Programming Guide for Geant4 users

Geant4 Tutorial @ Japan 2007

Geant4 Collaboration

KEK/CRC

Contents

G4-Types

- G4cout, G4cerr

CLHEP Staffs

- Units
- Vector and Rotation matrix
- Random number generation

C++ features in Geant4

- Inheritance
- Singleton

G4 TYPES

Signature for Geant4 classes

Geant4 does not yet introduce *namespace*.

Instead, each class part of the Geant4 kernel has its name beginning *with the prefix G4*.

- e.g., *G4GeometryManager, G4Run,* etc.
- to keep *an homogeneous naming style*
- according to the Geant4 coding style conventions

G4 types

Instead of the raw C types, *G4* types are used within the Geant4 code,

- in order to assure portability
- *G4* types implement the right generic type for a given architecture.
  - ✓ *G4int*
  - ✓ *G4long*
  - ✓ *G4float*
  - ✓ *G4double*
  - ✓ *G4bool*
  - ✓ *G4complex*
  - ✓ *G4String (almost compatible with STL string)*

G4cout, G4cerr

G4cout and *G4cerr* are *ostream* objects defined by Geant4.

- *G4endl* is also provided.
- `G4cout << ”Hello Geant4!” << G4endl;`

Messages can be *treated differently* in (G)UIs other than a command line terminal.

- The user should not use std::cout, etc.
- Ordinary file I/O is OK.

Unit system

Vector and Rotation matrix

Random number generation

CLHEP STAFFS

Unit system

All variables should be given with their units defined in *"SystemOfUnits.h"* of CLHEP.

Each hard-coded number must be multiplied by its proper unit.

- `G4double radius = 10.0 * cm;`
- `G4double kineticE = 1.0 * GeV;`

To get a number, it must be divided by a proper unit.

- `G4cout << eDep / MeV << " [MeV]" << G4endl;`

By this unit system, source code becomes more readable and importing / exporting physical quantities becomes straightforward.

- For particular application, user can change the internal unit to suitable alternative unit without affecting to the result.

Typedefs to CLHEP classes

Typedefs to the corresponding classes of the CLHEP

G4TwoVector, G4ThreeVector, G4RotationMatrix, G4LorentzVector and G4LorentzRotation

- Vector classes: defining 3-component *(x,y,z)* vector entities, rotation of such objects as 3x3 matrices, 4-component *(x,y,z,t)* vector entities and their rotation as 4x4 matrices.

G4Plane3D, G4Transform3D, G4Normal3D, G4Point3D, and G4Vector3D

- Geometrical classes: defining geometrical entities and transformations in 3D space.

The namespace “CLHEP” is introduced in the CLHEP 2.0 versions.

- Geant4 supports both CLHEP 1.9.x and CLHEP 2.x.
  - ✓ no needs for end users to declare “using namespace CLHEP”
  - ✓ “using namespace CLHEP::XXX”s, are declared for CLHEP classes used in Geant4.

An example

```
G4ThreeVector avec = G4ThreeVector(1., 0., 0.);
avec.mag();  // return 1.
avec.rotateZ(90.*deg)  ; // return (0., 1., 0.)

G4ThreeVector avec = G4ThreeVector(1., 0., 0.);
G4RotationMatrix arotM= G4RotationMatrix; // unit matrix
arotM.rotateZ(30.*deg);

// transformation matrix (active transformation)
G4Transform3D atransform= G4Transform3D(arotM, avec);
// 30 degree rotation around the Z axis + shift by (1.,0.,0.)
```

Random number generation

Using the static generator defined in the *HepRandom* class:

- Random values are shot using static methods *shoot()* defined for each distribution (engine) class;
  - ✓ `G4double anumber = HepRandom::shoot();`
- HepJamesRandom as default engine.
- Users can set its properties by *using the static methods* defined in the HepRandom class.
  - ✓`HepRandom::setTheSeed(1234567);`

Random engines

HepJamesRandom (default)

DRand48Engine

RandEngine

RanluxEngine

RanecuEngine

```
RanecuEngine theNewEngine;
HepRandom::setTheEngine(&theNewEngine);
```

Random distributions

A distribution-class can collect different algorithms and different calling sequences for each method to define distribution parameters or range-intervals;

RandFlat

- Class to shoot flat random values (integers or double) within a specified interval.

```
CLHEP::RandFlat::shoot(0., 1.);
```

RandExponential

- Class to shoot exponential distributed random values, given a mean.

RandGauss/RandGaussQ

- Class to shoot Gaussian distributed random values, given a mean (default = 0) or specifying also a deviation (default = 1).
- `CLHEP::RandGaussQ::shoot(mean, deviation);`

RandPoisson

- Class to shoot numbers according to the Poisson distribution, given a mean.

Inheritance

Singleton

C++ FEATURES IN GEANT4

C++ features in Geant4

The most advanced features exposed to users:

- *Inheritance*
  - ✓ The feature that makes a programming language “Object Oriented”
  - ✓ And which makes Geant4 versatile and extendable
- *Singletons*
  - ✓ The technique used for the many “managers” in Geant4
- *A little of templates*
  - ✓ The so-called “generic programming”

Inheritance : the keyword virtual

A class is a "*base class/abstract class/…* " if at least one of its methods is declared "virtual"

Example in G4UserSteppingAction:

- `virtual void UserSteppingAction(const G4Step*) { ; }`
- In this case, it has a *default* implementation.
- You can create an object of this type in memory:
  - ✓ `G4UserSteppingAction dummySteppingAction;`

Example in G4VUserDetectorConstruction:

- `virtual G4VPhysicalVolume* Construct() = 0;`
- It is a so-called "pure virtual method":
  - ✓ *It does not propose a default implementation*
- Creating such an object in memory is not possible
  - ✓ Only pointers on it can be declared:
  - ✓ `G4VUserDetectorConstruction* detector;`

Inheritance in Geant4

Used in many places:

- Geometry:
  - ✓ G4VSolid:
    - – Abstract interface to describe all geometrical shapes
    - – G4Box, G4Tubs, etc… are derived from G4VSolid
      - » (actually G4VCSGSolid, itself derived from G4VSolid)
- Physics:
  - ✓ G4VProcess:
    - – Abstract interface common to all physical processes:
    - – Gamma conversion, multiple scattering, photo-fission, etc…
- Sensitivity:
  - ✓ G4VSensitiveDetector, G4VHit, etc…
- User interfaces:
  - ✓ Detector construction: G4VUserDetectorConstruction
  - ✓ User actions: G4UserTrackingAction, G4UserSteppingAction,…
  - ✓ …

Syntax for inheritance

Example of detector construction, in example N03

In header file G4VUserDetectorConstruction.hh :

```
class G4VUserDetectorConstruction
{
  …
  public:
    virtual G4VPhysicalVolume* Construct() = 0;  // pure virtual!
  …
};
```

In header file ExN03DectectorConstruction.hh :

```
class ExN03DectectorConstruction : public G4VUserDetectorConstruction
{
  …
  public:
    virtual G4VPhysicalVolume* Construct(); // concrete implementation
  …
};
```

Inheritance: a few more things

Remember about publicity keywords:

- "public:" fields are accessible to all
- "protected:" ←
  - ✓ These fields are accessible to daughter classes
- "private:": fields accessible to the class only (and to "friend" classes)

Destructor of a base class is declared virtual.

- In general, to allow your stuff to be deleted when the destructor of the base class is called.

Initialization

- A construction of a daughter class proceeds as follows:
  - ✓ First the constructor of base class is called, then the constructor of the daughter class is called

```
Daughter::Daughter()
  : Parent()
{ … }
```

Singleton

A singleton is a class for which only one instance can be created in memory.

- Most of *manager classes* are implemented as singleton in Geant4.
- G4RunManager, G4EventManager, G4TrackingManager, …

In most cases, the object is like *global* objects with a *static* getter method.

- *can be referred in any places*

-
  ```
  static G4RunManager*
            G4RunManager::GetRunManager();
  ```

Technique makes use of the keyword *static.*

- If in a class declaration a data member is declared static, all objects in memory of that class will share the same data member

Singleton: an example

In **G4SDManager.hh**:

```
class G4SDManager {
  public:
    static G4SDManager* GetSDMpointer();
  …
  private:
    G4SDManager();
  …
  private:
    static G4SDManager* fSDManager;
  …
};
```

In **G4SDManager.cc**:

```
G4SDManager* G4SDManager::fSDManager = 0;

G4SDManager* G4SDManager::GetSDMpointer()
{
  if (!fSDManager)
  {
    fSDManager = new G4SDManager();
  }
  return fSDManager;
}
```

- Since the constructor is *private*, only the class can create a *G4SDManager*
- The *static* pointer (and thus unique) is initialized to zero.
- Then, upon first call to *GetSDMpointer(),* the unique instance is created by

```
G4SDManager* manager = G4SDManager::GetSDMpointer();
```