Создание приложения на базе набора библиотек QGIS на Python

Обсудить в форуме Комментариев — 123

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/qgis-standalone.html

Подробное описание процесса создания небольшого приложения на базе QGIS с помощью Python.

Данная статья объясняет процесс создания самостоятельного отдельного приложения на основе QGIS. Одной из целей QGIS, помимо предоставления пользовательского приложения ГИС является также поддержка набора библиотек, которые могут быть использованы для создания новых приложений. Начиная с версии 0.9 разработка может вестись на языках C++ и Python. Мы рекомендуем использовать QGIS 1.0.0 или более новую версию в качестве основы для ваших приложений так как именно с этой версии предоставляется стабильный API.

Статья основана на переводе соответствующей главы руководства QGIS Coding and Compilation Guide (Глава 4. Creating PyQGIS Applications) и обучающих примерах, разработанных Tim Sutton для C++ и портированных на Python Martin Dobias. В блоге QGIS можно найти несколько примеров приложений PyQGIS. В статье более подробно с иллюстрациями разъяснены некоторые моменты процесса. Пример, приведенный в руководстве, ориентирован на опытных пользователей, нуждается в основательной доработке из-за специфической формы предоставления кода (нумерация линий и смещение отступов) и его сложно правильно воспроизвести не имея навыков работы с Python.

Статья проверена на QGIS 1.5, в новых версиях QGIS могут быть внесены изменения делающие данный пример неработоспособным.

В качестве примера рассмотрим приложение, которое должно:

- загружать векторный слой
- обеспечивать возможность сдвига, увеличения/уменьшения
- показывать слой полностью
- менять цветовую легенду слоя

Это минимальный набор возможностей, который при необходимости можно расширить.

Содержание

Подготовка

Будем исходить из того, что все программное обеспечение установлено при помощи установщика OSGeo4W (подробнее) используя путь по умолчнанию C:\OSGeo4W. Кроме самого QGIS со всеми зависимостями, необходимо выбрать следующие пакеты в категории Libs:

```
pyqt4, qt4-devel, qt4-doc, qt4-libs
```

В этих пакетах находятся инструменты разработчика, необходимые библиотеки и документация.

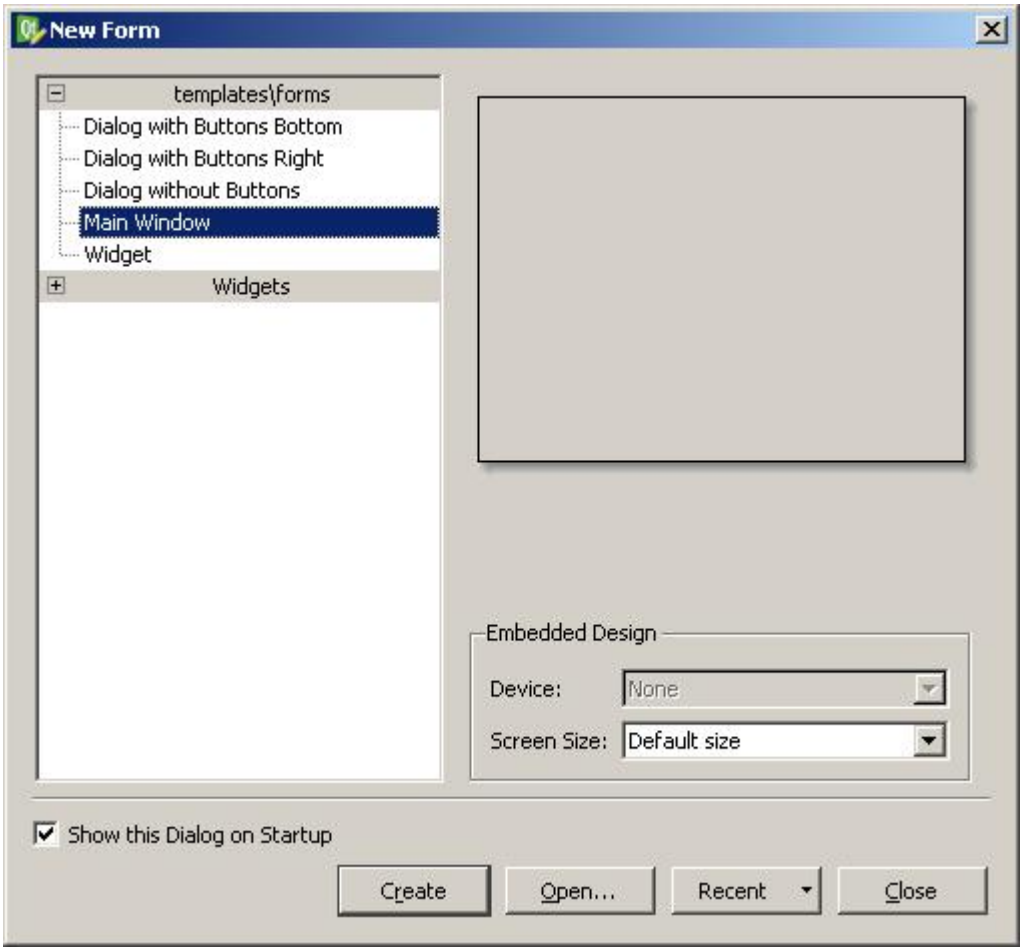
Разработка графического интерфейса

Создание основного окна

Интерфейс для нашего приложения, как и сам QGIS разрабатывается с помощью Qt. В состав Qt входит программа Qt Designer, которая позволяет создавать и изменять формы в режиме WYSIWYG, без редактирования кода. Формы и диалоги, созданные таким образом можно затем сконвертировать в код на Python.

Т.к. мы создаем минималистичное приложение, то и интерфейс у нас будет простой. Используя Qt Designer мы создадим простую форму, которая будет отправной точкой для дальнейшего усовершенствования. Для создания новой формы:

1. создадим папку для нашего приложения
2. запустим Qt Designer (по умолчанию, C:\OSGeo4W\bin\designer.exe)
3. должно появиться окно мастера «New Form», если мастер не запустился, вызовем его из меню «File → New Form...»

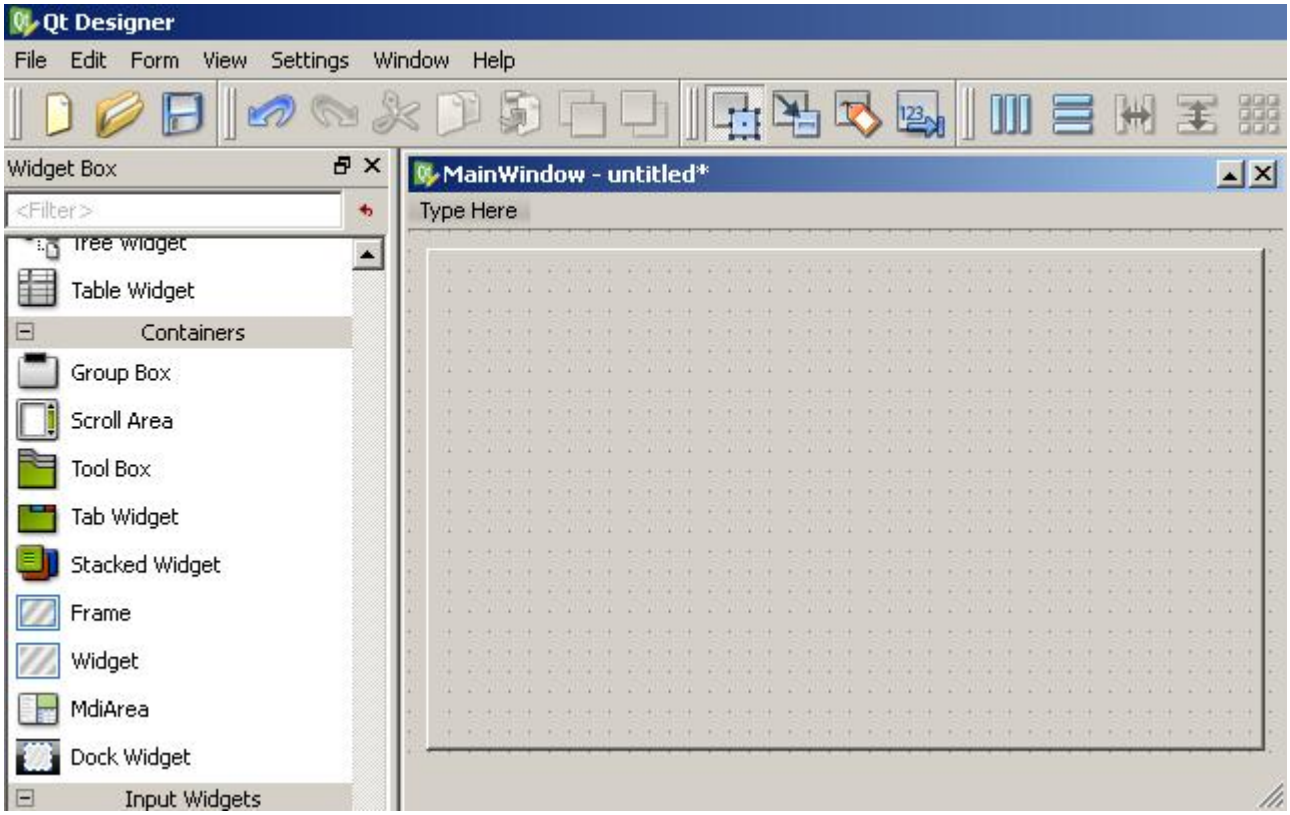


4. из списка «templates/forms» выберем «Main Window» и нажмем на «Create»
5. изменим размеры созданного окна так, чтобы удобно было работать

Найдем в списке элементов (Widget Box) виджет «Frame» находящийся в группе «Containers» и перетащим его в наше окно. После этого щелкнем мышью где-либо на самой форме, но вне контейнера который мы только что добавили (т.е. выделим наше главное окно). Нажмем на кнопку «Lay Out in a Grid», эта операция увеличит

размер добавленного контейнера до размеров формы в целом. Это также свяжет их размеры, так что при изменении размера формы будет меняться и размер контейнера.

Должно получиться что-то похожее на это

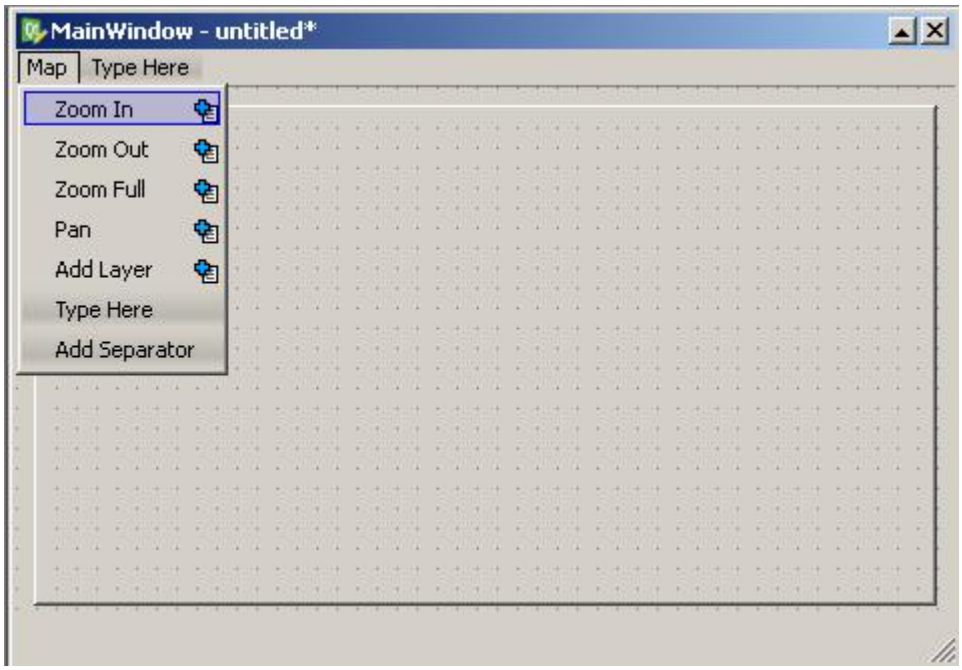


Сохраним созданную форму как mainwindow.ui в папку, созданную на первом шаге.

Добавление меню

Как вы уже заметили, на форме присутствует строка меню и строка состояния.

Создадим меню. Для этого выделяем надпись «Type Here» и вводим название пункта, например «Map», ввод завершаем нажатием на Enter. Появится первый пункт меню, теперь последовательно добавляем команды, нажимая на «Type Here» но уже в меню. Создадим 5 пунктов, как это показано на иллюстрации.



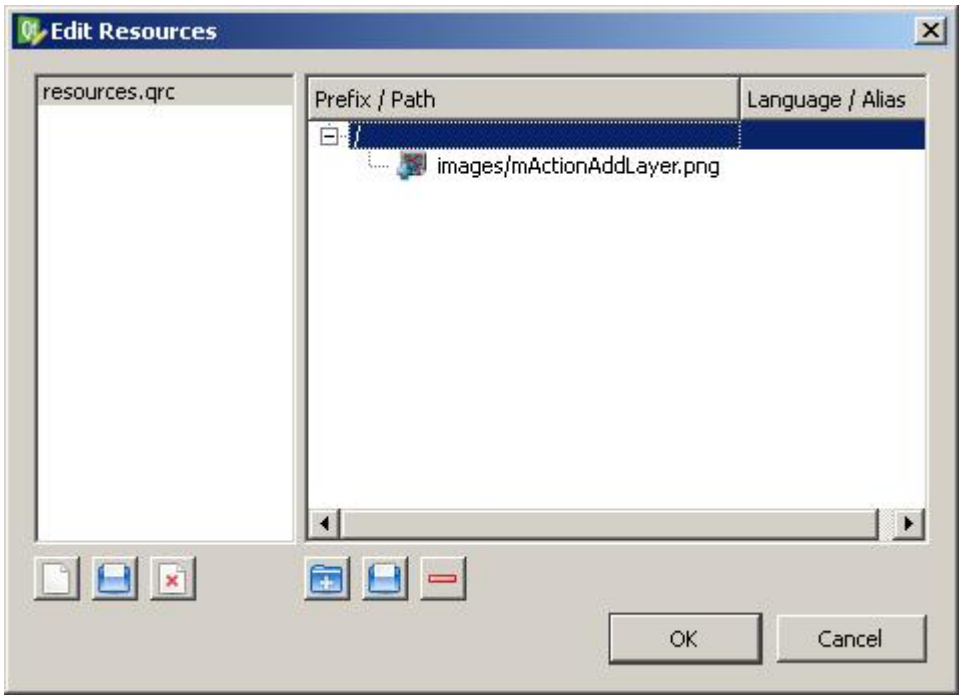
Создание файла ресурсов

Для каждого пункта меню нам нужна иконка, эта же иконка будет отображаться на панели инструментов. Т.к. приложение может быть установлено в разные каталоги, то иконки удобно хранить в ресурсном файле. Файл ресурсов это платформо-независимый способ хранения необходимых приложению данных, таких как иконки, файлы локализации, звуки и т.д., внутри самого приложения или в одном большом файле.

Создадим для иконок папку images в каталоге нашего приложения, сами иконки возьмем из папки themes\default установочной папки QGIS (по умолчанию C:\OSGeo4W\apps\qgis-dev\themes\default\).

Все готово для создания файла ресурсов. Откроем диалог «Resource Editor» (Редактор ресурсов) в Qt Designer, нажмем кнопку «Edit Resources», а затем кнопку «New Resource File». Создадим ресурсный файл с названием resources.qrc в той же директории, где находится mainwindow.ui.

Создадим новый префикс, нажав кнопку «Add Prefix». В качестве имени указываем «/» (без кавычек). Теперь добавим сюда наши 5 иконок для инструментов.



Если мы откроем файл resources.qrc в текстовом редакторе, то увидим следующее:

```
<RCC>
    <qresource prefix="/" >
        <file>images/mActionAddLayer.png</file>
        <file>images/mActionPan.png</file>
        <file>images/mActionZoomFullExtent.png</file>
```

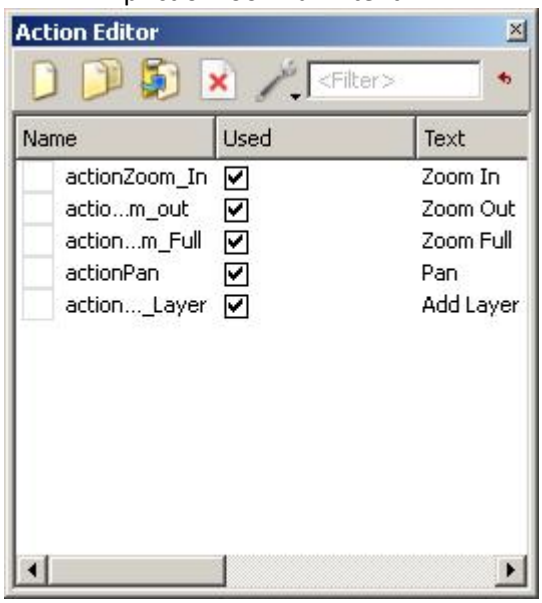
```
        <file>images/mActionZoomIn.png</file>
        <file>images/mActionZoomOut.png</file>
    </qresource>
</RCC>
```

Назначение действий

При создании меню Qt Designer для каждой команды автоматически создал так называемое «действие» (action) и дал ему название.

В нашем случае действий не так много, и можно все оставить как есть, но в больших проектах лучше давать действиям свои названия. Сделать это можно при помощи диалога «Action Editor» (Редактор команд). Переименуем команды, соответствующие каждому пункту меню. Так, назовем команду для пункта Zoom In: mpActionZoomIn, Zoom Out: mpActionZoomOut и т.д. В итоге, у нас должен получиться список из 5 команд:

- mpActionZoomIn
- mpActionZoomOut
- mpActionPan
- mpActionAddLayer
- mpActionZoomFullExtent



Назначим каждому действию соответствующую иконку, для этого дважды щелкнем на действии и выберем иконку из созданного на предыдущем этапе ресурсного файла. Пустые квадраты, обозначающие действия должны замениться на иконки:

Сохраним изменения и закроем Qt Designer.

Компиляция формы и ресурсного файла

Компиляция формы

Скомпилируем созданную ранее форму используя компилятор интерфейсов PyQt. Откроем командную строку OSGeo4W, перейдем в папку нашего приложения и дадим команду

```
pyuic4 -o mainwindow_ui.py mainwindow.ui
```

Результатом компиляции будет представление только что созданной формы на языке Python, следующего содержания:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'mainwindow.ui'
#
# Created: Sun Jun 22 18:37:53 2008
#      by: PyQt4 UI code generator 4.3.3
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")

MainWindow.resize(QtCore.QSize(QtCore.QRect(0,0,579,330).size()).expandedTo(MainWindow.
minimumSizeHint()))

        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.gridlayout = QtGui.QGridLayout(self.centralwidget)
        self.gridlayout.setMargin(9)
        self.gridlayout.setSpacing(6)
        self.gridlayout.setObjectName("gridlayout")

        self.frameMap = QtGui.QFrame(self.centralwidget)
        self.frameMap.setFrameShape(QtGui.QFrame.StyledPanel)
        self.frameMap.setFrameShadow(QtGui.QFrame.Raised)
        self.frameMap.setObjectName("frameMap")
        self.gridlayout.addWidget(self.frameMap,0,0,1,1)
        MainWindow.setCentralWidget(self.centralwidget)

        self.menubar = QtGui.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0,0,579,22))
        self.menubar.setObjectName("menubar")

        self.menuMap = QtGui.QMenu(self.menubar)
        self.menuMap.setObjectName("menuMap")
        MainWindow.setMenuBar(self.menubar)

        self.statusbar = QtGui.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.mpActionZoomIn = QtGui.QAction(MainWindow)
        self.mpActionZoomIn.setIcon(QtGui.QIcon(":/mActionZoomIn.png"))
        self.mpActionZoomIn.setObjectName("mpActionZoomIn")

        self.mpActionZoomOut = QtGui.QAction(MainWindow)
        self.mpActionZoomOut.setIcon(QtGui.QIcon(":/mActionZoomOut.png"))
        self.mpActionZoomOut.setObjectName("mpActionZoomOut")
```

```
        self.mpActionPan = QtGui.QAction(MainWindow)
        self.mpActionPan.setIcon(QtGui.QIcon(":/mActionPan.png"))
        self.mpActionPan.setObjectName("mpActionPan")

        self.mpActionAddLayer = QtGui.QAction(MainWindow)
        self.mpActionAddLayer.setIcon(QtGui.QIcon(":/mActionAddLayer.png"))
        self.mpActionAddLayer.setObjectName("mpActionAddLayer")
        self.menuMap.addAction(self.mpActionZoomIn)
        self.menuMap.addAction(self.mpActionZoomOut)
        self.menuMap.addAction(self.mpActionPan)
        self.menuMap.addAction(self.mpActionAddLayer)
        self.menubar.addAction(self.menuMap.menuAction())

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow",
"MainWindow", None, QtGui.QApplication.UnicodeUTF8))
        self.menuMap.setTitle(QtGui.QApplication.translate("MainWindow", "Map", None,
QtGui.QApplication.UnicodeUTF8))
        self.mpActionZoomIn.setText(QtGui.QApplication.translate("MainWindow", "Zoom
In", None, QtGui.QApplication.UnicodeUTF8))
        self.mpActionZoomOut.setText(QtGui.QApplication.translate("MainWindow", "Zoom
Out", None, QtGui.QApplication.UnicodeUTF8))
        self.mpActionPan.setText(QtGui.QApplication.translate("MainWindow", "Pan",
None, QtGui.QApplication.UnicodeUTF8))
        self.mpActionAddLayer.setText(QtGui.QApplication.translate("MainWindow", "Add
Layer", None, QtGui.QApplication.UnicodeUTF8))

import resources_rc
```

Компиляция ресурсного файла

Как можно видеть из последней строки скопилированного файла формы (mainwindow_ui.py) туда автоматически была включена строка

```
import resources_rc
```

однако такого файла еще не существует и его необходимо создать, скомпилировав файл ресурсов:

```
pyrcc4 -o resources_rc.py resources.qrc
```

Создание приложения

Форма создана. Теперь нам нужно создать код приложения, который и будет выполнять реальную работу. Создать сам файл программы можно в любом редакторе (Notepad и т.п.) или в PythonWin. Разберем код приложения по частям. Скачать полный код можно здесь.

В самой первой строке укажем кодировку исходного текста. Хорошим тоном считается упоминание авторов исходного кода и всех, кто внес изменения

```
# -*- coding: utf-8 -*-
# Пример основан на C++ Tutorial (C) 2006 Tim Sutton
# Портирование на Python: Martin Dobias, при участии Gary Sherman, изменения для
FOSS4G2007
# Адаптация, перевод, изменения sim@gis-lab.info
# Лицензия GNU GPL 2
```

Импортируем модули Qt, QGIS и созданную нами форму.

```
from PyQt4 import QtCore, QtGui
from qgis.core import *
from qgis.gui import *
import sys,os
```

```
# Импорт созданного интерфейса
from mainwindow_ui import Ui_MainWindow
```

Приложению необходимо знать путь к папке установки QGIS

```
qgis_prefix = os.getenv( "QGISHOME" )
```

Создадим новый класс MainWindow, который будет описывать наше приложение:

```
class MainWindow( QtGui.QMainWindow, Ui_MainWindow ):
  def __init__( self ):
    QtGui.QMainWindow.__init__( self )

    # Требуется Qt4 для инициализации пользовательского интерфейса
    self.setupUi( self )
```

Установим заголовок окна, создадим карту и зададим светло-голубой цвет фона:

```
    # Устанавливаем заголовок окна
    self.setWindowTitle( "QGIS Demo" )

    # Создаем карту
    self.canvas = QgsMapCanvas()
    # Задаем цвет фона карты
    self.canvas.setCanvasColor( QtGui.QColor( 200, 200, 255 ) )
    self.canvas.enableAntiAliasing( True )
    self.canvas.show()
```

Расположим компоненты-виджеты в окне приложения, окно карты так же является одним из компонентов:

```
    self.layout = QtGui.QVBoxLayout( self.frame )
    self.layout.addWidget( self.canvas )
```

Далее необходимо задать действия для каждого из инструментов и соединить их с соответствующими методами, которые будут определены чуть позднее в этом же классе:

```
    self.connect( self.mpActionAddLayer, QtCore.SIGNAL( "triggered()" ), self.addLayer
);
    self.connect( self.mpActionZoomIn, QtCore.SIGNAL( "triggered()" ), self.zoomIn );
    self.connect( self.mpActionZoomOut, QtCore.SIGNAL( "triggered()" ), self.zoomOut );
    self.connect( self.mpActionPan, QtCore.SIGNAL( "triggered()" ), self.pan );
    self.connect( self.mpActionZoomFullExtent, QtCore.SIGNAL( "triggered()" ),
self.zoomFull );
```

Создадим панель инструментов «Map» на которую добавим наши действия:

```
    self.toolbar = self.addToolBar( "Map" );
    self.toolbar.addAction( self.mpActionAddLayer );
    self.toolbar.addAction( self.mpActionZoomIn );
    self.toolbar.addAction( self.mpActionZoomOut );
    self.toolbar.addAction( self.mpActionPan );
    self.toolbar.addAction( self.mpActionZoomFullExtent );
```

Создадим инструменты:

```
    self.toolPan = QgsMapToolPan( self.canvas )
    self.toolPan.setAction( self.mpActionPan )
    self.toolZoomIn = QgsMapToolZoom( self.canvas, False ) # false = уменьшить
    self.toolZoomIn.setAction( self.mpActionZoomIn )
    self.toolZoomOut = QgsMapToolZoom( self.canvas, True ) # true = увеличить
    self.toolZoomOut.setAction( self.mpActionZoomOut )
```

И соответствующие им процедуры, для обычных инструментов они будут выглядеть вот так:

```
  def zoomIn( self ):
    self.canvas.setMapTool( self.toolZoomIn )

  def zoomOut( self ):
    self.canvas.setMapTool( self.toolZoomOut )

  def pan( self ):
    self.canvas.setMapTool( self.toolPan )

  def zoomFull( self ):
    self.canvas.zoomFullExtent()
```

Для инструмента добавляющего слой, описание будет более развернутым. Необходимо определить какой слой будет загружен и откуда, а так же сделать так, чтобы охват карты был равен охвату загруженного слоя, чтобы показать его полностью. Сами данные — файл test.shp можно загрузить вместе с исходным кодом и вспомогательными материалами, располагаться они должны в той же папке, откуда будет запускаться программа.

```
  def addLayer( self ):
    layerPath = "test.shp"
    layerName = "test"
    layerProvider = "ogr"

    layer = QgsVectorLayer( layerPath, layerName, layerProvider )

    if not layer.isValid():
      return

    QgsMapLayerRegistry.instance().addMapLayer( layer );

    self.canvas.setExtent( layer.extent() )

    cl = QgsMapCanvasLayer( layer )
    layers = [ cl ]
    self.canvas.setLayerSet( layers )
```

Далее, создадим процедуру инициализирующую библиотеки QGIS и запускающую нашу программу:

```
def main( app ):
  QgsApplication.setPrefixPath( qgis_prefix, True )
  QgsApplication.initQgis()

  wnd = MainWindow()
  wnd.show()

  retval = app.exec_()

  QgsApplication.exitQgis()
  sys.exit( retval )
```

И запустим ее на выполнение:

```
if __name__ == "__main__":
  app = QtGui.QApplication( sys.argv )

  main( app )
```

После создания нашей программы, остается только ее запустить. Но перед запуском необходимо установить несколько переменных окружения.

Переменные окружения в Windows XP доступны через: «Пуск → Настройки → Панель управления → Система → Дополнительно → Переменные среды» («Start → Settings → Control Panel → System → Advanced → Environmental Variables»).

Создадим новую переменную QGISHOME. указывающую на каталог с установленным QGIS. Основываясь на значении этой переменной, наше приложении будет искать необходимые ему бибилиотеки, в частности, библиотеки провайдеров данных, позволяющие работать с разными источниками данных

```
QGISHOME=c:\OSGeo4W\apps\qgis-dev
```

Переменная PYTHONPATH должна содержать путь к файлам QGIS Python API. Обратите внимание, эта переменная указывает на папку python внутри каталога QGIS, а не на папку, в которую установлен Python.

```
PYTHONPATH=c:\OSGeo4W\apps\qgis-dev\python
```

К переменной PATH надо добавить путь к каталогу bin папки QGIS

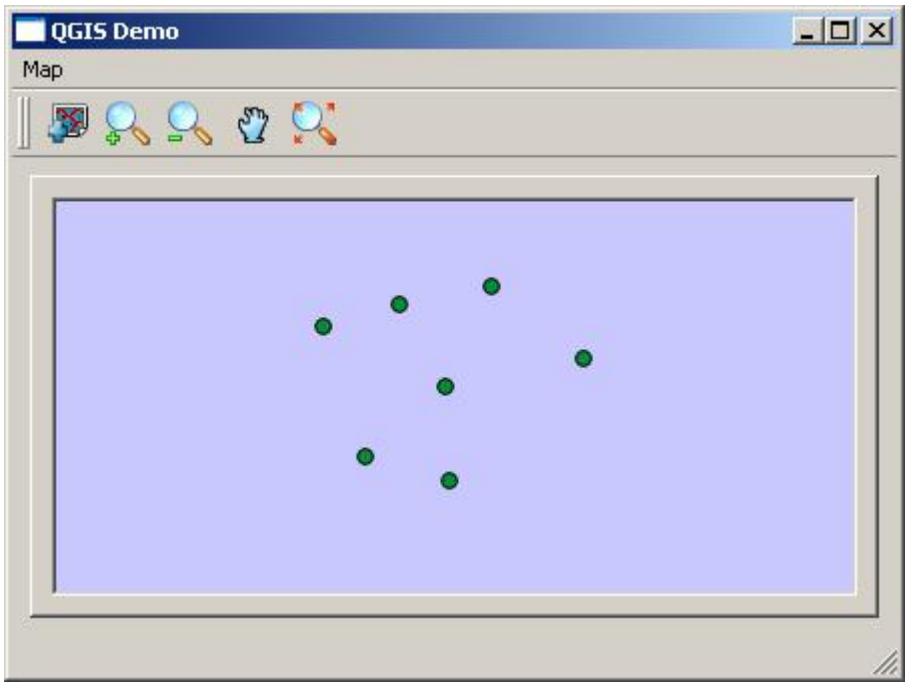
```
PATH=c:\OSGeo4W\apps\qgis-dev\bin;%PATH%
```

Чтобы не устанавливать эти переменные при каждом запуске приложения, и чтобы избежать возможных конфликтов можно создать простой командный файл примерно такого содержания (предполагается, что QGIS установлен в папку по умолчанию):

```
set QGISHOME=c:\OSGeo4W\apps\qgis-dev
set PYTHONPATH=c:\OSGeo4W\apps\qgis-dev\python
set PATH=c:\OSGeo4W\apps\qgis-dev\bin;%PATH%

start python d:\devel\simple_main_window\mainwindow.py
```

Исправьте пути на свои и сохраните файл с расширением .bat. Теперь можно запускать приложение. После запуска появится вот такое окно:



Как видите, в создании приложений PyQGIS нет ничего сложного. Написав меньше чем 150 строк кода, мы получили приложение, умеющее загружать шейп-файлы и перемещаться по карте. Если вы поработаете с программой чуть дольше, то увидите, что некоторые особенности QGIS работают и здесь, например, масштабирование при врашении колесика мыши, перетаскивание при зажатом пробеле и др.

Исходные коды примера, а так же необходимые изображения, ресурсный файл и тестовый набор данных можно скачать одним файлом.

Ссылки по теме

- QGIS tutorials in Python
- Добавление списка слоёв в приложение PyQGIS
- Документация по QGIS

Обсудить в форуме Комментариев — 123

Последнее обновление: 2014-05-15 00:20

Дата создания: 16.08.2008
Автор(ы): Максим Дубинин, Александр Бруй