

Geant 4

User Application

<http://cern.ch/geant4>

The full set of lecture notes of this Geant4 Course is available at
<http://www.ge.infn.it/geant4/events/nss2003/geant4course.html>

How to use Geant4

- User Requirements

- Design of a Geant4 application



- Implementation

Focus on :

- User initialisation classes
- User action classes

Capture User Requirements

*Define the scope of the software system to be built
 (“what it should do”)*

The application developer
must study:

- Experimental set-up
- Functionality needed
- Physics involved

1. General

- UR 1.1 Configure the **Run**
 - UR 1.2 Configure the **Event** Loop
-

2. Description of the experimental set-up

- UR 2.1 Describe a **geometrical set-up**: a Si-W tracker, a CsI calorimeter and an anti-coincidence system made out of plastic scintillators.
 - UR 2.2 Record the **coordinates of impact** of tracks in the layers of the tracker. Record the energy release in the strips of the tracker.
 - UR 2.3 Record the **energy deposited** in each element of the **calorimeter** at every event.
 - UR 2.4 Record the **energy deposited** in each element of the **anticoincidence** at every event.
 - UR 2.5 **Digitise** the hits, setting a threshold for the energy deposit in the tracker.
 - UR 2.6 Generate a **trigger** signal combining signals from different detectors.
-

3. Physics

- UR 3.1 Generate **primary events** according to various distributions relevant to gamma astrophysics
 - UR 3.2 Activate **electromagnetic** processes appropriate to the energy range of the experiment.
 - UR 3.3 Activate **hadronic** processes appropriate to the energy range of the experiment.
-

4. Analysis

- UR 4.1 Plot the **x-y distribution** of impact of the track.
 - UR 4.2 Plot **histograms** during the simulation execution.
 - UR 4.3 Store significant quantities in a **ntuple** (energy release in the strips, hit strips) for further analysis.
 - UR 4.4 Plot the **energy distribution** in the calorimeter.
-

5. Visualisation

- UR 5.1 Visualise the experimental **set-up**.
 - UR 5.2 Visualise **tracks** in the experimental set-up.
 - UR 5.3 Visualise **hits** in the experimental set-up.
-

6. User Interface

- UR 6.1 **Configure the tracker**, by modifying the number of active planes, the pitch of the strips, the area of silicon tiles, the material of the converter
 - UR 6.2 **Configure the calorimeter**, by modifying the number of active elements, the number of layers.
 - UR 6.3 **Configure the source**.
 - UR 6.4 **Configure digitisation** by modifying threshold
 - UR 6.5 **Configure the histograms**
-

7. Persistency

- UR 7.1 Produce an intermediate **output** of the simulation at the level of hits in the tracker.
 - UR 7.2 **Store** significant results in FITS format.
 - UR 7.3 **Read in** an intermediate output for further elaboration.
-

Identify a candidate architecture

How the components (geometry, physics, detector, etc.) collaborate in order to satisfy the User Requirements

Documentation:

- G. Booch, OO analysis and design with applications, Addison-Wesley, 1994
- R. Martin, Designing OO C++ applications using the Booch method, Prentice Hall, 1994
- E. Gamma et al., Design Patterns, Addison-Wesley, 1995

Overview of Geant4 advanced examples

Particles

Geometry

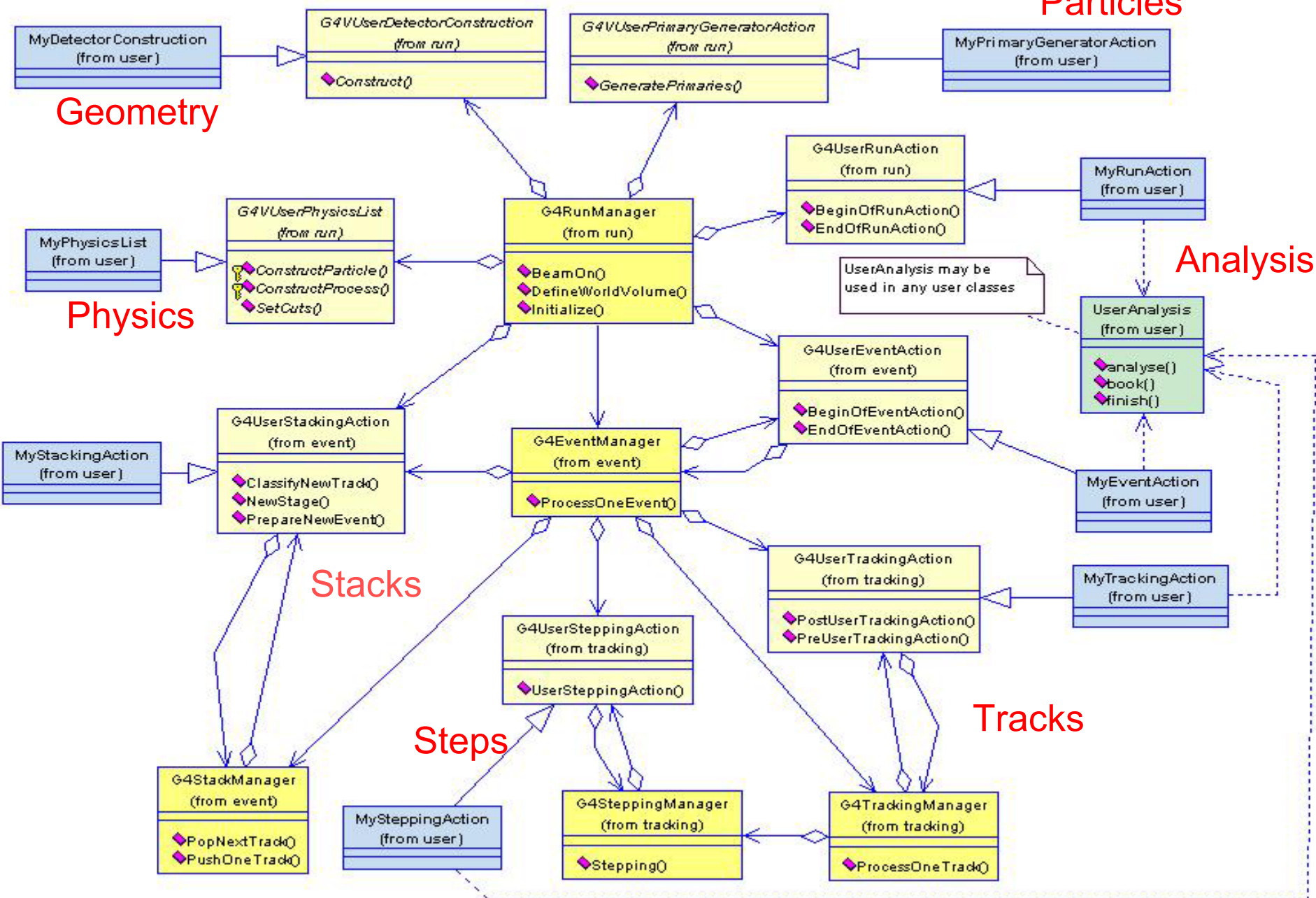
Physics

Stacks

Steps

Tracks

Analysis



User classes

Initialisation classes

- *G4VUserDetectorConstruction*
- *G4VUserPhysicsList*

Action classes

- *G4VUserPrimaryGeneratorAction*
- *G4UserRunAction*
- *G4UserEventAction*
- *G4UserTrackingAction*
- *G4UserStackingAction*
- *G4UserSteppingAction*

Mandatory classes:

- *G4VUserDetectorConstruction*
describe the experimental set-up
- *G4VUserPhysicsList*
select the physics you want to activate
- *G4VUserPrimaryGeneratorAction*
generate primary events

The main program

- Geant4 does not provide the main()
- In his/her main(), the user must
 - construct G4RunManager (or his/her own derived class)
 - notify the mandatory user classes to G4RunManager
 - *G4VUserDetectorConstruction*
 - *G4VUserPhysicsList*
 - *G4VUserPrimaryGeneratorAction*
- The user can define
 - VisManager, (G)UI session, optional user action classes
- in his/her main()

The main program

```
{...  
// Construct the default run manager  
G4RunManager* runManager = new G4RunManager;  
  
// Set mandatory user initialization classes  
MyDetectorConstruction* detector= new MyDetectorConstruction;  
runManager->SetUserInitialization(detector);  
runManager->SetUserInitialization(new MyPhysicsList);  
  
// Set mandatory user action classes  
runManager->SetUserAction(new MyPrimaryGeneratorAction );  
  
// Set optional user action classes  
MyEventAction* eventAction = new MyEventAction();  
runManager->SetUserAction(eventAction);  
MyRunAction* runAction = new MyRunAction();  
runManager->SetUserAction(runAction);  
...}
```

Describe the experimental set-up

- Derive your own concrete class from the `G4VUserDetectorConstruction` abstract base class
- Implement the `Construct()` method
 - construct all necessary materials
 - define shapes/solids required to describe the geometry
 - construct and place volumes of your detector geometry
 - define sensitive detectors and identify detector volumes to associate them to
 - associate magnetic field to detector regions
 - define visualisation attributes for the detector elements

How to define materials 1

Different kinds of materials can be defined

Isotopes
Elements
Molecule
compounds and mixtures isotopes

```
PVPhysicalVolume* MyDetectorConstruction::Construct()
```

```
{...
```

```
  a = 207.19*g/mole;
```

```
  density = 11.35*g/cm3;
```

```
  G4Material* Pb = new G4Material(name="Pb", z=82., a, density);
```

```
  density      = 5.458*mg/cm3;
```

```
  pressure     = 1*atmosphere;
```

```
  temperature  = 293.15*kelvin;
```

```
  G4Material* Xenon = new G4Material(name="XenonGas", z=54.,  
    a=131.29*g/mole, density, kStateGas , temperature ,pressure);
```

```
  ..... }
```

Lead

Xenon
gas

How to define materials 2

```
G4double a = 1.01*g/mole;
```

```
G4Element* H = new G4Element(name="Hydrogen",symbol="H" , z= 1., a);
```

```
a = 12.01*g/mole;
```

```
G4Element* C = new G4Element(name="Carbon" ,symbol="C" , z= 6., a);
```

```
G4double density = 1.032*g/cm3;
```

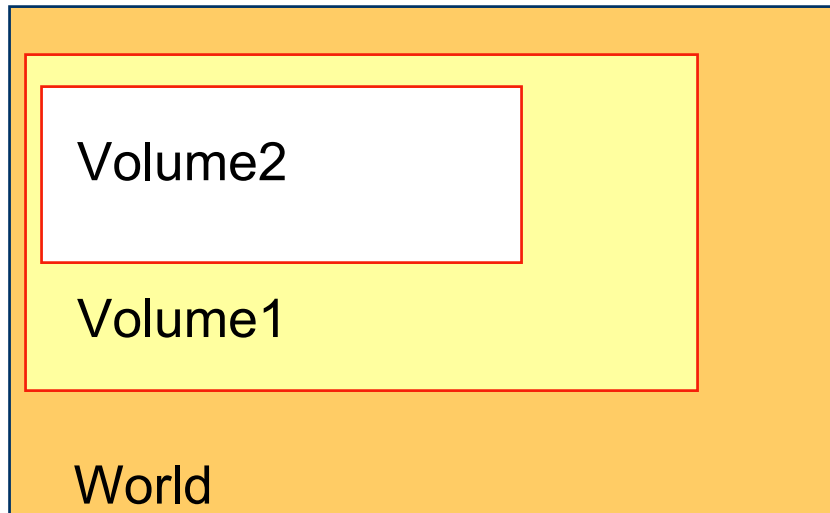
```
G4Material* Sci = new G4Material(name = "Scintillator", density,  
ncomponents = 2);
```

```
Sci -> AddElement(C, natoms = 9);
```

```
Sci -> AddElement(H, natoms = 10);
```

Define detector geometry

- Three conceptual layers
 - **G4VSolid** -- shape, size
 - **G4LogicalVolume** -- material, sensitivity, magnetic field, etc.
 - **G4VPhysicalVolume** -- position, rotation
- A unique physical volume (the world volume), which represents the experimental area, must exist and fully contain all other components



Mother volume: containing volume

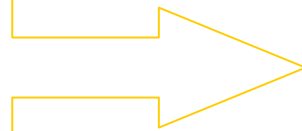
Ex: Volume1 is mother of Volume 2

The mother must contain entirely the daughter volume

Select physics processes

- Geant4 does not have any default particles or processes
- Derive your own concrete class from the G4VUserPhysicsList abstract base class
 - define all necessary **particles**
 - define all necessary **processes** and assign them to proper particles
- **Pure virtual methods**
 - define **production thresholds** (in terms of range)

```
ConstructParticles()  
ConstructProcesses()  
SetCuts()
```



Physics List 1

```
MyPhysicsList :: MyPhysicsList(): G4VUserPhysicsList()
{
    defaultCutValue = 1.0*cm; ← define production thresholds
                              (the same for all particles)
}

MyPhysicsList:: ~MyPhysicsList(){}
void MyPhysicsList :: ConstructParticles()
{
    G4Electron::ElectronDefinition();
    G4Positron::PositronDefinition(); ← define particles involved
    G4Gamma::GammaDefinition();
}

void MyPhysicsList :: SetCuts()
{
    SetCutsWithDefault(); ← Set the cut
}
```


Physics List 2

```
MyPhysicsList :: MyPhysicsList(): G4VUserPhysicsList()
{
    cutForGamma = 1.0*cm;           define production thresholds
    cutForElectron = 1. *mm;
    cutForPositron = 0.1*mm;
};

void MyPhysicsList :: SetCuts()
{
    SetCutValue(cutForGamma, "gamma");
    SetCutValue(cutForElectron, "e-");
    SetCutValue(cutForPositron, "e+");
}
```

the user can define
different cuts!

Physics List 3

```
void MyPhysicsList :: ConstructParticles()
{
```

```
if (particleName == "gamma") {
    pManager->AddDiscreteProcess(new G4PhotoElectricEffect());
    pManager->AddDiscreteProcess(new G4ComptonScattering());
    pManager->AddDiscreteProcess(new G4GammaConversion());

} else if (particleName == "e-") {
    pManager->AddProcess(new G4MultipleScattering(), -1, 1,1);
    pManager->AddProcess(new G4eIonisation(), -1, 2,2);
    pManager->AddProcess(new G4eBremsstrahlung(), -1,-1,3);

} else if (particleName == "e+") {
    pManager->AddProcess(new G4MultipleScattering(), -1, 1,1);
    pManager->AddProcess(new G4eIonisation(), -1, 2,2);
    pManager->AddProcess(new G4eBremsstrahlung(), -1,-1,3);
    pManager->AddProcess(new G4eplusAnnihilation(), 0,-1,4);
}
} select physics processes to be activated for each particle type
```

Primary events

• *Derive your own concrete class from the `G4VUserPrimaryGenerator Action` abstract base*

• *Define primary particles in terms of:*

• *Pure virtual methods:
`GeneratePrimaries()`*

particle type

initial position

initial direction

initial energy

Generate primary events

```
MyPrimaryGeneratorAction :: MyPrimaryGeneratorAction()
{
  G4int n_particle = 1;
  particleGun = new G4ParticleGun (n_particle);
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle = particleTable->FindParticle("e-");
  particleGun->SetParticleDefinition(particle);
  particleGun->SetParticlePosition(G4ThreeVector(x,y,z));
  particleGun->SetParticleMomentumDirection(G4ThreeVector(x,y,z));
  particleGun->SetParticleEnergy(energy);
}
MyPrimaryGeneratorAction :: ~MyPrimaryGeneratorAction() {}
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{ particleGun->GeneratePrimaryVertex(anEvent); }
```

Optional user action classes

- *Five virtual classes whose methods the user may override in order to gain control of the simulation at various stages.*
- *Each method of each action class has an empty default implementation, allowing the user to inherit and implement desired classes and methods.*
- *Objects of user action classes must be registered with G4RunManager.*

Optional user action classes

G4UserRunAction

- **BeginOfRunAction(const G4Run*)**
 - example: book histograms
- **EndOfRunAction(const G4Run*)**
 - example: store histograms

G4UserEventAction

- **BeginOfEventAction(const G4Event*)**
 - example: event selection
- **EndOfEventAction(const G4Event*)**
 - example: analyse the event

G4UserTrackingAction

- **PreUserTrackingAction(const G4Track*)**
 - example: decide whether a trajectory should be stored or not
- **PostUserTrackingAction(const G4Track*)**

G4UserSteppingAction

- **UserSteppingAction(const G4Step*)**
 - example: kill, suspend, postpone the track

G4UserStackingAction

- **PrepareNewEvent()**
 - reset priority control
- **ClassifyNewTrack(const G4Track*)**
 - Invoked every time a new track is pushed
 - Classify a new track (*priority control*)
Urgent, Waiting, PostponeToNextEvent, Kill
- **NewStage()**
 - invoked when the Urgent stack becomes empty
 - change the classification criteria
 - event filtering (*event abortion*)

Summary

- User classes
- Mandatory classes
- Optional action classes
- How to initialise the classes (mandatory and optional) in the main