How to Create a Python GUI in ROS

Wyatt Newman
October, 2014

This is a step-by-step explanation of how to create a very simple Python GUI in ROS. Refer to the example code in the class repository under: …/catkin/src/examples/example_guis. The examples here are based on prototypes courtesy of Reeve Zhong, Team HKU.

Before following these instructions, first create a *.ui user-interface XML description file. See the document “How to Design a Graphical Interface with QT Creator” for how to do this. Take note of the location of the UI file, mainwindow.ui (which may be renamed, if desired, to *.ui).

In the following, it will be assumed that we are creating a new package called “**example_button_gui**.”

Part I: Boilerplate

Setting up a new GUI package is fairly tedious.

In creating a new GUI package, it will be easiest to **copy/edit existing files**. Refer to the example in **…/catkin/src/examples/example_gui_pub** for these files. The name “example_gui_pub” will appear frequently in the example code, and it should be changed to reflect your new package's name. We first discuss copying over files and editing them to be consistent with your new package.

1) Create your new package. Navigate to the directory where your GUI source code will reside (under …/catkin/src/...). In this terminal, enter the following to create a new package called "example_button_gui."

```
  hku_create_pkg example_button_gui rospy qt_gui rqt_gui
```

2) Add a plugin.xml file. In the new package directory (.../example_button_gui, in this case), copy over an example of
plugin.xml. (see: http://wiki.ros.org/rqt/Tutorials/Create%20your%20new%20rqt%20plugin for more detail/explanation regarding this file).

Inside this file, edit the line:

```
 <class name="Example Gui Pub"
type="example_gui_pub._example_gui_pub_module.ExampleGuiPub"
base_class_type="rqt_gui_py::Plugin">
```

Change “ Example Gui Pub” to (e.g.) “Example Btn GUI.” This name will show up on a list of plugins when you try to find and run your application. It must be a unique name among all of your plug-ins.

Edit the line:
example_gui_pub._example_gui_pub_module.ExampleGuiPub
to (e.g.):
example_button_gui._example_button_gui _module.ExampleBtnGUI

This is the concatenated name of the package (example_button_gui), the module (_example_button_gui_module) and class, (ExampleBtnGUI), which is used for the import statement.

In this example,our package name is "example_button_gui". The designer writes a module, in this case called _example_button_gui_module.py, and within this module a class name is defined, in this case: ExampleBtnGUI. (we will need to refer to these module and class names in Part II below).

In plugin.xml, also edit the line:

```
<label>Example Gui Pub</label>
```

to (e.g.):

```
<label>Example Button Gui</label>
```

The next line:

```
<statustip>Simple GUI to start LIDAR spinning</statustip>
```

should display the message "Simple GUI to greet the worldSimple GUI to start LIDAR spinning" when hovering over the plugin label. However, this does not seem to be working.

3) Create a ui directory. Create a directory under your package name (example_button_gui, in this case), called "ui":

```
roscd  example_button_gui
mkdir ui
```

4) Create a "scripts" directory. Create this directory within your package; e.g., for the example package example_button_gui:

```
roscd  example_button_gui
mkdir scripts
```

5) Create a src directory. Create a subdirectory under src with the same name as your package; in this example:

```
roscd example_button_gui/src
mkdir example_button_gui
```

6) Add your ui file. Copy your *.ui file into the "ui" directory. To help clarify, rename this file “example_button_gui.ui”.
As noted in the document “How to Design a Graphical Interface with QT Creator,” looking inside this file, there will be a line referring to the button widget's object name, e.g. “simpleBtnDemo” from the example. (For more interesting GUI designs, there will be multiple widgets with distinct object names). The name will appear something like this:

```
<widget class="QPushButton" name="simpleBtnDemo">
```

We will need to refer to this name (or these names, for multiple widgets) to write callbacks to respond to user interactions (e.g. button clicks). (See Part II, below)

7) Add a setup.py file. Copy the example file “setup.py” into your package directory. Edit setup.py to change:

```
packages=['example_gui_pub'],
```

to your new package's name:

```
packages=['example_button_gui'],
```

8) Add a top-level Python file. Copy an example python program into your "scripts" directory from another gui package's "scripts" folder, e.g. use “example_gui_pub.py” (or from here: http://wiki.ros.org/rqt/Tutorials/Writing%20a%20Python%20Plugin). Rename this file to match your new package name:

```
   example_button_gui.py
```

Edit this file:
the line:

```
    from example_gui_pub import _example_gui_pub_module
```

should be changed to:

```
            from example_button_gui import _example_button_gui_module
```

The first name is your package name, and the second name is the name of the module that you will write (see step xxx, below). By convention, we will use the same root name as your package, prepend a “_” symbol, and post-pend “_module.”

Additionally in this file, edit the line:

```
   plugin = 'ExampleGuiPub'
```

Change this to (e.g.):

```
   plugin = 'ExampleBtnGUI'
```

The new name above should match the class name used in step 2:

```
            example_button_gui._example_button_gui _module.ExampleBtnGUI
```

This must also match the class name in your module, described below in Part II.

9) Add an init file. Create an init script in your src subdirectory:

```
  roscd example_button_gui/src/ example_button_gui
  touch __init__.py
```

This will create a file with the above name. This file is empty, but its existence is necessary to inform Python to treat the directory as containing executable Python code.

10) Edit CMakeLists.txt:
This file was created by hku_create_pkg. You will see the lines:

```
            project(example_button_gui)
            find_package(catkin_simple REQUIRED)
            catkin_simple()
```

However, you need to manually insert the line “catkin_python_setup(), so it reads:

```
            project(example_button_gui)
            find_package(catkin_simple REQUIRED)
            catkin_python_setup()
            catkin_simple()
```

11) Copy over example module code. Navigate to the directory:

```
    roscd    example_button_gui/src/example_button_gui
```

In this subdirectory, copy in an example module file from:
…/example_gui/pub/src/example_gui_pub/_example_gui_pub_module.py
(or copy from: http://wiki.ros.org/rqt/Tutorials/Writing%20a%20Python%20Plugin).

Change the name of this file in your package to:
 _example_button_gui_module.py

This completes the preparing a new package, and you are now ready to write your code.

Part II: Writing the Module Code:

This is where the interesting work is done. Edit the file:
 _example_button_gui_module.py

This name of this module must match the module name in step (2) of Part I.

Edit the class name. Change:
```
    class ExampleGuiPub(Plugin):
```
to:
```
    class ExampleBtnGUI(Plugin):
```
(consistent with step 2 of Part I, above).

Edit these lines: Use the same name in editing the following lines:
```
    super(ExampleBtnGUI, self).__init__(context)
    # Give QObjects reasonable names
    self.setObjectName('ExampleBtnGUI')
```

Edit the following lines to point to the UI directory. Use your package name, and the name of the *.ui file in your /ui directory (renamed example_button_gui.ui in step 6 of Part I, above).
```
    # Get path to UI file which is a sibling of this file
    # put your package name in the argument:
    pkg_dir = roslib.packages.get_pkg_dir('example_button_gui')
    #put the user-interface file name in the argument below
    ui_file_path = os.path.join(pkg_dir, 'ui/example_button_gui.ui')
```

optionally, change name in this line:
```
    # Give QObjects reasonable names
    self._widget.setObjectName('ExampleGuiUi')
```

Connect the button action to a callback function: Refer to the name(s) assigned to GUI objects, as per the *.ui file: simpleBtnDemo, in this case, and establish a name for the callback function to be associated with this button being clicked, e.g. btn_clicked_CB. Use these two names in the edited code, as below:
```
    # Now Set up our own interface:
    # ==> Connect the button to a functions
    # specify the name of the function to be written as the callback—e.g., btn_clicked_CB.
    self._widget.simpleBtnDemo.clicked.connect(self.btn_clicked_CB )
```

this links the widget "simpleBtnDemo" to a callback function we will write, in this case called "btn_clicked_CB"

Write the callback function Compose a fuction called "btn_clicked_CB"; here is an example:

```
   def btn_clicked_CB(self):
        #print text in terminal
        print "Hello World"
        #printing text in the ui label box
        self._widget.StatusReturn.setText('Hello, World')
```

Here is a more complex example. Upon click of the button, it publishes a message to the topic "multisense_sl/set_spindle_speed" with the argument 3.0 using message type: std_msgs/Float64. (rqrs addl line at top: from std_msgs.msg import Float64)

```
   def btn_clicked_CB(self):
        #set the desired spindle-speed value in variable: spin_speed
        spin_speed = 3.0
        self._widget.StatusReturn.setText('set LIDAR speed to ' + str(spin_speed))
        #publish the data to multisense_sl
            #instantiate a publisher object in ROS, with specified topic and message type:
        self.pub = rospy.Publisher('multisense_sl/set_spindle_speed', Float64, queue_size=10)
        #use this publisher to send the desired message
        self.pub.publish(spin_speed)
```

This completes writing your callback code.

Note: rather than creating a publisher in a callback every time you want to publish a message, it should be more reliable to create it in the class __init__() and assign it to e.g. self.pub.

To extend this simple example to more interesting cases, you can review the available widget options. The widget library that rqt wraps is Qt4 using PyQt4. A full reference to the classes and methods is available here: http://pyqt.sourceforge.net/Docs/PyQt4/classes.html . The names of elements available in the UI editor should correspond to class names in that list.

Running your GUI Application:
You can start up your GUI application within “rqt” as follows:

In a terminal window, enter:

```
    rqt --force-discover
```

(see: http://wiki.ros.org/rqt/UserGuide#Running_rqt for more details and options; usually, just “rqt” will work, but new applications may require “forcing” discovering the first time)

A window will pop up. Under the menu item "Plugins", select your new GUI name. This would be Example Btn GUI , as chosen in step-2 of Part I, above.

Your new application will appear. Click your graphical button to invoke your callback-function response (e.g., to start Atlas's LIDAR spinning).

Alternatively, you can run your GUI from a command line, or start it up from a launch file. To do so, in a terminal, enter this command (using your package name, example_button_gui):

```
rosrun  example_button_gui  example_button_gui.py
```

Running GUI's from the command line is enabled by steps 7 and 10 in part I above. However, some odd behavior has been observed with this (works for some packages and not others). Additional steps/edits may be added to these instructions to correct this behavior.