Interaction with the kernel

Date

Optional User action classes

- **Optionally you can define additional user action classes**
  - **Define your actions** by deriving concrete classes from:
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStakingAction
    - G4UserTrackingAction
    - G4UserSteppingAction
- **As for the mandatory classes, you also need to instantiate them in the main() function of your application and set them to the run manager by using the SetUserAction() function**

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

Geant4 terminology: overview

	Object	Description
Run	G4Run	Largest unit of simulation, that consist of a sequence of events: If a defined number of events was processed a run is finished.
Event	G4Event	Basic simulation unit in Geant4: If a defined number of primary tracks and all resulting secondary tracks were processed an event is over.
Track	G4Track	A track is NOT a collection of steps: It is a snapshot of the status of a particle after a step was completed (but it does NOT record previous steps). A track is deleted, if the particle leaves world, has zero kinetic energy, ...
Step	G4Step	Represents a particle step in the simulation and includes two points (pre-step point and post-step point).

Examples of usage I

- **G4UserRunAction**
  - Has two methods (BeginOfRunAction and EndOfRunAction) and can be used to initialise, analyse and store histogram
  - Everything User want to know at this stage
- **G4UserEventAction**
  - Has two methods (BeginOfEventAction and EndOfEventAction)
    - One can apply an event selection, for example

Examples of usage II

- **G4UserStakingAction**
  - Classify priority of tracks
- **G4UserTrackingAction**
  - has two methods (PreUserTrakingAction and PostUserTrackinAction)
    - For example used to decide if trajectories should be stored
- **G4UserSteppingAction**
  - Has a method which is invoked at the end of a step

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

Retrieving information from steps and tracks

Class association

- **Relations between G4Track, G4Step and G4StepPoint**
  - Bi-directional association between G4Step and G4Track
  - G4Step has pointers to two G4StepPoint (pre-step and post-step point)

Endpoint of particle step

- **The G4StepPoint class represents the endpoint of a particle step and it contains (among other things):**
  - Geometrical/Material information
    - Coordinates of the particle position
    - The pointer to the physical volume that contains the position
    - The pointer to the material associated with this volume
  - Physics process information
    - Pointer to physics process that defined the step-length of current and previous step

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

Particle step

- **G4Step represents a step of a particle is propagating**
- **A G4Step object stores transient information of the step**
  - In the tracking algorithm, G4Step is updated each time a process was invoked
  - You can extract information from a step after the step is completed
    - Both, the ProcessHits() function of your sensitive detector and UserSteppingAction() of your step action class file receive a pointer to G4Step
    - Typically , you may retrieve information in these functions (for example fill histograms in Stepping action)

Particle step

- **A G4Step object contains**
  - The two endpoints (pre and post step) so one has access to the volumes containing these endpoints
  - Changes in particle properties between the points
    - Difference of particle energy, momentum, ...
  - More step related information
    - Energy deposition on step, step length, time-of-flight, ...
  - A pointer to the associated G4Track object
- **G4Step provides various Get methods to access these information or object istances**
  - G4StepPoint * GetPreStepPoint(), ...

Example: step information in SD

```
// in source file of your sensitive detector:

MySensitiveDetector::ProcessHits(G4Step* step,
                                 G4TouchableHistory*) {

  // Total energy deposition on the step (= energy deposited by energy loss
  //  process and energy of secondaries that were not created since their
  // energy was < Cut):
  G4double energyDeposit = step -> GetTotalEnergyDeposit();

  // Difference of energy , position and momentum of particle between pre-
  // and post-step point
  G4double deltaEnergy = step -> GetDeltaEnergy();
  G4ThreeVector deltaPosition = step -> GetDeltaPosition();
  G4double deltaMomentum = step -> GetDeltaMomentum();

  // Step length
  G4double stepLength = step -> GetStepLength();

}
```

Example: check if step is on boundaries

```
// in the source file of your user step action class:

#include "G4Step.hh"

UserStepAction::UserSteppingAction(const G4Step* step) {

  G4StepPoint* preStepPoint = step -> GetPreStepPoint();
  G4StepPoint* postStepPoint = step -> GetPostStepPoint();

  // Use the GetStepStatus() method of G4StepPoint to get the status of the
  // current step (contained in post-step point) or the previous step
  // (contained in pre-step point):
  if(preStepPoint -> GetStepStatus() == fGeomBoundary) {

     G4cout << "Step starts on geometry boundary" << G4endl;
  }
  if(postStepPoint -> GetStepStatus() == fGeomBoundary) {

     G4cout << "Step ends on geometry boundary" << G4endl;

     // You can retrieve the material of the next volume through the
     // post-step point :
     G4Material* nextMaterial = step -> GetPostStepPoint()->GetMaterial();
  }
}
```

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

Particle Track

- **A Geant4 track represented by G4Track, is NOT a collection of G4Step objects, but**
  - It is a snapshot of the status of a particle after a step was completed (i.e. it has information corresponding the post-step point of the step)
    - Kinetic energy, momentum direction, time, etc
    - It also contains a pointer to a dynamic particle class
  - It does not record information of previous steps
  - However, it has also some information, that is not subject to changes during stepping: trackID, info about primary vertex
    - Primary have trackID=1, secondaries have higher track ID's

Particle Track

- **We learned that G4Track and G4Step have no memory of previous steps, and no G4Track object is available at the end of an event**
  - However, if you activate the use of G4Trajectory object, some track information will be available

G4Trajectory will be available in G4Event

→ Has a collection of G4TrajectorPoint → Store particular information of G4Step

→ Store particular information of G4Tracks

Particle Track

- After each step the track can change its state
- The status can be (in red can only be set by the User)

Track Status	Description
fAlive	The particle is continued to be tracked
fStopButAlive	Kin. Energy = 0, but AtRest process will occur
fStopAndKill	Track has lost identity (has reached world boundary, decayed, ...), Secondaries will be tracked
fKillTrackAndSecondaries	Track and its secondary tracks are killed
fSuspend	Track and its secondary tracks are suspended (pushed to stack)
fPostponeToNextEvent	Track but NOT secondary tracks are postponed to the next event (secondaries are tracked in current event)

Example: retrieving information from tracks

```
// retrieving information from tracks (given the G4Track object "track"):

if(track -> GetTrackID() != 1)  {
    G4cout << "Particle is a secondary" << G4endl;

    // Note in this context, that primary hadrons might loose their identity
    if(track -> GetParentID() == 1)
                G4cout << "But parent was a primary" << G4endl;

    G4VProcess* creatorProcess = track -> GetCreatorProcess();

    if(creatorProcess -> GetProcessName()  == "LowEnergyIoni") {
          G4cout << "Particle was created by the Low-Energy " <<
                << "Ionization process" << G4endl;
    }
}
```

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

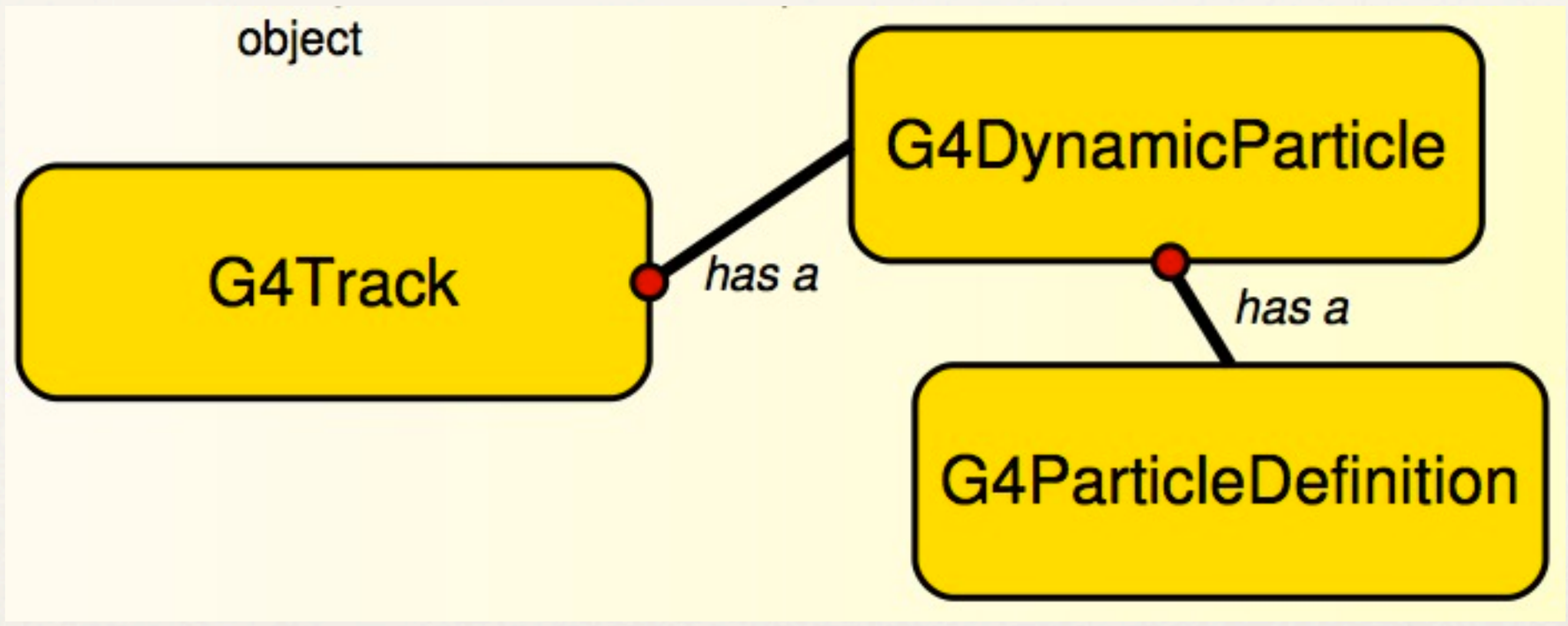
Geant4 Particle Terminology

Class	What does it represent?	What does it contain?
G4Track	Represents a particle that travels in space and time	Information relevant to tracking the particle, e.g. position, time, step,..., and *dynamic information*
G4DynamicParticle	Represents a particle that is subject to interactions with matter	Dynamic information, e.g. particle momentum, kinetic energy, ..., and *static information*
G4ParticleDefinition	Defines a physical particle	Static information, e.g. particle mass, charge, ... Also physics processes relevant to the particle

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

Class hierarchy

- **G4Track has a pointer to a G4DynamicParticle**
- **G4DynamicParticle has a pointer to a G4ParticleDefinition**



Examples: particle information from step

```
#include “G4ParticleDefinition.hh“
#include “G4DynamicParticle.hh“
#include “G4Step.hh“
#include “G4Track.hh“

// Retrieve from the current step the track  (after PostStepDoIt of step is
// completed):
G4Track* track = step -> GetTrack();

// From the track you can obtain the pointer to the dynamic particle:
const G4DynamicParticle* dynParticle = track -> GetDynamicParticle();

// From the dynamic particle, retrieve the particle definition:
G4ParticleDefinition* particle = dynParticle -> GetDefinition();

// The dynamic particle class contains e.g. the kinetic energy after the step:
G4double kinEnergy = dynParticle -> GetKineticEnergy();

// From the particle definition class you can retrieve static information like
// the particle name:
G4String particleName = particle -> GetParticleName();

G4cout << particleName << “: kinetic energy of “
       << kinEnergy/MeV << “ MeV“
       << G4endl;
```

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns

Example: static particle information

```
#include “G4ParticleDefinition.hh“
#include “G4ParticleTable.hh“

G4ParticleDefinition* proton = G4Proton::Definition();

G4double protonPDGMass = proton -> GetPDGMass();
G4double protonPDGCharge = proton -> GetPDGCharge();

G4int protonPDGNumber = proton -> GetPDGEncoding();

G4String protonPartType = proton -> GetParticleType(); // “baryon“
G4String protonPartSubType = proton -> GetParticleSubType(); // “nucleon“

G4int protonBaryonNmb = proton -> GetBaryonNumber();


G4ParticleTable* ptable = G4ParticleTable::GetParticleTable();
G4ParticleDefinition* pionPlus = ptable -> FindParticle(“pi+”);

G4bool particleIsStable = pionPlus -> GetPDGStable();

G4double pionPlusLifeTime = pionPlus -> GetPDGLifeTime();

G4double pionPlusIsospin = pionPlus -> GetPDGIsospin();
```

cirrone@lns.infn.it, http://workgroup.lngs.infn.it/geant4lns