be « declared » to have limited the step if it returns the smallest length;
  - **NotCandidateForSelection**: the process will not be « declared » to have limited the step, even if it returns the smallest length;
- In practice, only the multiple-scattering makes use of the « NotCandidateForSelection » signal;
**AlongStepGetPhysicalInteractionLength**

```cpp
G4double AlongStepGetPhysicalInteractionLength(
    const G4Track& track,
    G4double previousStepSize,
    G4double currentMinimumStep,
    G4double& proposedSafety,
    G4GPIILSelection* selection
).
```

- **previousStepSize**: as before 😊;
- **currentMinimumStep**: is the current step size value, during the loop to determine the step length.
- **proposedSafety**, read-write argument: It is the current safety. The process can propose a new safety value; the stepping will retain the smallest one.
- **selection**: in practice only the multiple-scattering returns the NotCandidateForSelection 😞 signal.
AtRestGetPhysicalInteractionLength

- G4double AtRestGetPhysicalInteractionLength(
  const G4Track& track,
  G4ForceCondition* condition);

- G4double returned is a time;
  - Not a length!

- condition can take the values NotForced and Forced;
PostStepGetPhysicalInteractionLength

- `G4double PostStepGetPhysicalInteractionLength(
  const G4Track& track,
  G4double   previousStepSize,
  G4ForceCondition* condition)

- `condition` can take the values NotForced, Forced, Conditionally, ExclusivelyForced;

- `previousStepSize` is the size of the last step done:
  - Processes usually use this value to update the total amount of interaction length 😊;
  - This value is also used by the transportation for some optimisation (avoid redoing geometry operations when the last step was « small enough »);
DoIt signature (1)

- virtual G4VParticleChange* AtRestDoIt(
  const G4Track& track,
  const G4Step& step
) = 0;

- virtual G4VParticleChange* AlongStepDoIt(
  const G4Track& track,
  const G4Step& step
) = 0;

- virtual G4VParticleChange* PostStepDoIt(
  const G4Track& track,
  const G4Step& step
) = 0;
DoIt signature (2)

- All DoIt methods have the same signature:
  - They receive `const G4Track` and `G4Step`;
    - i.e., they are not allowed to change directly the track, nor the step;
  - They return a `G4VParticleChange`:
    - This `G4VParticleChange` returns the changes of the track to the stepping;
      - Not the « delta »;
    - And the eventual creation of secondaries;
    - Need to be familiar with, to implement a process 😊;
G4VParticleChange (1)

- **G4VParticleChange** is defined in source/track
- It defines the virtual methods:
  - `virtual G4Step* UpdateStepForAtRest(G4Step*);`
  - `virtual G4Step* UpdateStepForAlongStep(G4Step*);`
  - `virtual G4Step* UpdateStepForPostStep(G4Step*);`
- Which are used to **communicate the changes to be applied on the primary**;
  - They return the **G4Step** after having updated it;
- Each concrete **G4VParticleChange** should modify only the necessary members of the G4Step;
  - Can be relevant if your **G4VParticleChange** is often used;
G4VParticleChange (2)

- To create **secondaries created by the process**, the following methods have to be used:
  - `void SetNumberOfSecondaries(G4int);`
    - To declare the maximum number of secondaries which will be created by the process;
  - `void AddSecondary(G4Track* aSecondary);`
    - Which has to be called for each secondary created;
- **G4VParticleChange** also defines an **Initialize()** method used to initialize the members which will be changed by the process;
**G4TrackStatus**

- **G4TrackStatus** defines the possible status a track can undertake;
- It is needed when writing a process:

  - `fAlive`, // Continue the tracking
  - `fStopButAlive`, // Invoke active rest physics processes and // and kill the current track afterward
  - `fStopAndKill`, // Kill the current track
  - `fKillTrackAndSecondaries`, // Kill the current track and also // associated secondaries.
  - `fSuspend`, // Suspend the current track
  - `fPostponeToNextEvent` // Postpones the tracking of the current // track to the next event.
Example with G4GammaConversion (1)

- Example with **G4GammaConversion**, which uses a particle change defined in the base class **G4VDiscreteProcess**;

```cpp
G4VParticleChange* G4GammaConversion::PostStepDoIt(
    const G4Track& aTrack,
    const G4Step& aStep)
{
    aParticleChange.Initialize(aTrack);
    // Does the physics...
    aParticleChange.SetNumberOfSecondaries(2);
    // ...
    G4double localEnergyDeposit = 0.;
    
    if (ElectKineEnergy > fminimalEnergy)
    {
        // create G4DynamicParticle object for the particle
        G4DynamicParticle* aParticle1 = new G4DynamicParticle(          
            G4Electron::Electron(), ElectDirection, ElectKineEnergy);
        aParticleChange.AddSecondary(aParticle1);
    }
    else { localEnergyDeposit += ElectKineEnergy; }
}
```
Example with G4GammaConversion (2)

// the e+ is always created (even with Ekine=0) for further annihilation.
// ...
if (PositKineEnergy < fminimalEnergy)
    { localEnergyDeposit += PositKineEnergy; PositKineEnergy = 0.;}
// ...
// create G4DynamicParticle object for the particle2
G4DynamicParticle* aParticle2 = new G4DynamicParticle(G4Positron::Positron(), PositDirection, PositKineEnergy);
aParticleChange.AddSecondary(aParticle2);

aParticleChange.SetLocalEnergyDeposit(localEnergyDeposit);

//
// Kill the incident photon
//
aParticleChange.SetMomentumChange( 0., 0., 0.);
aParticleChange.SetEnergyChange( 0.);
aParticleChange.SetStatusChange( fStopAndKill );
return G4VDiscreteProcess::PostStepDoIt( aTrack, aStep );
Conclusion

- The addition of a process is something relatively « easy » in GEANT4:
  - Adding a new process only requires the user to write his code for his process:
    - ie no change in the G4 kernel;
    - Note this is possible because of the choice of object oriented technology;