Building Mapping Applications with QGIS

Create your own sophisticated applications to analyze and display geospatial information using QGIS and Python

Erik Westra
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 “Getting Started with QGIS”
- A synopsis of the book’s content
- More information on Building Mapping Applications with QGIS

About the Author

Erik Westra has been a professional software developer for over 25 years, and has worked almost exclusively with Python for the past decade. Erik's early interest in graphical user interface design led to the development of one of the most advanced urgent courier dispatch systems used by messenger and courier companies worldwide. In recent years, he has been involved in the design and implementation of systems that match seekers and providers of goods and services across a range of geographical areas. This work has included the creation of real-time geocoders and map-based views of constantly changing data. He is based in New Zealand, and works for companies worldwide.

He is the author of Python Geospatial Development, Packt Publishing.

I would like to thank Ruth, the love of my life, for all her support and encouragement. I would also like to thank my two children, Anneke and Oscar, for reminding me what is important in life.
Building Mapping Applications with QGIS

As software applications become more and more a part of people's lives, the concepts of location and space become more important. Developers are regularly finding themselves having to work with location-based data. Maps, geospatial data, and spatial calculations are increasingly becoming just another part of the everyday programming repertoire.

A decade ago, geospatial concepts and development was limited to experts in the Geographic Information Sciences. These people spent years working with maps and the complex mathematics that underlie them. Often coming from a university background, these specialists would spend years becoming familiar with a particular Geographic Information System (GIS), and would make a career of using that system to draw maps and process geospatial data.

While the ever-popular Google Maps meant that anyone can view and manipulate a map, the more advanced custom display and processing of geospatial data was still limited to those who used a professional GIS system. All this changed with the advent of freely available (and often open source) tools for manipulating and displaying geospatial data. Now, anybody can learn the necessary concepts and start building their own mapping applications from scratch. Rather than being limited to the minimal capabilities and restrictive licensing terms of Google Maps, developers can now build their own mapping systems to meet their own requirements, and there are no limits to what can be done.

While the necessary tools and libraries are freely available, the developer still needs to put them together into a workable system. Often, this is a rather complex process and requires a lot of understanding of geospatial concepts, as well as how to compile the necessary wrappers and configure the tools to work on a particular computer.

Fortunately, now there is an even easier way to include geospatial programming tools and techniques within your Python applications. Thanks to the development of the freely available QGIS system, it is now easy to install a complete geospatial development environment, which you can use directly from within your Python code. Whether you choose to build your application as a plugin for the QGIS system, or write a standalone mapping application using QGIS as an external library, you have complete flexibility in how you use geospatial capabilities within your code.
What This Book Covers

Chapter 1, Getting Started with QGIS, shows you how to install and run the QGIS application, and introduces the three main ways in which Python can be used with QGIS.

Chapter 2, The QGIS Python Console, explores the QGIS Python Console window, and explains how it acts as a useful tool while building your own custom mapping applications. It also gives you a taste of what can be done with Python and QGIS, and improves your confidence and familiarity with the QGIS environment.

Chapter 3, Learning the QGIS Python API, introduces the Python libraries available for the QGIS Python developer, and shows how these libraries can be used to work with geospatial data and create useful and interesting maps based on your geospatial data.

Chapter 4, Creating QGIS Plugins, introduces the concept of a QGIS plugin, and explains how to write a plugin using Python. We take an in-depth look at how plugins work, and how to create a useful geospatial application as a QGIS plugin. We also look at the possibilities and limitations of QGIS plugins.

Chapter 5, Using QGIS in an External Application, completes the process of building standalone Python applications that make use of the QGIS Python libraries. You will learn how to create a wrapper script to handle platform-specific dependencies, design and build a simple but complete standalone mapping application, and learn about the structure of an application built on top of QGIS. Along the way, you will become a far more competent QGIS programmer as you build your own turnkey mapping application from scratch.

Chapter 6, Mastering the QGIS Python API, delves once more into the PyQGIS library, looking at some more advanced aspects of this library, as well as various techniques for working with QGIS using Python.

Chapter 7, Selecting and Editing Features in a PyQGIS Application, looks at how Python programs built using PyQGIS can allow the user to select, add, edit, and delete geospatial features within a map interface.

Chapter 8, Building a Complete Mapping Application Using Python and QGIS, covers the process of designing and building a complete turnkey mapping application called "ForestTrails". You will design the application, implement the overall user interface, and construct a suitable high-resolution basemap for use by the application.

Chapter 9, Completing the ForestTrails Application, covers the completion of the implementation of the "ForestTrails" mapping application by implementing the various map-editing tools, as well as writing a feature to find the shortest available path between two points on the map.
Getting Started with QGIS

This chapter provides an overview of the QGIS system and how you can work with it using the Python programming language. In particular, this chapter will cover the following:

- Downloading, installing, and running QGIS
- Becoming familiar with the QGIS application
- Using Python within QGIS
- Using the Python Console as a window into the QGIS environment
- Working of a QGIS Python plugin
- Interacting with the QGIS Python API from an external Python program

About QGIS

QGIS is a popular, free, and open source Geographic Information System (GIS), which runs on all major operating systems. People often use QGIS to view, edit, and analyze geospatial data. For our purposes, however, QGIS is more than just a GIS system; it is also a geospatial programming environment, which we can use to build our own geospatial applications using Python.

QGIS has a comprehensive website (http://qgis.org), which makes it easy to download, install, and use.

Before reading further, you should spend 15 minutes looking through the website and getting familiar with the application and the documentation available online. In particular, you should check out the Documentation page, where three important manuals are available: QGIS User guide/Manual, QGIS Training manual, and PyQGIS cookbook.
Getting Started with QGIS

QGIS User guide/Manual provides in-depth user documentation, which you might find useful. QGIS Training manual is a detailed introduction to GIS systems and concepts based on QGIS; you might find it useful to work through this course if you aren't already familiar with geospatial data and techniques. Finally, PyQGIS cookbook will be an essential reference to use as you develop your own mapping applications built on top of QGIS.

Installing and running QGIS

If you haven't already installed QGIS, click on the Download Now button on the main QGIS web page to download the QGIS software. What you do next depends on which operating system you are running on your computer:

- For MS Windows, you can download a double-clickable installer that installs QGIS and all the required libraries in one go. Make sure you use the OSGeo4W installer, which includes the Python interpreter, QGIS itself, and all the required libraries.
- For Mac OS X, you'll need to visit the Kyngchaos website (http://www.kyngchaos.com/software/qgis) to download and install the GDAL and matplotlib libraries before installing a version of QGIS specially built for your operating system. All the required packages are available from the Kyngchaos site.
- For Unix-like systems, you'll use a package manager to download, compile, and install QGIS and the required libraries from an appropriate package repository. More information about installing on a Unix-like system can be found at http://qgis.org/en/site/forusers/alldownloads.html#linux.

Once you have installed the QGIS system, you can run it just like any other application on your computer, for example, by double-clicking on the QGIS icon in your Applications folder.
If everything goes well, the QGIS application will start up and you will be greeted with the following window:

![QGIS Window](image)

The exact appearance of the window might vary depending on your operating system. Don't worry, as long as a window appears, which looks something like the one shown in the previous screenshot, you are running QGIS.

You don't need to worry too much about the QGIS user interface right now; the QGIS User Guide describes the interface and various options in great detail. Rather than duplicating this information, let's take a look under the hood to see how QGIS works.
Understanding QGIS concepts

To understand QGIS, you will have to become familiar with the following basic terms and concepts:

- QGIS works with geospatial information loaded from a variety of data sources. These data sources can include vector and raster data files on a disk, a variety of spatial databases, and even web services such as Web Map Service (WMS) servers that provide geospatial data from the Internet.
- Wherever the data comes from, it is retrieved by QGIS and displayed as a map layer. Map layers can be shown or hidden, and also customized in various ways to affect the way the data is displayed on the map.
- The map layers are then combined and displayed on a map.
- Finally, the various map layers, the map, and the other settings, all make up a project. QGIS always has one and only one project that it is working with. The project consists of all the map layers, the map display options, and the various settings that are currently loaded into QGIS.

These concepts are related in the following manner:
Note that the data sources are outside QGIS. While the map layer refers to a data source, the data itself is stored somewhere else, for example, in a file on a disk or within a database.

Whenever you are working with QGIS, you are always working within the current project. You can save projects and reload them later, or start a new project to reset QGIS back to its original state.

### Linking QGIS and Python

While QGIS itself is written in C++, it includes extensive support for Python programming. A Python interpreter is built in, and can be used interactively via the Python Console, or to run plugins written in Python. There is also a comprehensive API for querying and controlling the QGIS application using Python code.

There are three ways in which you can use Python to work with the QGIS system:

- **Python Console**: You can open this console, which runs the interactive Python interpreter built into QGIS, allowing you to type in commands and see the results immediately.

- **Python plugin**: These are Python packages designed to be run within the QGIS environment.

- **External applications**: You can use the QGIS Python API in your own applications. This lets you use QGIS as a geospatial processing engine, or even build your own interactive applications based on QGIS.

No matter how you use Python and QGIS, you will make extensive use of the QGIS Python libraries, which are often referred to as PyQGIS. They provide a complete programmatic interface to the QGIS system, including calls to load data sources into layers, manipulate the map, export map visualizations, and build custom applications using the QGIS user interface. While an in-depth examination of the PyQGIS library will have to wait until Chapter 3, *Learning the QGIS Python API*, we will start dabbling with it right away in the next section on the Python Console.

For the remainder of this chapter, we will examine each of the three ways in which you can work with QGIS and Python.
Exploring the Python Console

The QGIS Python Console window can be accessed by using the Python Console item in the Plugins menu. When you select this command, the Python Console will appear in the lower-right corner of the QGIS window. Here’s what the Python Console looks like when you first open it:

While the Python Console is an excellent tool for interacting with an existing QGIS project, we are going to use it to create a new project from scratch. Before we can do this, though, we’ll need to download some geospatial data sources for our QGIS project.

We are going to need a suitable base map for our project, as well as some river and city information to display on top of this base map. Let's use the Natural Earth website to obtain the information we need. Go to [http://naturalearthdata.com](http://naturalearthdata.com) and click on the Downloads tab.

Firstly, we'll want to download a nice-looking base map for our project. To do this, select the Raster link under the Medium scale data, 1:50m section, choose the Natural Earth 1 dataset, and click on the Download small size link under the Natural Earth I with Shaded Relief and Water heading.

Next, we need an overlay, which will show lakes and rivers on top of our base map. To get this information, go back to the Downloads tab and select the Physical link under the Medium scale data, 1:50m section. The dataset you want is called Rivers, Lake Centerlines, so click on the Download rivers and lake centerlines link to obtain this file.
Finally, we'll want to highlight the cities on top of our base map. Go back to the Downloads page and select the Cultural link under the Medium scale data, 1:50m heading. At the bottom is a section labelled Urban Areas. Click on the Download urban areas link to download this file.

Once you've done all this, you should have the following three files:

- A raster base map in a file named NE1_50M_SR_W.zip
- Lake and river vector data in a file named ne_50m_rivers_lake_centerlines.zip
- Urban area vector data in a file named ne_50m_urban_areas.zip

Since these are ZIP archives, you will need to unzip these files and store them somewhere at a convenient location on your hard disk.

You'll need to type in the full path to these datasets, so you might want to put them somewhere convenient, for example, in your home or user directory. In this way, the path you type won't be too long.

Now that we have our data, let's use the QGIS Python Console to import this data into a project. If you've already loaded some data into QGIS (for example, by following the tutorial in the QGIS User Guide), choose the New option from the Project menu to start again with a blank project. Then, type the following into the QGIS Python Console:

```python
layer1 = iface.addRasterLayer("/path/to/NE1_50M_SR_W/NE1_50M_SR_W.tif", "basemap")
```

Make sure you replace /path/to/ with the full path to the NE1_50M_SR_W directory you downloaded. Assuming you typed the path correctly, the Natural Earth 1 base map should appear in the QGIS window:
As you can see, our base map is a bit small right now. You can use the various panning and zooming commands in the toolbar at the top of the window to make it bigger, but let's use Python to do the same thing:

```python
iface.zoomFull()
```

This will expand the base map to fill the entire window.

Now that we have a base map, let's add our two vector layers to the project. To do this, type the following:

```python
layer2 = iface.addVectorLayer("/path/to/ne_50m_urban_areas/ne_50m_urban_areas.shp", "urban", "ogr")
```

Once again, make sure you replace `/path/to/` with the full path to the `ne_50m_urban_areas` directory you downloaded earlier. The urban areas shapefile will be loaded into the QGIS project and will appear as a series of colored areas on top of the base map. Let's zoom in to an area of California so that we can see what this looks like more clearly. To do this, type the following commands into the Python Console window:

```python
iface.mapCanvas().setExtent(QgsRectangle(-125, 31, -113, 38))
iface.mapCanvas().refresh()
```

This will zoom in on the map in so that an area of California, including Los Angeles and the southern part of San Francisco, is now shown on the map:
Finally, let's add our river and lake data to our project. To do this, enter the following into the Python Console:

```python
layer3 = iface.addVectorLayer("/path/to/ne_50m_rivers_lake_centerlines/ne_50m_rivers_lake_centerlines.shp", "water", "ogr")
```

If you look at the map, you'll see that the rivers and lakes are now visible. However, they are drawn in a default green color. Let's change this so that the water is now blue:

```python
from PyQt4.QtGui import QColor
layer3.rendererV2().symbols()[0].setColor(QColor("#4040FF"))
iface.mapCanvas().refresh()
```

This code might be a bit confusing, but don't worry—we'll learn about renderers and symbols in Chapter 3, *Learning the QGIS Python API*.

Now that we are finished, you can save your project using the **Save As...** item in the **Project** menu. As you can see, it's quite possible to set up and customize your QGIS project using Python.

### Examining a Python plugin

While the Python Console is a fantastic tool for interactive coding, it isn't all that useful if you want to use Python to extend the functionality of QGIS. This is where QGIS plugins come in; you can create (or download) a plugin that adds new features or changes the way QGIS works.

Because QGIS is written using the Qt framework, QGIS plugins make use of the Python bindings in Qt, which are called **PyQt**. We will download and install PyQt and the related tools when we start to build our own plugins in Chapter 4, *Creating QGIS Plugins*.

To get an idea of how a Python plugin works, let's take a look at the **Zoom to Point** plugin. As the name suggests, this plugin lets you zoom to display a given coordinate on the map. It's also written in Python, and is a convenient example for learning about plugins in general.

Before we can use it, we have to install this plugin. Choose the **Manage and Install Plugins...** item from the **Plugins** menu, and click on the **Not Installed** tab. You should see **Zoom to Point** listed near the bottom of the list of available plugins; click on this plugin, and then click on the **Install Plugin** button to download and install it.
Let's run this plugin to see how it works; with the project you created earlier still loaded, click on the **Zoom to Point** plugin's icon in the toolbar, which looks like this:

![Zoom to Point icon](image)

Try entering the longitude/latitude of your current location (if you don't know it, you might find [http://itouchmap.com/latlong.html](http://itouchmap.com/latlong.html) helpful). You should see the base map, urban areas, and waterways for your current location.

Don't forget that x equals longitude and y equals latitude. It's easy to get them the wrong way around.

Now that we know what the plugin does, let's see how it works. The downloaded plugins are stored in a hidden directory named `.qgis2` in your user or home directory. Go to this hidden directory using your favorite file manager (for Mac OS X, you can use the *Go to Folder...* item in the Finder's *Go* menu), and find the `python/plugins` subdirectory. This is where the Python plugins are stored.

Depending on your operating system and the version of QGIS you are using, the name of this hidden directory might be different. If you can't find it, look for a directory named `.qgis` or `.qgis2` or something similar.

You should see a directory named `zoomtopoint` (the full path to this directory will be `~/.qgis2/python/plugins/zoomtopoint`). Inside this directory, you will find the various files that make up the Zoom to Point plugin:

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>.py</td>
<td>1 KB</td>
<td>Python Source File</td>
</tr>
<tr>
<td>COPYING</td>
<td>19 KB</td>
<td>Document</td>
</tr>
<tr>
<td>icon.png</td>
<td>1 KB</td>
<td>Portable Network Graphics image</td>
</tr>
<tr>
<td>Makefile</td>
<td>2 KB</td>
<td>Document</td>
</tr>
<tr>
<td>metadata.txt</td>
<td>516 bytes</td>
<td>Plain Text File</td>
</tr>
<tr>
<td>resources.py</td>
<td>6 KB</td>
<td>Python Source File</td>
</tr>
<tr>
<td>resources.qrc</td>
<td>107 bytes</td>
<td>Document</td>
</tr>
<tr>
<td>ui_zoomtopoint.py</td>
<td>5 KB</td>
<td>Python Source File</td>
</tr>
<tr>
<td>ui_zoomtopoint.ui</td>
<td>5 KB</td>
<td>Document</td>
</tr>
<tr>
<td>zoompoint.py</td>
<td>4 KB</td>
<td>Python Source File</td>
</tr>
<tr>
<td>zoompointdialog.py</td>
<td>1 KB</td>
<td>Python Source File</td>
</tr>
<tr>
<td>zoompointdialog.ui</td>
<td>5 KB</td>
<td>Document</td>
</tr>
</tbody>
</table>
Let's see what these various files do:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>.py</td>
<td>This is a standard Python package initialization file. This file also initializes the plugin and makes it available to the QGIS system.</td>
</tr>
<tr>
<td>COPYING</td>
<td>This is a copy of the GNU General Public License (GPL). Since the Zoom to Point plugin is generally available, this defines the license under which it can be used.</td>
</tr>
<tr>
<td>icon.png</td>
<td>As the name suggests, this is the plugin's toolbar icon.</td>
</tr>
<tr>
<td>Makefile</td>
<td>This is a standard *nix Makefile used to automate the process of compiling and deploying the plugin.</td>
</tr>
<tr>
<td>metadata.txt</td>
<td>This file contains the plugin's metadata, including the full name of the plugin, a description, the current version number, and so on.</td>
</tr>
<tr>
<td>resources.qrc</td>
<td>This is a Qt resource file that defines the various resources such as images and sound files used by the plugin.</td>
</tr>
<tr>
<td>resources.py</td>
<td>This indicates the contents of the resources.qrc file, compiled into a Python module.</td>
</tr>
<tr>
<td>ui_zoomtopoint.ui</td>
<td>This is a Qt user interface template that defines the main UI for the plugin.</td>
</tr>
<tr>
<td>ui_zoomtopoint.py</td>
<td>This indicates the contents of the ui_zoomtopoint.ui file compiled into a Python module.</td>
</tr>
<tr>
<td>zoomtopoint.py</td>
<td>This file contains the main Python code for the plugin.</td>
</tr>
<tr>
<td>zoomtopointdialog.ui</td>
<td>This is a copy of the ui_zoomtopoint.ui file. It looks like this file was included by accident, as the plugin can run without it.</td>
</tr>
<tr>
<td>zoomtopointdialog.py</td>
<td>This Python module defines a QtGui.QDialog subclass that loads the dialog box's contents from ui_zoomtopoint.py.</td>
</tr>
</tbody>
</table>

Open the zoomtopoint.py module in your favorite text editor. As you can see, this contains the main Python code for the plugin, in the form of a ZoomToPoint class. This class has the following basic structure:

```python
class ZoomToPoint:
    def __init__(self, iface):
        self.iface = iface
```

[17]
If you open the `__init__.py` module, you'll see how this class is used to define the plugin's behavior:

```python
def classFactory(iface):
    from zoomtopoint import ZoomToPoint
    return ZoomToPoint(iface)
```

When the plugin is loaded, a parameter named `iface` is passed to the `classFactory` function. This parameter is an instance of QgsInterface, and provides access to the various parts of the running QGIS application. As you can see, the class factory creates a `ZoomToPoint` object, and passes the `iface` parameter to the initializer so that `ZoomToPoint` can make use of it.

Notice how `ZoomToPoint.__init__`, in the `zoomtopoint.py` module, stores a reference to the `iface` parameter in an instance variable, so that the other methods can refer to the QGIS interface using `self.iface`. For example:

```python
def __init__(self, iface):
    self.iface = iface

def initGui(self):
    ...
    self.iface.addPluginToMenu("&Zoom to point...", self.action)
```

This allows the plugin to interact with and manipulate the QGIS user interface.

The four methods defined by the `ZoomToPoint` class are all quite straightforward:

- `__init__`: This method initializes a new `ZoomToPoint` object.
- `initGui`: This method initializes the plugin's user interface, preparing it to be used.
- `unload`: This method removes the plugin from the QGIS user interface.
- `run`: This method is called when the plugin is activated, that is, when the user clicks on the plugin's icon in the toolbar, or selects the plugin from the Plugins menu.
Don't worry too much about all the details here; we'll look at the process of initializing and unloading a plugin in a later chapter. For now, take a closer look at the run() method. This method essentially looks like the following:

```python
def run(self):
    dlg = ZoomToPointDialog()
    ...  
    dlg.show()
    result = dlg.exec_()
    if result == 1:
        x = dlg.ui.xCoord.text()
        y = dlg.ui.yCoord.text()
        scale = dlg.ui.spinBoxScale.value()
        rect = QgsRectangle(float(x) - scale,
                          float(y) - scale,
                          float(x) + scale,
                          float(y) + scale)
        mc=self.iface.mapCanvas()
        mc.setExtent(rect)
        mc.refresh()
    ...
```

We've excluded the code that remembers the values the user entered previously, and copies those values back into the dialog when the plugin is run. Looking at the previous code, the logic seems to be fairly straightforward and is explained as follows:

- Create a ZoomToPointDialog object.
- Display the dialog box to the user.
- If the user clicks on the OK button, extract the entered values, use them to create a new bounding rectangle, and set the extent of the map to this rectangle.

While this plugin is quite straightforward and the actual code doesn’t do all that much, it is a useful example of what a Python plugin should look like, as well as the various files that are needed by a Python plugin. In particular, you should note that:

- A plugin is simply a directory that contains a Python package initialization file (__init__.py), some Python modules, and other files created using Qt Designer.
- The __init__.py module must define a top-level function named ClassFactory that accepts an iface parameter and returns an object that represents the plugin.
Getting Started with QGIS

- The plugin object must define an `initGui()` method, which is called to initialize the plugin's user interface, and an `unload()` method, which is called to remove the plugin from the QGIS application.
- The plugin can interact with and manipulate the QGIS application via the `iface` object passed to the class factory.
- The `resources.qrc` file lists various resources such as images, which are used by the plugin.
- The `resources.qrc` file is compiled into a `resources.py` file using the PyQt command-line tools.
- Dialog boxes and other windows are created using a Qt Designer template, which are typically stored in a file with a name of the form `ui_Foo.ui`.
- The UI template files are then compiled into Python code using the PyQt command-line tools. If the template is named `ui_foo.ui`, then the associated Python module will be named `ui_foo.py`.
- Once the user interface for a dialog box has been defined, you create a subclass of `QtGui.QDialog`, and load that user interface module into it. This defines the contents of the dialog box based on your template.
- Your plugin can then display the dialog box as required, extracting the entered values and using the results to interact with QGIS via the `iface` variable.

Plugins are a useful way of extending and customizing QGIS. We will return to the topic of QGIS plugins in Chapter 4, Creating QGIS Plugins, where we will create our own plugin from scratch.

Writing an external application

The final way to work with Python and QGIS is to write a completely standalone Python program that imports the QGIS libraries and works with them directly. In many ways, this is an ideal way of writing your own custom mapping applications, because your program doesn't have to run within the existing QGIS user interface. There are, however, a few things you need to be aware of when you attempt to use Python and QGIS in this way:

1. Your Python program needs to be able to find the QGIS Python libraries before it can be run. Since these are bundled into the QGIS application itself, you will need to add the directory where the PyQGIS libraries are installed in your Python path.
2. You also need to tell the PyQGIS libraries where the QGIS application's resources are stored.
3. As the application is running outside the QGIS application, you won't have access to the\texttt{iface} variable. You also can't use those parts of the PyQGIS library that assume you are running inside QGIS.

None of this is too onerous, though it can trip you up the first time you attempt to access PyQGIS from your external Python code. Let's take a look at how we can avoid these traps when writing your own Python programs.

Firstly, to allow your program to access the PyQGIS libraries, you need to modify your Python path (and possibly some other environment variables) before you can import any of the QGIS packages. For MS Windows, you can do this by running the following in the command line:

\begin{verbatim}
SET OSGEO4W_ROOT=C:\OSGeo4W
SET QGIS_PREFIX=%OSGEO4W_ROOT%\apps\qgis
SET PATH=%PATH%;%QGIS_PREFIX%\bin
SET PYTHONPATH=%QGIS_PREFIX%\python;%PYTHONPATH%
\end{verbatim}

If you are running Mac OS X, the following commands will set up the Python path for you:

\begin{verbatim}
export PYTHONPATH="$PYTHONPATH:/Applications/QGIS.app/Contents/Resources/python"
export DYLD_FRAMEWORK_PATH="/Applications/QGIS.app/Contents/Frameworks"
export QGIS_PREFIX="/Applications/QGIS.app/Contents/Resources"
\end{verbatim}

For computers that run a version of Linux, you can use the following:

\begin{verbatim}
export PYTHONPATH="/path/to/qgis/build/output/python/"
export LD_LIBRARY_PATH="/path/to/qgis/build/output/lib/"
export QGIS_PREFIX="/path/to/qgis/build/output/"
\end{verbatim}

Obviously, you will need to replace \texttt{/path/to/qgis} with the actual path of your QGIS installation.

If you have QGIS installed in a nonstandard location, you might need to modify these commands before they will work. To check if they have worked, start up the Python interpreter and enter the following command:

\begin{verbatim}
>>> import qgis
\end{verbatim}
If everything goes well, you'll simply see the Python prompt:

```
>>> 
```

On the other hand, you might see the following error:

**ImportError: No module named qgis**

In this case, the `PYTHONPATH` variable has not been set up correctly, and you will have to check the commands you entered earlier to set this environment variable, and possibly modify it to allow for a nonstandard location of the QGIS libraries.

Note that in some cases, this isn't enough because the Python libraries are only wrappers around the underlying C++ libraries; you might also need to tell your computer where to find these C++ libraries. To see if this is a problem, you can try to do the following:

```python
import qgis.core
```

You might get an error that looks like this:

**ImportError: libqgis_core.so.1.5.0: cannot open shared object file: No such file or directory**

You will have to tell your computer where to find the underlying shared libraries. We will return to this later when we look at writing our own external applications; if you want to see the details, skip ahead to *Chapter 5, Using QGIS in an External Application.*

With the path set, you can now import the various parts of the PyQGIS library that you want to use, for example:

```python
from qgis.core import *
```

Now that we have access to the PyQGIS libraries, our next task is to initialize these libraries. As mentioned earlier, we have to tell PyQGIS where to find the various QGIS resources. We do this using the `QgsApplication.setPrefixPath()` function, like this:

```python
import os
QgsApplication.setPrefixPath(os.environ['QGIS_PREFIX'], True)
```

This uses the `QGIS_PREFIX` environment variable we set earlier to tell QGIS where to find its resources. With this done, you can then initialize the PyQGIS library by making the following call:

```python
QgsApplication.initQgis()
```
We can now use PyQGIS to do whatever we want in our application. When our program exits, we also need to inform the PyQGIS library that we are exiting:

```python
QgsApplication.exitQgis()
```

Putting all this together, our minimal Python application looks like this:

```python
import os
from qgis.core import *

QgsApplication.setPrefixPath(os.environ['QGIS_PREFIX'], True)
QgsApplication.initQgis()

# ...

QgsApplication.exitQgis()
```

Of course, this application doesn't do anything useful yet—it simply starts up and shuts down the PyQGIS libraries. So let's replace the "..." line with some useful code that displays a basic map widget. To do this, we need to define a QMainWindow subclass, which displays the map widget, and then create and use a QApplication object to display this window and handle the various user-interface events while the application is running.

Both QMainWindow and QApplication are PyQt classes. We will be working extensively with the various PyQt classes as we develop our own external applications using QGIS and Python.

Let's start by replacing the "..." line with the following code, which displays a map viewer and then runs the application's main event loop:

```python
app = QApplication(sys.argv)
viewer = MapViewer("/path/to/shapefile.shp")
viewer.show()
app.exec_()
```

As you can see, a MapViewer instance (which we will define shortly) is created and displayed, and the QApplication object is run by calling the exec_() method. For simplicity, we pass the name of a shapefile to display within the map viewer.
Running this code will cause the map viewer to be displayed, and the application will run until the user closes the window or chooses the **Quit** command from the menu.

Now, let's define the `MapViewer` class. Here is what the class definition looks like:

```python
class MapViewer(QMainWindow):
    def __init__(self, shapefile):
        QMainWindow.__init__(self)
        self.setWindowTitle("Map Viewer")

        canvas = QgsMapCanvas()
        canvas.useImageToRender(False)
        canvas.setCanvasColor(Qt.white)
        canvas.show()

        layer = QgsVectorLayer(shapefile, "layer1", "ogr")
        if not layer.isValid():
            raise IOError("Invalid shapefile")

        QgsMapLayerRegistry.instance().addMapLayer(layer)
        canvas.setExtent(layer.extent())
        canvas.setLayerSet([QgsMapCanvasLayer(layer)])

        layout = QVBoxLayout()
        layout.addWidget(canvas)

        contents = QWidget()
        contents.setLayout(layout)
        self.setCentralWidget(contents)
```

Don't worry too much about the details of this class; we basically just create a window and place a `QgsMapCanvas` object within it. We then create a map layer (an instance of `QgsVectorLayer`) and add it to the map canvas. Finally, we add the canvas to the window's contents.

Notice that `QgsMapCanvas` and `QgsVectorLayer` are both part of PyQGIS, while `QMainWindow`, `QVBoxLayout`, and `QWidget` are all PyQt classes. This application uses the PyQGIS classes within a PyQt application, mixing the classes from both sources. This is possible because QGIS is built using Qt, and the various PyQGIS classes are based on PyQt.
To turn the preceding code into a working application, all we need to do is add some more `import` statements to the top of the module:

```python
import sys
from PyQt4.QtGui import *
from PyQt4.QtCore import Qt
```

### Downloading the example code

You can download the example code files from your account at [http://www.packtpub.com](http://www.packtpub.com) for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.

If you run this application, the map viewer will be displayed, showing the contents of the shapefile referred to by the code. For example:

![Map Viewer](image)

This application is still a bit ugly—you can see white space at the top and bottom this map because it doesn't take into account the aspect ratio of the map data. There's also no feature of zooming in or scrolling around the map. However, these can be added quite easily, and as you can see, it's not very difficult to create your own standalone mapping applications built on top of QGIS.
Summary

In this chapter, we became familiar with QGIS and the various ways in which it can be used as a Python geospatial development system. We installed and explored the QGIS application itself, and then looked at how Python can be used with QGIS. We saw how QGIS uses data sources, map layers, maps, and projects to organize and work with geospatial data. Next, we examined the three ways in which you can use Python and QGIS: by typing commands into the Python Console, by writing a Python plugin or by writing an external application that makes use of the QGIS Python API.

We then looked at the extensive set of Python libraries that come with QGIS, called PyQGIS, which you can use for geospatial development. We saw how to use the QGIS Python Console to directly manipulate the QGIS project, add layers, zoom in and out, change options, and so on.

Next up, we downloaded and examined a QGIS Python plugin. In doing this, we learned that QGIS plugins are simply Python packages installed in a hidden directory named .qgis2 (or .qgis) within your home or user directory. A plugin makes use of the Qt library to define and build resources such as user interface templates.

Finally, we saw how we can write external Python applications that load the PyQGIS libraries from within the QGIS system, and then use those libraries within a larger PyQt application.

In the next chapter, we will explore the QGIS Python Console in more detail, and use it to become more familiar with the PyQGIS library, and also see how we can use it within our own Python geospatial development projects.
Where to buy this book

You can buy Building Mapping Applications with QGIS from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.