Extending the Functionality of QGIS with Python Plugins Workshop FOSS4G 2008

Dr. Marco Hugentobler Dr. Horst Düster Tim Sutton

September 2008

1 Introduction

QGIS is a popular desktop GIS written in C++. Besides the possibility of writing extensions in C++, there is the python interface to QGIS (PyQGIS). There are three ways to use PyQGIS:

- From a python command line console which is mainly interesting for debugging.
- As QGIS plugins written in Python. Those plugins may be enabled in the plugin manager just like C++ plugins.
- Standalone applications in Python with their own user interfaces may use the functionality of the QGIS core library.

This workshop provides a beginner's tutorial for writing QGIS Python plugins.

2 Why Python?

Python is a scripting language which was designed with the goal of being easy to program. It has a mechanism that automatically releases memory that is no longer used (garbagge collector). A further advantage is that many programs that are written in C++ or Java offer the possibility to write extensions in Python, e.g. OpenOffice or Gimp. Therefore it is a good investment of time to learn the Python language.

3 License

PyQGIS plugins use functionality of libqgis_core.so and libqgis_gui.so. As both are licensed undere GPL, QGIS Python plugins must be licenced under the GPL too. This means you may use your plugins for any purpose and you are not forced to publish them. If you do publish them however, they must be published under the conditions of the GPL license. With Python programs, this restriction is not as important as for compiled programs, because the source code is visible anyway.

4 What needs to be installed to get started

On the lab computers, everything for the workshop is already installed. If you program Python plugins at home, you will need the following libraries and programs:

- QGIS
- Python
- Qt
- PyQT
- PyQt development tools

If you use Linux, there are binary packages for all major distributions. For Windows, the PyQt installer already contains Qt, PyQt and the PyQt development tools.

5 A PyQGIS Example Plugin in three Steps

Our example plugin is intentionally kept simple. It adds a button to the menu bar of QGIS. If the button is clicked, a file dialog appears where the user may load a shape file.

For each python plugin, a dedicated folder that contains the plugin files needs to be created. By default, QGIS looks for plugins in $QGIS_DIR/share/qgis/python/plugins (in our workshop /usr/share/qgis/python/plugins). On Linux, there is also the possibility to have plugins in $HOME/.qgis/python/plugins such that it is only visible for one user.

5.1 Step 1: Make the plugin manager recognise the plugin

To provide the neccessary information for QGIS, the plugin needs to implement the methods 'name()', 'description()' and 'version()' which return descriptive strings. A plugin also needs a method 'classFactory(QgisInterface)' which is called by the plugin manager to create an instance of the plugin. The argument of type QGisInterface is used by the plugin to access functions of the QGIS instance. We are going to work with this object in step 2.

Note that, in contrast to other programing languages, indention is very important. The Python interpreter throws an error if it is not correct.

First we create the plugin folder 'foss4g_plugin' in QGIS/python/plugins. Then we add two new textfiles into this folder, 'foss4gplugin.py' and '__init__.py'.

- The file foss4gplugin.py contains the plugin class:

```
# -*- coding: utf-8 -*-
# Import the PyQt and QGIS libraries
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
# Initialize Qt resources from file resources.py
import resources
```

```
class FOSS4GPlugin:

    def __init__(self, iface):
# Save reference to the QGIS interface
        self.iface = iface

    def initGui(self):
        print 'Initialising GUI'

    def unload(self):
        print 'Unloading plugin'
```

- The file __init__.py contains the methods 'name', 'description', 'version' and 'classFactor'y. As we are creating a new instance of the plugin class, we need to import the code of this class:

```
# -*- coding: utf-8 -*-
from foss4gplugin import FOSS4GPlugin
def name():
    return "FOSS4G example"
def description():
    return "A simple example plugin to load shapefiles"
def version():
    return "Version 0.1"
def classFactory(iface):
    return FOSS4GPlugin(iface)
```

Now the plugin has the neccessary infrastructure to appear in the QGIS plugin manager and be loaded / unloaded.

5.2 Step2: Add a button and a menu

5.2.1 Icon

To make the icon graphic available for our program, we need a so-called resource file. In the resource file, the graphic is contained in hexadecimal notation. Fortunately, we don't need to care about its representation because we use the pyrcc compiler, a tool that reads the file 'resources.qrc' and creates a resource file.

The file 'foss4g.png' and the resource file can be downloaded from http://karlinapp.ethz.ch/python_foss4g. Move these files into the directory of the example plugin. Open a shell, cd to the plugin directory and enter: <path_to_QGIS_folder>/pyrcc4 -o resources.py resources.qrc.

5.2.2 Add a menu and a button

In this section, we implement the content of the methods 'initGui()' and 'unload()'. We need an instance of the class 'QAction' that executes the 'run()' method of the plugin. With the action object, we are then able to generate the menu entry and the button:

```
import resources

  def initGui(self):
    # Create action that will start plugin configuration
    self.action = QAction(QIcon(":/plugins/foss4g_plugin/foss4g.png"), "FOSS4G plugin",
self.iface.getMainWindow())
    # connect the action to the run method
    QObject.connect(self.action, SIGNAL("activated()"), self.run)

      # Add toolbar button and menu item
    self.iface.addToolBarIcon(self.action)
    self.iface.addPluginMenu("FOSS-GIS plugin...", self.action)

      def unload(self):
    # Remove the plugin menu item and icon
    self.iface.removePluginMenu("FOSSGIS Plugin...", self.action)
    self.iface.removeToolBarIcon(self.action)
```

5.3 Step3: Load a layer from a shape fileSchritt3

In this step we implement the real functionality of the plugin in the 'run()' method. The Qt4 method 'QFileDialog::getOpenFileName' opens a file dialog and returns the path to the chosen file. If the user cancels the dialog, the path is a null object, which we test for. We then call the method 'addVectorLayer' of the interface object which loads the layer. The method only needs three arguments: the file path, the name of the layer that will be shown in the legend and the data provider name. For shapefiles, this is 'ogr' because QGIS internally uses the OGR library to access shapefiles:

```
      def run(self):
    fileName = QFileDialog.getOpenFileName(None,QString.fromLocal8Bit("Select a file:"),
"", "*.shp *.gml")
    if fileName.isNull():
      QMessageBox.information(None, "Cancel", "File selection canceled")       else:
      vlayer = self.iface.addVectorLayer(fileName, "myLayer", "ogr")
```

6 Further information

As you can see, you need information from different sources to write PyQGIS plugins. Plugin writers need to know Python and the QGIS plugin interface as well as the Qt4 classes and tools. At the beginning it is best to learn from examples and copy the mechanism of existing plugins. Using the QGIS plugin installer, which itself is a Python plugin, it is possible to download a lot of existing Python plugins and to study their behaviour.

We finish the tutorial with a selection of online documentation that may be usefull for PyQGIS programers:

- QGIS wiki: http://wiki.qgis.org/qgiswiki/PythonBindings
- QGIS API documentation: http://doc.qgis.org/index.html
- Qt documentation: http://doc.trolltech.com/4.3/index.html
- PyQt: http://www.riverbankcomputing.co.uk/pyqt/
- Python tutorial: http://docs.python.org/
- A book about desktop GIS and QGIS. It contains a chapter about PyQGIS plugin programing: http://www.pragprog.com/titles/gsdgis/desktop-gis