QGIS - CREATING PLUGINS TUTORIAL

http://www.qgistutorials.com/en/docs/building_a_python_plugin.html
http://www.digital-geography.com/build-qgis-plugin/#.VlHghHYvdD8
repositories - http://plugins.qgis.org/

possible approaches to start:

- rewriting hello world
- using generator
- using plugin builder

1. HELLO WORLD

1. download https://plugins.qgis.org/plugins/HelloWorld/version/2.0/
2. move to Users/.../.qgis2/python/plugins
3. open QGIS
4. activate plugin in plugins tab

- **REWRITING HELLO WORLD**

1. open HelloWorld.py

```
PYTHON_PLUGINS_PATH/
  MyPlugin/
    __init__.py     --> *required*
    mainPlugin.py   --> *required*
    metadata.txt    --> *required*
    resources.qrc   --> *likely useful*
    resources.py    --> *compiled version, likely useful*
    form.ui         --> *likely useful*
    form.py         --> *compiled version, likely useful*
```

What is the meaning of the files:

- `__init__.py` = The starting point of the plugin. It has to have the **`classFactory()`** method and may have any other initialisation code.
- `mainPlugin.py` = The main working code of the plugin. Contains all the information about the actions of the plugin and the main code.
- `resources.qrc` = The .xml document created by Qt Designer. Contains relative paths to resources of the forms.
- `resources.py` = The translation of the .qrc file described above to Python.
- `form.ui` = The GUI created by Qt Designer.
- `form.py` = The translation of the form.ui described above to Python.
- `metadata.txt` = Required for QGIS >= 1.8.0. Containts general info, version, name and some other metadata used by plugins website and plugin infrastructure. Since QGIS 2.0 the metadata from `__init__.py` are not accepted anymore and the `metadata.txt` is required.

2. find function "*hello_world*" and rename it to "*getListOfLayers*"
3. change the code of this function to:

```
layers = self.iface.legendInterface().layers()
for layer in layers:
   QMessageBox.information(self.iface.mainWindow(), 'layers', layer.name())
return
```

4. change 2 line of method initGui(self) to:

```
QObject.connect(self.action, SIGNAL("activated()"), self.getListOfLayers)
```

5. you can also change the name of the class and file

2. QGIS PLUGIN GENERATOR

generator - http://www.dimitrisk.gr/qgis/creator/
tutorial - http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/plugins.html

1. open generator and fill info
2. run generator
3. add metadata.txt file

Metadata name	Required	Notes
name	True	a short string containing the name of the plugin
qgisMinimumVersion	True	dotted notation of minimum QGIS version
qgisMaximumVersion	False	dotted notation of maximum QGIS version
description	True	short text which describes the plugin, no HTML allowed
about	True	longer text which describes the plugin in details, no HTML allowed
version	True	short string with the version dotted notation
author	True	author name
email	True	email of the author, will *not* be shown on the web site
changelog	False	string, can be multiline, no HTML allowed
experimental	False	boolean flag, True or False
deprecated	False	boolean flag, True or False, applies to the whole plugin and not just to the uploaded version
tags	False	comma separated list, spaces are allowed inside individual tags
homepage	False	a valid URL pointing to the homepage of your plugin
repository	True	a valid URL for the source code repository
tracker	False	a valid URL for tickets and bug reports
icon	False	a file name or a relative path (relative to the base folder of the plugin's compressed package)
category	False	one of Raster, Vector, Database and Web

example of metadata.txt:

```
; the next section is mandatory

[general]
name=HelloWorld
```

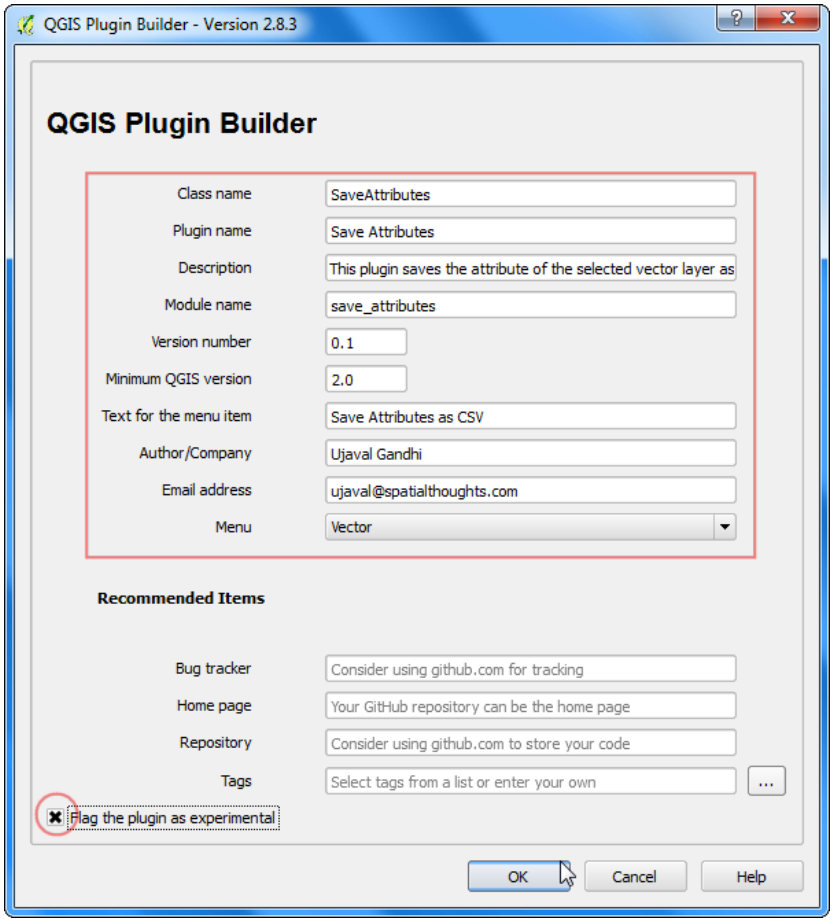
```
email=me@example.com
author=Just Me
qgisMinimumVersion=2.0
description=This is an example plugin for greeting the world.
    Multiline is allowed:
    lines starting with spaces belong to the same
    field, in this case to the "description" field.
    HTML formatting is not allowed.
about=This paragraph can contain a detailed description
    of the plugin. Multiline is allowed, HTML is not.
version=version 1.2
tracker=http://bugs.itopen.it
repository=http://www.itopen.it/repo
; end of mandatory metadata

; start of optional metadata
category=Raster
changelog=The changelog lists the plugin versions
    and their changes as in the example below:
    1.0 - First stable release
    0.9 - All features implemented
    0.8 - First testing release

; Tags are in comma separated value format, spaces are allowed within the
; tag name.
; Tags should be in English language. Please also check for existing tags and
; synonyms before creating a new one.
tags=wkt,raster,hello world

; these metadata can be empty, they will eventually become mandatory.
homepage=http://www.itopen.it
icon=icon.png

; experimental flag (applies to the single version)
experimental=True

; deprecated flag (applies to the whole plugin and not only to the uploaded version)
deprecated=False

; if empty, it will be automatically set to major version + .99
qgisMaximumVersion=2.0
```

3. PLUGIN BUILDER

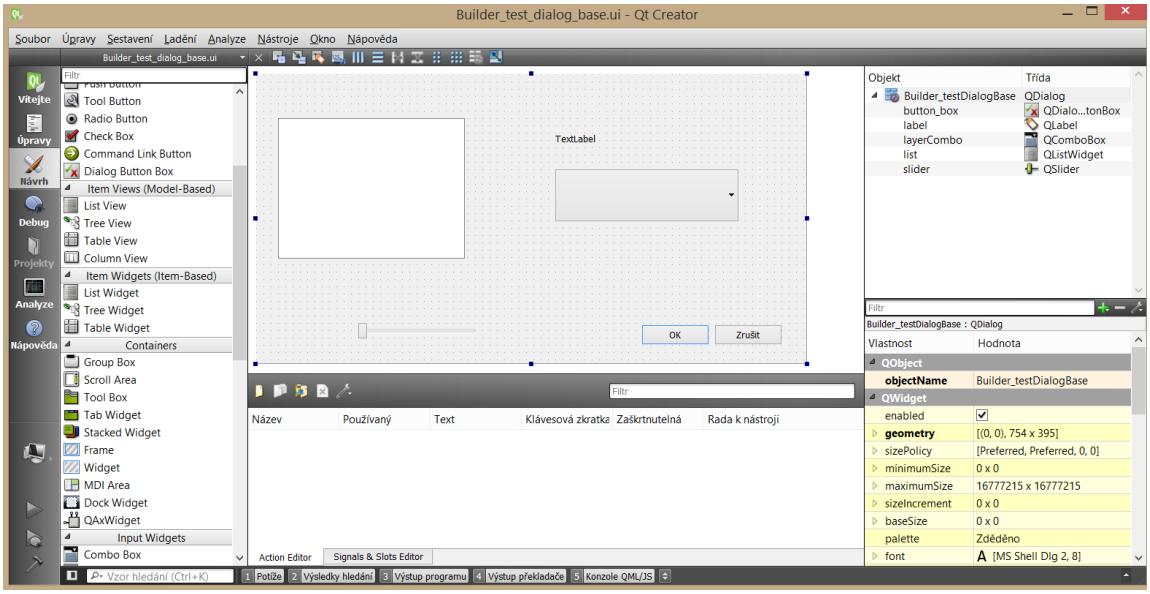
tutorial1 - http://anitagraser.com/2014/04/26/getting-started-writing-qgis-2-x-plugins/
tutorial2 - http://www.qgistutorials.com/en/docs/building_a_python_plugin.html
tutorial3 - http://anitagraser.com/tag/plugin-development/

pyQt documentation - http://pyqt.sourceforge.net/Docs/PyQt4/qtgui.html
Qt designer - http://download.qt-project.org/official_releases/qt/5.0/5.0.2/qt-windows-opensource-5.0.2-mingw47_32-x86-offline.exe
http://sourceforge.net/projects/pyqt/files/PyQt4/PyQt-4.11.4/PyQt-win-gpl-4.11.4.zip/download

1. install and activate "*plugin builder*" in QGIS
2. run "plugin builder" and fill all information

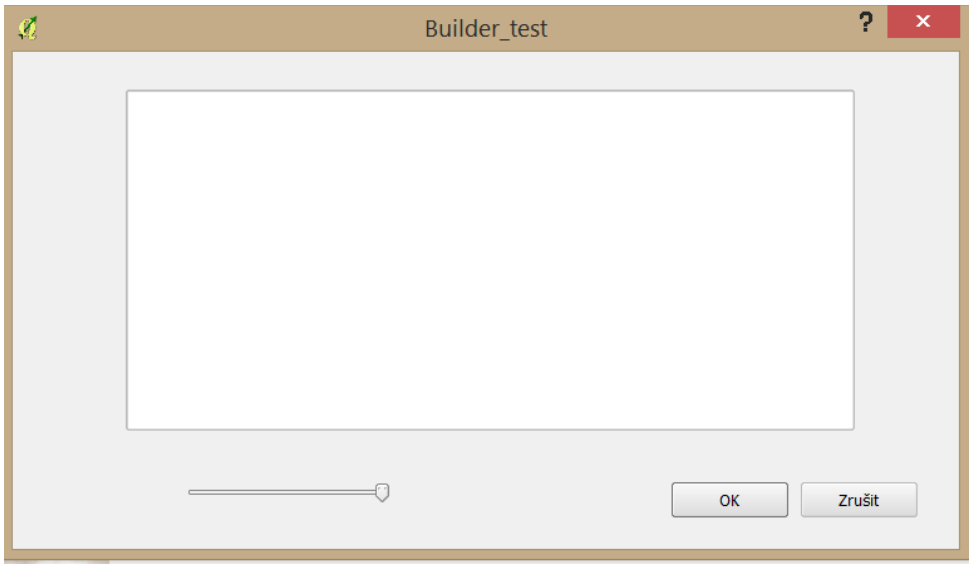


3. open .ui file in Qt Creator (http://www.qt.io/download-open-source/#section-2) or Qt Designer and define UI



- define names of components (upper right window)
- documentation for all components - http://pyqt.sourceforge.net/Docs/PyQt4/qtgui.html

(4. run command in shell `pyrcc4 -o resources.py resources.qrc`)



5. rewrite run function

```
            def run(self):
                """Run method that performs all the real work"""
                # show the dialog
                self.dlg.show()

                layers = QgsMapLayerRegistry.instance().mapLayers().values()

                for layer in layers:
                    self.dlg.label.setText(layer.name())
                    self.dlg.layerCombo.addItem( layer.name(), layer )
                    QListWidgetItem(layer.name(), self.dlg.list);

                result = self.dlg.exec_()

                if result == 1:
                    index = self.dlg.layerCombo.currentIndex()
                    layer = self.dlg.layerCombo.itemData(index)
                    QMessageBox.information(self.iface.mainWindow(),"hello world","%s
has %d features." %(layer.name(),layer.featureCount()))
```

6. import

```
        from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
        from PyQt4.QtGui import QAction, QIcon, QListWidgetItem, QListWidget,
QMessageBox
        from qgis.core import *
```

remove `import resources`

PyQGIS

complete API - http://qgis.org/api/
cookbook - http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/intro.html
repository for all plugins - https://plugins.qgis.org/plugins

loading ids of all features:

```
        def run(self):
            self.dlg.show()

            layers = QgsMapLayerRegistry.instance().mapLayers().values()
            layer = layers[0] # only first layer will be used

            features = layer.getFeatures()

           # iterating over features
            for feature in features:

                # QListWidgetItem takes only string
                if isinstance(str(feature.id()), str):
                    QListWidgetItem(str(feature.id()), self.dlg.list)
```

get number of vertices in line:

```
+   from qgis.core import QgsFeature

    def run(self):
            layer = self.iface.activeLayer()
            iter = layer.getFeatures()
            for feature in iter:
                    geom = feature.geometry()
                    if geom.type() == QGis.Line:
                            line = geom.asPolyline()
                            QMessageBox.information(self.iface.mainWindow(), "line " +
                            str(feature.id()), "line has " + str(len(line)) + " vertices")
```

creating and editing attributes:

```
    # in QGIS console
    from PyQt4.QtCore import *
    import random

    layer = iface.activeLayer()
```

```
pr = layer.dataProvider()
pr.addAttributes([QgsField("blabla", QVariant.Double)])

iter = layer.getFeatures()
layer.startEditing()

index = pr.fields().fieldNameIndex("blabla")

for feature in iter:
        layer.changeAttributeValue(feature.id(), index, random.randint(1, 10))
        # first parameter is id, second id of attribute and third new value

# for QGIS plugin
# + from PyQt4.QtCore import QVariant
# + from qgis.core import QgsField
```

creating new geometry:

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import  *

layer = iface.activeLayer()
layer.startEditing()

feat = QgsFeature()
feat.setGeometry(QgsGeometry.fromPoint(QgsPoint(0, 0)))
layer.dataProvider().addFeatures([feat])
layer.commitChanges()
iface.mapCanvas().refresh()
```

other snippets:

```
# creating new layer
lineLayer = QgsVectorLayer("LineString", "new Name", "memory")

# save changes
lineLayer.commitChanges()

# set selection
layer.setSelectedFeatures([])

# selected Features will have red color (default yellow)
from PyQt4.QtGui import *
iface.mapCanvas().setSelectionColor(QColor("red") )
```

```
# delete features
layer.dataProvider().deleteFeatures([5, 10])
```