User Interface: UI commands

Mihaly Novak (CERN, EP-SFT)

Getting Started with Geant4 at CERN, Geneva (Switzerland), 25-31 May 2021

- What are user interfaces for?
- UI command syntax
- UI command submission
- Macro file
- Batch mode v.s. interactive mode
- Further information
- Extension: Write your own UI command!

User Interface: UI commands

WHAT ARE USER INTERFACES FOR?

What are user interfaces for?

- **Geant4 is a toolkit**:
  - provides all the necessary **components** needed **to describe and to solve** particle transport **simulation** problems
  - **problem definitions/description:** geometry, particles, physics, etc.
  - **problem solution**: step-by-step particle transport computation
  - while providing **interaction point**s for the user
- **Application programmer**:
  - **develops the simulation application** by making use of the components provided by the toolkit
  - requires solid **knowledge of** both the **C++** programming language **and the** simulation **toolkit**
- **End-user**:
  - **runs the** simulation **application** with the possibility of controlling its flow
  - **doesn't need to have C++** programming **experience**
- **User interfaces makes this possible: control the program flow of a Geant4 simulation application without using C++ language**

User Interface: UI commands

UI COMMAND SYNTAX

- A UI command (e.g. `/run/verbose 1`) consists of:
  - command directory
  - command
  - parameter(s)
- A parameter can be a type of string, boolean, integer or double:
  - space is a delimiter
  - use double-quotes (“”) for strings
- A parameter can be omitted. Its **default** value will be taken in this case:
  - predefined default value or current value according to its definition
  - using the **default** value for the **first parameter** while **setting** the **second**: `/directory/command ! second`
  - i.e. the exclamation mark “!” can be used as a place holder

User Interface: UI commands

UI COMMAND SUBMISSION

- Geant4 UI commands can be issued in 3 different ways by:
  - (G)UI **interactive** command submission (see more later)
  - **batch** mode using a **macro file** (see more later)
  - **hard-coded** commands in the application (slow):

```
G4UImanager* UI = G4UImanager::GetUIpointer();
UI->ApplyCommand("/run/verbose 1");
```

- The availability of the individual commands, the ranges of parameters, the available candidates on individual command parameter **may vary** according to the implementation of your application
  - some commands are available only for limited Geant4 **application state(s):** e.g. `/run/beamOn 100` is available only for ***Idle*** states

- Command will be refused in case of (see example later):
  - Wrong application state
  - Wrong type of parameter
  - Insufficient number of parameters
  - Parameter out of its range:
    - for integer or double type parameter
  - Parameter out of its candidate list
    - for string type parameter
  - Command not found

User Interface: UI commands

MACRO FILE

- A macro file is an ASCII file that contains UI commands
- All commands must be given with their **full-path directories**
- Use "**#**" for **comment a line**
  - from the first "#" to the end of the line will be ignored
  - comment lines will be echoed if `/control/verbose` is set to `2`
- Macro file can be executed
  - interactively or in other macro files

    `/control/execute macro_file_name`
  - hard-coded

```
G4UImanager* UI = G4UImanager::GetUIpointer();
UI->ApplyCommand("/control/execute macro_file_name");
```

User Interface: UI commands

BATCH MODE V.S. INTERACTIVE MODE

See the main method of the B1 example: */examples/basic/B1/exampleB1.cc*

The 3 different ways of command submission: hard-coded, batch and interactive.

```
int main(int argc,char** argv)
{
  // Detect interactive mode (if no arguments) and define UI
  //
  G4UIExecutive* ui = 0;
  if ( argc == 1 ) {
    ui = new G4UIExecutive(argc, argv);
  }
  ...
```

detect interactive mode

```
  ...
  // Get the pointer to the User Interface manager
  G4UImanager* UImanager = G4UImanager::GetUIpointer();
  // Process macro or start UI session
  //
  if ( ! ui ) {
    // batch mode
    G4String command = "/control/execute ";
    G4String fileName = argv[1];
    UImanager->ApplyCommand(command+fileName);
  } else {
    // interactive mode
//    UImanager->ApplyCommand("/control/execute init_vis.mac");
    ui->SessionStart();
    delete ui;
  }
```

batch mode:
process macro

interactive mode:
start an UI session

- **Hard-coded command execution**:
  - can be easily identified in the previous slide
  - the C++ code needs to be rebuild after all changes
- **Batch mode using a Macro file**:
  - in case of batch mode, the commands in the macro file will be executed by processing the macro file using the (hard-coded) `/control/execute macrofile` command
- **Interactive mode**:
  - commands are submitted and processed one-by-one trough a a Geant4 User Interface Session
  - there are several different types of interfaces available:
    - **Qt-GUI**, **GAG-GUI**(java based), **Xm-GUI** (Motif based) or simple C-shell like character terminals (**tcsh**, **csh**)
    - we will use the **tcsh** terminal and the **Qt-GUI**

- **More on the interactive session interface types**:
  - all the different session interface types derived from the abstract ***G4UIsession*** base class
  - an instance of a ***G4UIsession*** (one of the available user interfaces listed in the previous slide) is created and its ***SessionStart()*** method is invoked in the main
  - this will set up the selected UI session
  - the ***G4UIExecutive*** can be used to select the most appropriate UI type available in the current environment:
    - GUI-s will have higher priory over terminals
    - this can be changed by explicitly selecting the required session type (by its name e.g. "**tcsh**") at the construction of the ***G4UIExecutive*** object:

```
G4UIExecutive* ui = 0;
  if ( argc == 1 ) {
    ui = new G4UIExecutive(argc, argv, "tcsh");
  }
```

User Interface: UI commands

FURTHER INFORMATION

- Detailed documentation is available in the Geant4 **Book For Application Developers**: **Control** section
- A list of available built-in commands can be found at the **Built-in Commands** subsection
- How to define custom commands can be found at the **User Interface - Defining New Commands** subsection
- One can use the application to get a list of available commands including the custom ones by:
  - plain text format to standard output

    ```
    /control/manual [directory]
    ```

  - HTML file(s) - one file per one (sub-)directory

    ```
    /control/createHTML [directory]
    ```

Useful Geant4 interactive session terminal commands

- Interactive terminal supports some Unix-like commands for directory.
  - **`cd`**, **`pwd`** - change and display current command directory
    - by setting the current command directory, you may omit (part of) directory string
  - **`ls`** - list available UI commands and sub-directories
- It also supports some other commands.
  - **`history`** - show previous commands
  - **`!historyID`** - re-issue previous command
  - arrow keys and tab (TC-shell only)
  - **`?UIcommand`** - show current parameter values of the command
  - **`help`** [***`UIcommand`***] - help
  - **`exit`** - job termination
- Above commands are interpreted in the interactive terminal and are not passed to Geant4 kernel. You cannot use them in a macro file!

User Interface: UI commands

WRITE YOUR OWN UI COMMAND! (EXTENSION-I.)

Write your own UI command!

- Possibility of creating your own UI command to manipulate some of the properties of your own object (e.g. write a command to be able to `setTargetMaterial [g4 NIST material name]` used in **`YourDetectorConstruction`**)
- You need to write your **`Messenger`** class (with defining/adding your **`UIcommand`**-s) to your object and add to your target object (e.g. **`YourDetectorConstruction`**) which is responsible for instantiating and deleting the messenger object
- **`Messenger`** (covered only what we need, see more at the **User Interface** documentation):
  - handles **`UIcommand`**-s targeting a given object (e.g. **`YourDetectorConstruction`**) and derived from the **`G4UImesenger`** base class (e.g. `class YourDetectorMessenger : public G4UImessenger { … }`)
  - **`UIcommand`**-s (**`UIdirectory`**(-is)) are created in its CTR and deleted in its DTR

```
18      //
19      // create the "det" command directory first then add commands
20      fDirCMD = new G4UIdirectory("/yourApp/det/");
21      fDirCMD->SetGuidance("UI commands specific to the detector construction of this application");
22      //
23      // UI command to set the target thickness
24      fTargetThicknessCMD = new G4UIcmdWithADoubleAndUnit("/yourApp/det/setTargetThickness",this);
25      fTargetThicknessCMD->SetGuidance("Sets the Thickness of the Target.");
26      fTargetThicknessCMD->SetParameterName("TagetSizeX",false);
27      fTargetThicknessCMD->SetRange("TagetSizeX>0.");
28      fTargetThicknessCMD->SetUnitCategory("Length");
29      fTargetThicknessCMD->AvailableForStates(G4State_PreInit, G4State_Idle);
30      fTargetThicknessCMD->SetToBeBroadcasted(false);
```

Write your own UI command!

- Possibility of creating your own UI command to manipulate some of the properties of your own object (e.g. write a command to be able to `setTargetMaterial [g4 NIST material name]` used in **`YourDetectorConstruction`**)
- You need to write your **`Messenger`** class (with defining/adding your **`UIcommand`**-s) to your object and add to your target object (e.g. **`YourDetectorConstruction`**) which is responsible for instantiating and deleting the messenger object
- **`Messenger`** (covered only what we need, see more at the **User Interface** documentation):
  - handles **`UIcommand`**-s targeting a given object (e.g. **`YourDetectorConstruction`**) and derived from the **`G4UImesenger`** base class (e.g. `class YourDetectorMessenger : public G4UImessenger { … }`)
  - **`UIcommand`**-s (**`UIdirectory`**(-is)) are created in its CTR and deleted in its DTR

```
41  YourDetectorMessenger::~YourDetectorMessenger() {
42    delete fTargetThicknessCMD;
43    delete fTargetMaterialCMD;
44    delete fDirCMD;
45  }
```

Write your own UI command!

- Possibility of creating your own UI command to manipulate some of the properties of your own object (e.g. write a command to be able to `setTargetMaterial [g4 NIST material name]` used in **`YourDetectorConstruction`**)
- You need to write your **`Messenger`** class (with defining/adding your **`UIcommand`**-s) to your object and add to your target object (e.g. **`YourDetectorConstruction`**) which is responsible for instantiating and deleting the messenger object
- **`Messenger`** (covered only what we need, see more at the **User Interface** documentation):
  - handles **`UIcommand`**-s targeting a given object (e.g. **`YourDetectorConstruction`**) and derived from the **`G4UImesenger`** base class (e.g. `class YourDetectorMessenger : public G4UImessenger { … }`)
  - **`UIcommand`**-s (**`UIdirectory`**(-is)) are created in its CTR and deleted in its DTR
  - its virtual `void SetNewValue(G4UIcommand* cmd, G4String newValue)` is invoked when a command is invoked:
    - find out which one of your commands has triggered this call by comparing your command pointers (stored as members of your Messenger class) to the `cmd` parameter
    - convert the `newValue` command parameter string to the appropriate value
    - apply the corresponding changes to your target class

Write your own UI command!

```
48  void YourDetectorMessenger::SetNewValue(G4UIcommand* command, G4String newValue)
49  {
50      // set target thickness
51      if (command == fTargetThicknessCMD) {
52          G4double thickness = fTargetThicknessCMD->GetNewDoubleValue(newValue);
53          fYourDetector->SetTargetThickness(thickness);
54      }
55      // set tartget material name
56      if (command == fTargetMaterialCMD) {
57          fYourDetector->SetTargetMaterial(newValue);
58     }
59  }
```

- its virtual `void SetNewValue(G4UIcommand* cmd, G4String newValue)` is invoked when a command is invoked:
  - find out which one of your commands has triggered this call by comparing your command pointers (stored as members of your Messenger class) to the `cmd` parameter
  - convert the `newValue` command parameter string to the appropriate value
  - apply the corresponding changes to your target class

Write your own UI command!

- Possibility of creating your own UI command to manipulate some of the properties of your own object (e.g. write a command to be able to `setTargetMaterial [g4 NIST material name]` used in **`YourDetectorConstruction`**)
- You need to write your **`Messenger`** class (with defining/adding your **`UIcommand`**-s) to your object and add to your target object (e.g. **`YourDetectorConstruction`**) which is responsible for instantiating and deleting the messenger object
- **`Messenger`** (covered only what we need, see more at the **User Interface** documentation):
  - handles **`UIcommand`**-s targeting a given object (e.g. **`YourDetectorConstruction`**) and derived from the **`G4UImesenger`** base class (e.g. `class YourDetectorMessenger : public G4UImessenger { … }`)
  - **`UIcommand`**-s (**`UIdirectory`**(-is)) are created in its CTR and deleted in its DTR
  - its virtual `void SetNewValue(G4UIcommand* cmd, G4String newValue)` is invoked when a command is invoked:
    - find out **which** one of your **command**s has triggered this call by comparing your command pointers (stored as members of your Messenger class) to the `cmd` parameter
    - **convert** the `newValue` command parameter **string to** the appropriate **value type**
    - **apply** the corresponding changes **to** your **target** object
- **Note: the Messenger object needs to be created in the Target object but the Target object needs to be available in the Messenger object!!! (we will get back to this but first see the `UIcommand`-s)**

Write your own UI command!

- Possibility of creating your own UI command to manipulate some of the properties of your own object (e.g. write a command to be able to `setTargetMaterial [g4 NIST material name]` used in **`YourDetectorConstruction`**)
- You need to write your **`Messenger`** class (with defining/adding your **`UIcommand`**-s) to your object and add to your target object (e.g. **`YourDetectorConstruction`**) which is responsible for instantiating and deleting the messenger object
- **`Commands`** (covered only what we need, see more at the **User Interface** documentation):
  - the individual commands, created in the CTR of the **`Messenger`** class
  - they are composed form **UIcommand** and **UIparameter**
  - **`G4UIcommand`** can be used directly with **`G4UIparameter`** for creating UI commands for any parameter type
  - derivatives, according to the **type of the** associated **parameter,** are also available :
    - **`G4UIcmdWithAString, G4UIcmdWithADouble, G4UIcmdWithADoubleAndUnit`**, etc.
  - methods to set the properties of the underlying parameter are available (see next slide)
  - for these UI commands, with numeric and boolean parameters, the corresponding conversion methods (from **string to type** and **from type to string**) are implemented (e.g. `G4double G4UIcmdWithADoubleAndUnit::GetNewDoubleValue(const String newValue)`)

Some methods available for commands:

- `void SetGuidance(const char * aGuidance)`:
  - a description string regarding how to use the command
- `void AvailableForStates(G4ApplicationState s1,…)`:
  - sets the state(s), when the given command can be invoked: **`G4State_PreInit`** (initial condition); **`G4State_Init`** (during initialization); **`G4State_idle`** (ready to start a run)

Some methods, available for derivative commands to set parameter properties:

- `void SetParameterName(const char *name, G4bool omittable, G4bool currentAsDefault=false)`:
  - the user needs to provide a value if `omittable=false;` the current value of the parameter is used as default otherwise if `currentAsDefault` is also `true`
- `void SetDefaultUnit(const char* defUnit)` OR `SetUnitCategory(const char* unitCategory)`:
  - sets the unit of the parameter; either fixed (e.g. "`cm`") or categorical (e.g. "`Length`"); only for commands with numeric parameters
- `void SetRange(const char* rangeString)`:
  - range string parameter must be given in C++ syntax (e.g. `aCmd->SetRange("x>0. && y>z && z<(x+y)");`) only for commands with numeric parameters

Some methods available for commands:

- `void SetGuidance(const char * aGuidance)`:
  - a description string regarding how to use the command
- `void AvailableForStates(G4ApplicationState s1, ...)`:

Time to write our own UI command to set the target material, target thickness!

`unitCategory)`:

- sets the unit of the parameter; either fixed (e.g. "`cm`") or categorical (e.g. "`Length`"); only for commands with numeric parameters

- `void SetRange(const char* rangeString)`:
  - range string parameter must be given in C++ syntax (e.g. `aCmd->SetRange("x>0. && y>z && z<(x+y)");`) only for commands with numeric parameters