

# More on User Interface

Igor Semeniov  
LLR, CNRS - Ecole Polytechnique

Slides from

Laurent GARNIER, IRISA / INS2I / CNRS

Ivana Hrivnacova, IPN / IN2P3 / CNRS

Based on Makoto Asai (SLAC) slides

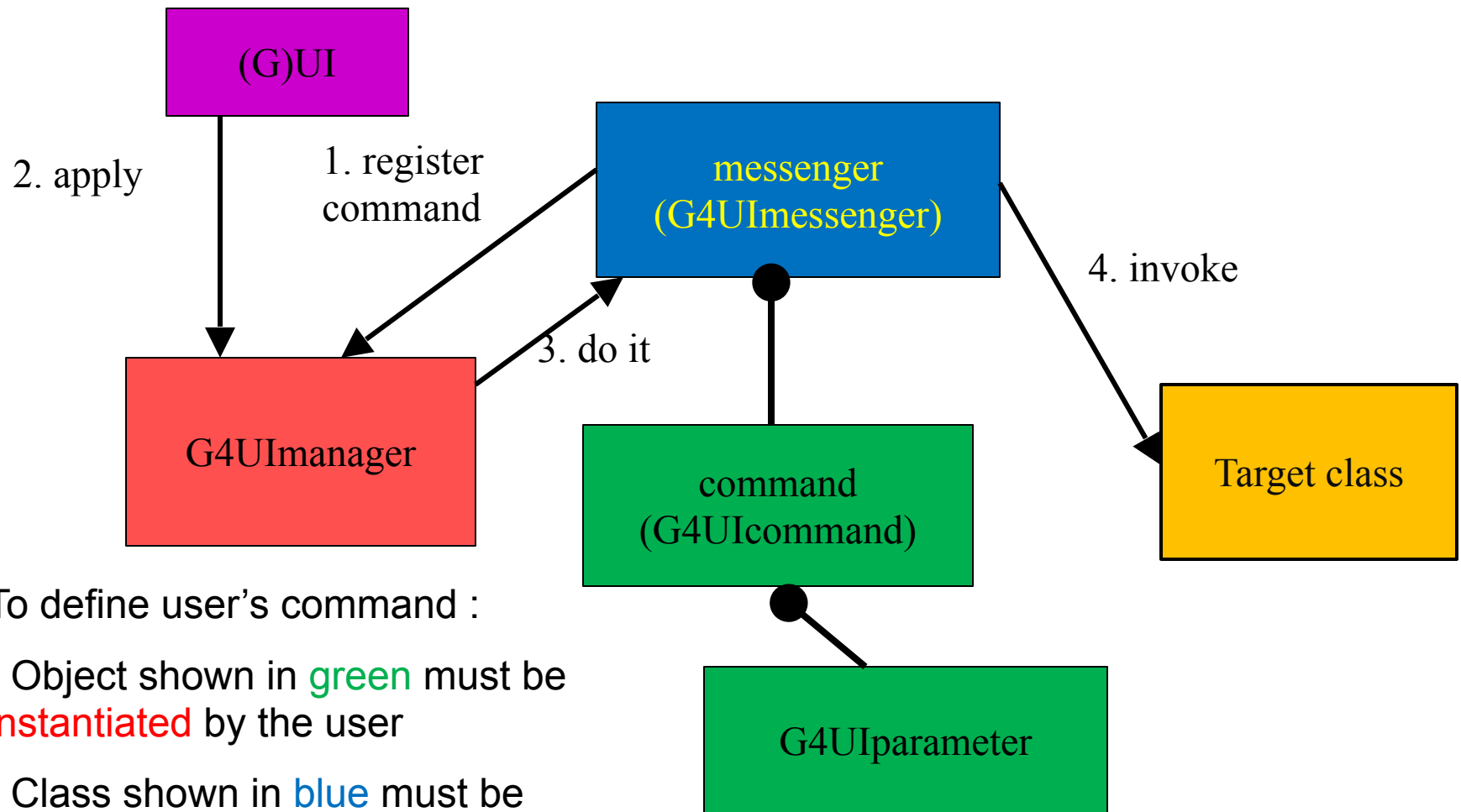


- Mechanism of UI command
- Defining basic UI command
- Defining complex UI command
- Using *G4GenericMessenger*

# Mechanism of UI command



# Mechanism of UI command



To define user's command :

- Object shown in **green** must be **instantiated** by the user
- Class shown in **blue** must be **implemented and instantiated** by the user

# Messenger class

- Each messenger class must be derived from **G4UImessenger** base class. A messenger class can handle more than one UI commands.
- A messenger class **should be instantiated by** the constructor of the **target class** to which commands should be delivered, and **should be deleted** by the destructor of the target class.
- Methods of messenger class
  - **Constructor**
    - Define (instantiate) commands / command directories
  - **Destructor**
    - Delete commands / command directories
  - void **SetNewValue**(G4UIcommand\* command, G4String newValue)
    - Convert "newValue" parameter string to appropriate value(s) and invoke an appropriate method of the target class
    - This method is invoked when a command is issued.
  - G4String **GetCurrentValue**(G4UIcommand\* command)
    - Access to an appropriate get-method of the target class and convert the current value(s) to a string
    - This method is invoked when the current value(s) of parameter(s) of a command is asked by (G)UI.

# Defining basic UI command



# Definition (instantiation) of a command

- To be implemented in the **constructor** of a messenger class.

```
A01DetectorConstMessenger::A01DetectorConstMessenger
(A01DetectorConstruction* tgt)
:target(tgt)
{
    fMydetDir = new G4UIdirectory("/mydet/");
    fMydetDir->SetGuidance("A01 detector setup commands.");

    fArmCmd = new G4UIcmdWithADoubleAndUnit("/mydet/armAngle",
        this);
    fArmCmd->SetGuidance("Rotation angle of the second arm.");
    fArmCmd->SetParameterName("angle", true);
    fArmCmd->SetRange("angle>=0. && angle<180.");
    fArmCmd->SetDefaultValue(30.);
    fArmCmd->SetDefaultUnit("deg");
}
```

- Guidance can (should) be more than one lines. The first line is utilized as a short description of the command.

## G4UIcommand and its derivatives

- **G4UIcommand** is a class which represent a UI command. G4UIparameter represents a parameter.
- G4UIcommand can be directly used for a UI command. Geant4 provides its derivatives according to the types of associating parameters. These derivative command classes already have necessary parameter class object(s), thus you don't have to instantiate G4UIparameter object(s).
  - **G4UIcmdWithoutParameter**
  - **G4UIcmdWithAString**
  - **G4UIcmdWithABool**
  - **G4UIcmdWithAnInteger**
  - **G4UIcmdWithADouble, G4UIcmdWithADoubleAndUnit**
  - **G4UIcmdWith3Vector, G4UIcmdWith3VectorAndUnit**
  - **G4UIdirectory**
- A UI command with other type of parameters must be defined by G4UIcommand base class with G4UIparameter.



## Parameter name(s)

- These methods are available for derivative command classes which take parameter(s).

```
void SetParameterName(  
    const char*parName,  
    G4bool omittable,  
    G4bool currentAsDefault=false);  
void SetParameterName(  
    const char*namX, const char*namY, const char*namZ,  
    G4bool omittable,  
    G4bool currentAsDefault=false);
```

- Parameter names are used in **help**, and also in the **definition of parameter range**.
- If **"omittable" is true**, the command can be issued without this particular parameter, and the default value will be used.
- If **"currentAsDefault" is true**, current value of the parameter is used as a default value, otherwise default value must be defined with SetDefaultValue() method.

# Range, unit and candidates

```
void SetRange(const char* rangeString)
```

- Available for a command with numeric-type parameters.
- Range of parameter(s) must be given in C++ syntax.  
`aCmd->SetRange("x>0. && y>z && z<(x+y)");`
- Not only comparison with hard-coded number but also comparison between variables and simple calculation are available.
- Names of variables must be defined by `SetParameterName()` method.

```
void SetDefaultUnit(const char* defUnit)
```

```
void SetUnitCategory(const char* unitCategory)
```

- Available for a command which takes unit.
- Once the default unit is defined, no other unit of different dimension will be accepted.
- Alternatively, you can define a dimension (unit category) without setting a default unit.

```
void SetCandidates(const char* candidateList)
```

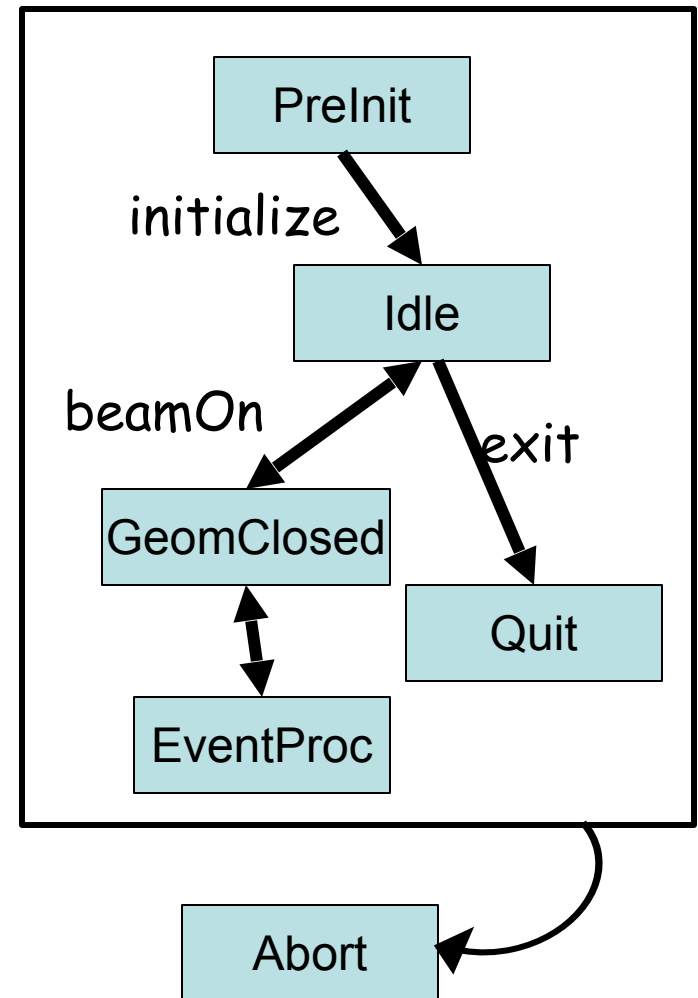
- Available for a command with string type parameter
- Candidates must be delimited by a space.
- Candidates can be dynamically updated.

# Available state

`void AvailableForStates (G4ApplicationState s1,...)`

- Define command's applicability for Geant4 states.
- Geant4 has six application states.
  - G4State\_PreInit
  - G4State\_Idle
  - G4State\_GeomClosed
  - G4State\_EventProc
  - (G4State\_Quit, G4State\_Abort)

application



# Conversion between string and values

- Derivatives of G4UCommand with numeric and boolean parameters have corresponding conversion methods.

- From a string to value

```
G4bool GetNewBoolValue(const char*)
```

```
G4int GetNewIntValue(const char*)
```

```
G4double GetNewDoubleValue(const char*)
```

```
G4ThreeVector GetNew3VectorValue(const char*)
```

- To be used in **SetNewValue()** method in messenger.

- **Unit is taken into account automatically.**

- From value to string

```
G4String ConvertToString(...)
```

```
G4String ConvertToString(..., const char* unit)
```

- To be used in **GetCurrentValue()** method in messenger.

# SetNewValue and GetCurrentValue

```
void A01DetectorConstMessenger
::SetNewValue(G4UIcommand* command,G4String newValue)
{
    if( command==fArmCmd )
    { target->SetArmAngle(fArmCmd->GetNewDoubleValue(newValue)); }
}
```

```
G4String A01DetectorConstMessenger
::GetCurrentValue(G4UIcommand* command)
{
    G4String cv;
    if( command==fArmCmd ) {
        cv = fArmCmd->ConvertToString(target->GetArmAngle(),"deg");
    }
    return cv;
}
```

# Complex commands



# Complex commands

- Complicated UI command means a UI command with parameters which is not included in the deliverable classes
- Eg. a command which requires two different parameters:  
    /path/my\_command parA parB
- Such a command can be defined by `G4UIcommand` class with `G4UIparameter`.

```
myCmd = new G4UIcommand("/path/my_command", this);  
myCmd->SetGuidance("My command with two parameters");  
  
G4UIparameter* parA = new G4UIparameter("parA", 'i', false);  
myCmd->SetParameter(parA);  
G4UIparameter* parB = new G4UIparameter("parB", 'd', false);  
myCmd->SetParameter(parB);
```

# G4GenericMessenger





# G4GenericMessenger

- Since Geant4 9.6 release it is possible to define UI commands using `G4GenericMessenger` class without a need of implementing your own messenger classed
- The commands can be created in four ways
  - Using `G4GenericMessenger::DeclareProperty()` / `G4GenericMessenger::DeclarePropertyWithUnit()`
    - Which associates the command with a class data member (of `G4int`, `G4double`, `G4String` or `G4ThreeVector` type)
  - Using `G4GenericMessenger::DeclareMethod()` / `G4GenericMessenger::DeclareMethodWithUnit()`
    - Which associates the command with a class member function

# Use of Generic Messenger (1/2)

To define a command associated with a class data member

MyRunAction.hh

```
class MyRunAction :  
    G4UserRunAction  
{  
public:  
    MyEventAction();  
    ~MyEventAction();  
  
private:  
    G4GenericMessenger* fMessenger;  
    G4bool fSaveData;  
};
```

MyRunAction.cc

```
MyUserAction::MyRunAction  
: G4UserRunAction(), fMessenger(0)  
{  
    fMessenger  
    = new G4GenericMessenger(  
        this, "/myRun/", "My run control");  
  
    // Define /myRun/setSaveData command  
    G4GenericMessenger::Command& setSaveDataCmd  
    = fMessenger  
    ->DeclareProperty("setSaveData",  
        fSaveData,  
        "(In)Activate saving data");  
  
    setSaveDataCmd. SetStates(G4State_Idle);  
};
```

# Use of Generic Messenger (2/2)

To define a command associated with a class member function

MyDetectorConstruction.hh

```
class MyDetectorConstruction :
    G4VUserDetectorConstruction
{
public:
    MyUserAction();
    ~MyUserAction();

    void SetMagField();

private:
    G4GenericMessenger* fMessenger;
};
```

MyDetectorConstruction.cc

```
MyDetectorConstruction::MyDetectorConstruction
: G4VUserDetectorConstruction(), fMessenger(0)
{
    fMessenger
    = new G4GenericMessenger(
        this, "/myDet/", "My detector control");

    // Define /myDet/setMagField command
    G4GenericMessenger::Command& setMagFieldCmd
    = fMessenger
    ->DeclareMethod("setMagField",
        &MyDetectorConstruction::SetMagField,
        "Define magnetic field value (in X direction)");

    setMagFieldCmd.SetUnitCategory("Magnetic flux");
};
```

# G4GenericMessenger : Command properties (1/2)

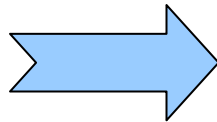
- The commands can be given their guidance, unit, unit category, range, candidates, applicable states via their set methods:
  - **SetStates**(G4ApplicationState s0);
  - **SetRange**(const G4String& range);
  - **SetGuidance**(const G4String& s) ;
  - **SetUnit**(const G4String&, unitSpec = UnitDefault);
  - **SetUnitCategory**(const G4String& u);
  - **SetCandidates**(const G4String&);
  - ...

# G4GenericMessenger : Command properties (2/2)

- Setting properties can be chained
  - No interim local variable is then needed

```
// Define /myRun/setSize command
G4GenericMessenger::Command& setSizeCmd
= fMessenger
  ->DeclareProperty("setSize",
    fSize,
    "Set size of something");

setSizeCmd. SetStates(G4State_PreInit);
setSizeCmd. SetUnitCategory("Length");
setSizeCmd. SetDefaultUnit("cm");
```



```
// Define /myRun/setSize command
fMessenger
->DeclareProperty("setSize",
  fSize,
  "Set size of something")
.SetStates(G4State_PreInit)
.SetUnitCategory("Length")
.SetDefaultUnit("cm");
```