2.7.2 Creating the GUI in QT Designer

Print

The GUI of our software application is not too complex but still uses many of the most common GUI elements such as a toolbar with a tool button, normal push buttons, group boxes, tabs, labels, text input fields, checkboxes, combo boxes, list views, and a status bar. You already got to know most of these in Section 2.6 but there are also some new ones. The GUI consists of two parts, the GUI for the main window and the GUI for the dialog box that is shown when the user clicks on the "Create new shapefile..." button. Therefore, we will be producing two different .ui files with QT Designer called gui_main.ui and gui_newshapefile.ui. The GUI of the main window uses a vertical layout to organize the different elements into four different rows as shown in the figure below.

Each of the rows is formed by a QGroupBox widget and then other widgets are arranged hierarchically within these group boxes using a combination of grid and horizontal layouts. The tool doesn't have a menu bar but a toolbar at the top with a button to exit the program and a status bar at the bottom. When creating the GUI in QT Designer, it will be important to name the widgets we need to refer to from our main code as indicated by the orange labels in the two figures below. As we already pointed out, the object names given to the widgets in QT Designer will be the names of the variables used for storing the widgets when the .ui file is compiled into a .py file.

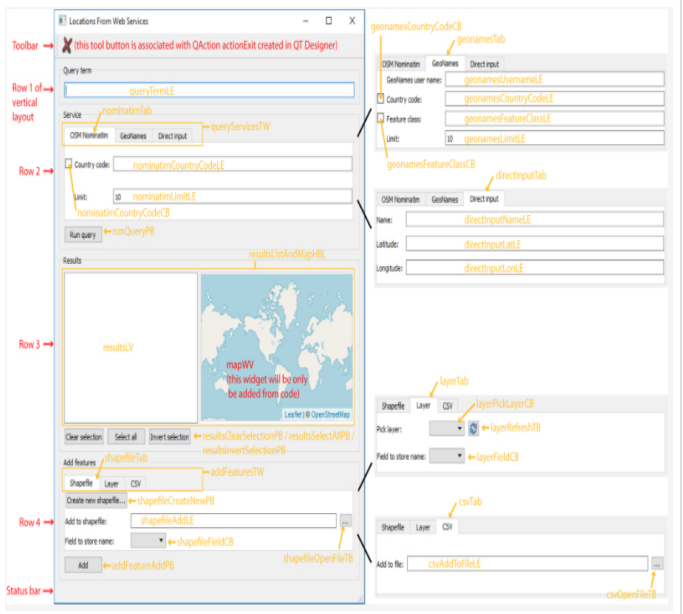

Figure 2.26 Location from Web Services main GUI with labels showing the object names of the different widgets

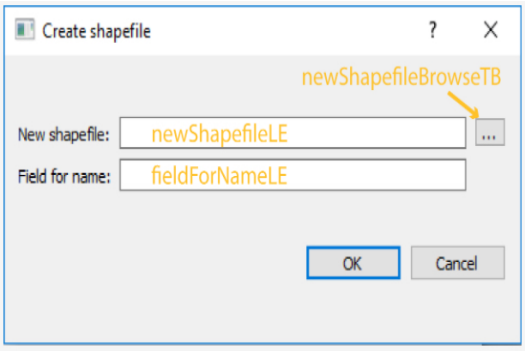

Figure 2.27 GUI of the "Create shapefile" dialog

The series of videos linked below shows the process of creating the GUI in QT Designer. A zip file with the resulting .ui files is available for download here. We recommend that you work along with the video, pausing it as needed, to create the GUI yourself to get some more practice with QT Designer. The downloadable .ui files are mainly intended as a fallback position in case you experience any difficulties while creating the GUI or later on when compiling the .ui files and using the produced Python code. If you cannot replicate what is shown in the video, please ask for help on the discussion forums.

Create the GUI along with this series of videos [~45 min of video materials]

At this point, you either have saved your two own .ui files or, if you ran into any issues, will continue with these files downloaded from the link posted above. We now need to compile the files into Python code with pyuic5. We do this by running the ArcGIS Pro python.exe from the Windows command line with the pyuic module. The python.exe file is located in the folder "C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\" (unless your main ArcGIS Pro installation is located in a different folder). So open a command shell, navigate to the folder containing the .ui files and then run the following two commands:

```
"C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-
py3\python.exe" -m PyQt5.uic.pyuic gui_main.ui -o gui_main.py
```

and

```
"C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-
py3\python.exe" -m PyQt5.uic.pyuic gui_newshapefile.ui -o
gui_newshapefile.py
```

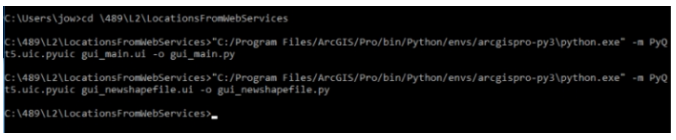

Figure 2.28 Pyuic5 commands to compile the two .ui files into Python files

The parameters given to pyuic5 are the name of the input .ui file and then –o followed by the name of the output file. You should now have the two files gui_main.py and gui_newshapefile.py in the project folder. Let us have a quick look at the produced code. Open the produced file gui_main.py in your preferred Python IDE and see whether you recognize and understand how the different elements are created and how their properties are set. Without going into the details, the code defines a class Ui_MainWindow with a method *setupUi(...)*. The parameter *MainWindow* is for passing a QMainWindow widget to the method. The rest of the code of the method than either

- changes properties of *MainWindow*,
- creates new widgets and layouts storing them as attributes of the Ui_MainWindow object and sets their properties, or
- adds widgets to *MainWindow* or to other widgets to create the hierarchical organization of the widgets.

```
1  class Ui_MainWindow(object):
2
3      def setupUi(self, MainWindow):
4          …
5          MainWindow.resize(605, 685)   # changes property of
```

```
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        …
        MainWindow.setCentralWidget(self.centralwidget) # a
        …
```

This all means that we can create a new QMainWindow widget in our code with …

```
mainWindow = QMainWindow()
```

… create an object of the UI_MainWindow class with

```
ui = gui_main.Ui_MainWindow()
```

… and then create the GUI for the main window by calling *ui.setupUi(…)* with *mainWindow* as the parameter:

```
ui.setupUi(mainWindow)
```

Whenever we need to access a widget created by *setupUi(…)*, we can do so by using the expression

```
ui.<object name of the widget>
```

where the object name is the name we gave the widget in QT Designer, e.g.

```
ui.queryTermLE
```

for the QLineEdit widget we created for entering the query term.

‹ 2.7.1 Overview of the software tool | up | 2.7.3 Main code of the tool and how it works ›

Authors and/or Instructors: James O'Brien, John A. Dutton e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University

Jan Oliver Wallgrün, John A. Dutton e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University

Jim Detwiler, John A. Dutton e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University

Andrew Murdoch, John A. Dutton e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University