Messenger mechanism in Geant4

Introduction

Messengers provides

Simplified way to interact with your simulation

User commands help configure simulations dynamically.

No need to modify the code (provided a messenger exist).

Very useful when running Geant4 in batch mode, and simulation needs to be run multiple time with minor changes (material, dimension etc).

Hundreds of messenger command already exist to make your life easier.

User can write the additional required commands depending on the application.

Some commonly used commands

Gun related commands

Event and Tracking related commands

Control commands

Messenger for all these are provided by Geant

Command	Description
`/gun/particle`	Set particle type (e.g., `gamma`).
`/gun/energy`	Set particle energy.
`/gun/position`	Set particle position.
`/gun/direction`	Set initial direction.
`/gun/number`	Set number of particles per event.

Command	Description
`/event/verbose`	Set event-level verbosity.
`/event/setMaxTime time`	Set maximum event processing time.
`/tracking/verbose`	Set tracking-level verbosity.
`/tracking/storeTrajectory`	Enable trajectory storage.

Command	Description
`/control/execute`	Execute a macro file.
`/control/macroPath`	Set path for macros.
`/run/numberOfThreads`	Set number of threads.

What if want to control the size of my detector from macro

Custom Messenger comes into the picture

Messengers can be written for any of your class.

They allows to change your simulation parameters from outside.

How to write a custom messenger

1) Traditional way : using G4UImessenger class

2) Convenient way : using G4GenericMessenger class

Traditional Messenger : Manual

Requires to write custom class inherited from G4UImessenger class

Involves significant boilerplate code.

Complex maintainance and debugging.

Needs to setup a mechanism between the Messenger class and its target.

A separate class needs to be written for each messenger.

For eg. Detector, Primary Generator etc.

Time consuming and prone to human error.

```
class MyMessenger : public G4UImessenger {
public:
    MyMessenger(MyClass* myObj) : obj(myObj) {
        cmd = new G4UIcmdWithADouble("/myApp/param", this);
        cmd->SetGuidance("Set a parameter value");
    }
    void SetNewValue(G4UIcommand* command, G4String value) override {
        if (command == cmd) obj->SetParam(cmd->GetNewDoubleValue(value));
    }
private:
    MyClass* obj;
    G4UIcmdWithADouble* cmd;
};
```

GenericMessenger comes to rescue

Simplifies the create of Messengers.

Allow easy access to class data members

No need to write a separate messenger class.

Only the messenger functions needs to be added to target class.

Easy for maintainance and debugging.

Needs explicit mechanism needs to be setup as there is not separate class

Recipe to create command using GenericMessenger

1) Create an object of G4GenericMessenger in your target class.

```
G4GenericMessenger* messenger = new G4GenericMessenger(this,
"/detector/", "Detector settings");
```

2) Add a command to change the data member of the class (if required)

```
messenger->DeclareProperty("setHalfWorldLength", fHalfWorldSize, "Set
parameter value");
```

3) Create a command and bind it to a callback function to do the required modification.

```
messenger->DeclareMethod("setXHalfLength", &MyClass::SetXHalfLength);
```

Definition of call back function

```
void DetectorConstruction::SetXHalfLength(double size)
{
  G4Box *box = static_cast<G4Box *>(G4LogicalVolumeStore::GetInstance()
->GetVolume("LogicalVolumeName")->GetSolid());
  box->SetXHalfLength(size);
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}
```

One should call ReinitializeGeometry function after making the changes to the geometry. So that the geometry should be rebuild with updated values.

The call-back function can accept only one parameter. Multiple parameter may be passed as string and can be broken into required parameters in the function.

Limitations / Comparison with Traditional Messenger class

Limited to simple command structures

May not support complex hierarchical command patterns (possible with some programming tricks)

GenericMessenger reduces code complexity significantly

Traditional classes provide more control at the cost of increase complexity and more boilerplate code

All the command (Tradition / Generic) can also be used from the macro file.

Conclusion and take away

G4GenericMessenger significantly reduces complexity

Enables more interactive and flexible simulations

Recommended for efficient Geant4 development

Assignments

1) Try to change the color of you NaI detector using UI commands

2) Modify your code to incorporate a messenger that should allow you to change the dimension of you cylindrical NaI crsytal and world volume