

10 Einführung QGIS

Dieses Kapitel basiert unter anderem auf dem „QGIS User Guide“ und dem „QGIS Training Manual“.

10.1 Was ist QGIS ?

Ein Geoinformationssystem resp. Geographisches Informationssystem (GIS) ist ein Informationssystem (IS), mit dem raumbezogene Daten digital **erfasst, bearbeitet, organisiert, analysiert** und **präsentiert** werden.

Heute gibt es eine Vielzahl an GIS von verschiedenen Herstellern, wie zum Beispiel ArcGIS von ESRI oder GeoMedia von Intergraph. Es gibt auch einige Open Source GIS, wie zum Beispiel QGIS. Open Source bedeutet, dass der Source Code der gesamten Applikation frei verfügbar ist. Das wichtigste dabei ist nicht, dass dieses GIS gratis ist, sondern dass dieses GIS frei weiterentwickelt werden kann – unabhängig von einem einzigen kommerziellen Anbieter. So ist QGIS heute auch in über 30 Sprachen verfügbar, dank freiwilligen Übersetzern. Softwareentwickler können Erweiterungen schreiben – dabei kann C++ oder Python verwendet werden.

Warum sollte QGIS verwendet werden ? Einige der Gründe sind:

- Gratis erhältlich – keine Kosten
- Die Software ist frei – Wenn neue Funktionalität gewünscht wird, kann diese entweder selbst implementiert werden oder man sponsert die Entwicklung dieser Funktionalität.
- QGIS wird immer weiterentwickelt
- Eine Vielzahl von Dokumentationen ist verfügbar. Es gibt auch eine grosse Community – man kann auch die Entwickler oder andere QGIS Nutzende direkt fragen.
- QGIS gibt es für Windows, MacOS und Linux.

QGIS unterstützt gängige Vektorformate und Rasterdaten wie z.B. Shapefiles oder GeoTIFF. Auch räumliche Datenbanken (wie z.B. PostGIS) werden unterstützt. Auch OGC Services wie z.B. WMS oder WFS Dienste können eingebunden werden.

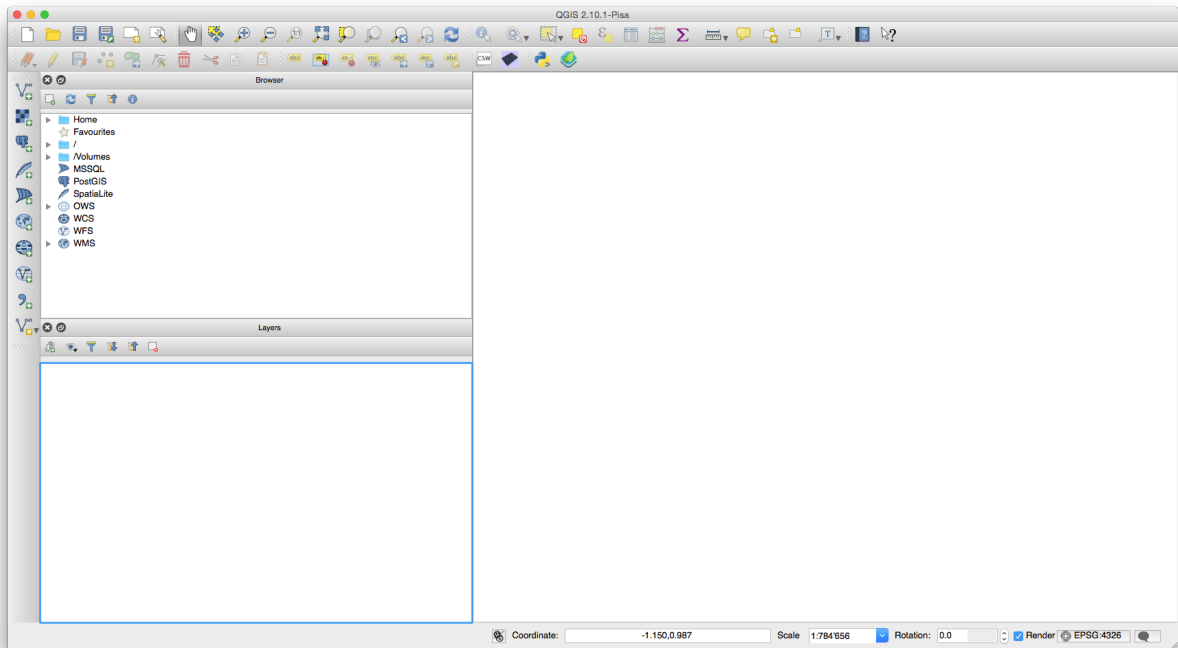
Falls irgendeine wichtige Funktion fehlt, so kann diese also Plugin oder im Core implementiert werden!

Die erste Version von QGIS wurde im Juli 2002 veröffentlicht. Die Version 1.0.0 wurde im Januar 2009 freigegeben und die Version 2.0.0 im September 2013.

QGIS ist unter <http://www.qgis.org/> erhältlich.

Für die ersten Versuche verwenden wir einige Demo-Daten, welche auf Moodle erhältlich sind (demo.zip).

Wird QGIS gestartet sehen wir folgendes Fenster.




10.2 Arbeiten mit Vektordaten

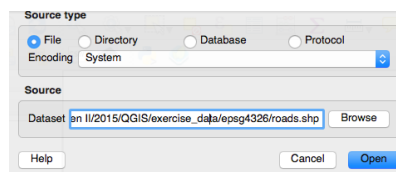
QGIS verwendet die OGR-Bibliothek (<http://www.gdal.org>) um Vektordatenformate zu lesen und schreiben. Unter anderem werden ESRI Shapefiles, MapInfo, MicroStation Formate, AutoCAD DXF, PostGIS, Spatialite, Oracle Spatial, MSSQL Spatial Datenbanken.

Auch PostgreSQL wird von QGIS unterstützt. Das OpenStreetMap Vektorformat wird auch direkt unterstützt.

In QGIS können Vektordaten auch aus zip-Archiven (und gzip-Archiven) gelesen werden, dies ist v.a. bei Shapefiles sehr nützlich, da diese aus verschiedenen Files bestehen.

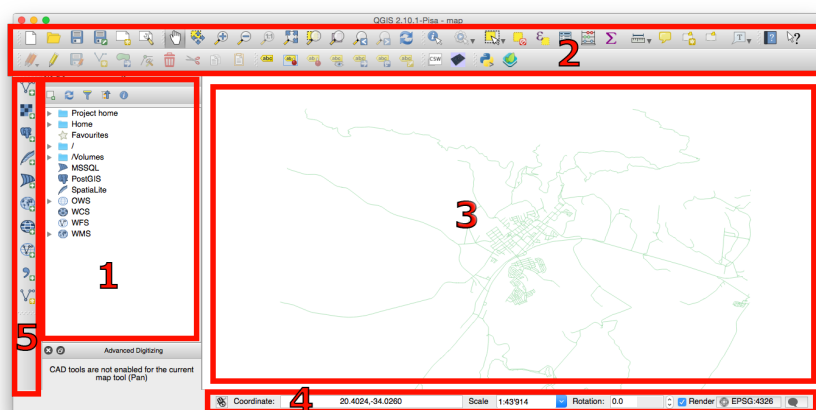
Wir sehen eine neue leere Karte. Auf der linken Seite befindet sich das Symbol . Mit diesem können wir einen Vektor-Layer hinzufügen. Wir wählen das folgende File:

```
exercise_data/epsg4326/roads.shp
```




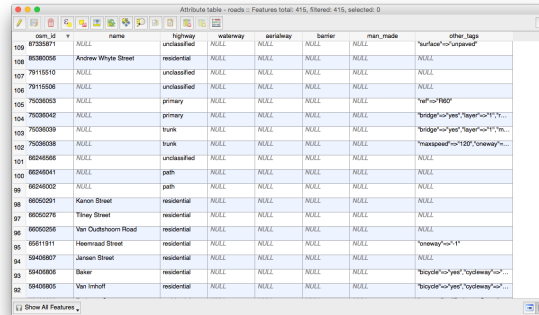
Mit dem Symbol  kann nun das Projekt gespeichert werden. Wir speichern das Projekt unter map.qgs

Die Benutzeroberfläche ist in QGIS folgendermassen aufgebaut:




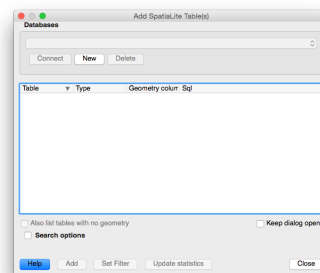
- 1: Layer-Liste / Browser
- 2: Toolbars
- 3: Map Canvas
- 4: Status Bar
- 5: Side Toolbar

Zunächst sehen wir uns alle Attribute des Layers an. Dies geschieht durch Klick auf den Layer („roads“) und dann auf das Symbol . Nun wird eine Tabelle mit allen Attributen angezeigt:



geom_id	name	highway	surface	barrier	man_made	other_tags
87259871		unclassified	NO	NO	NO	"surface"~>"unpaved"
80360995	Ardener Wyle Street	residential	NO	NO	NO	
79115510		unclassified	NO	NO	NO	
79115506		unclassified	NO	NO	NO	
75036053		primary	NO	NO	NO	"width"~>"100"
75036043		primary	NO	NO	NO	"bridge"~>"yes","type"~>"V..."
75036039		tunk	NO	NO	NO	"bridge"~>"yes","type"~>"V..."
75036038		tunk	NO	NO	NO	"maxspeed"~>"120","oneway"~>"..."
80240586		unclassified	NO	NO	NO	
80240541		path	NO	NO	NO	
80240502		path	NO	NO	NO	
80240501	Kipon Street	residential	NO	NO	NO	
80240500	Thway Street	residential	NO	NO	NO	
80240500	Van Outhaarn Road	residential	NO	NO	NO	
6611911	Heemraad Street	residential	NO	NO	NO	"oneway"~>"-1"
59403887	Jarvis Street	residential	NO	NO	NO	
59403886	Baker	residential	NO	NO	NO	"bicycle"~>"yes","oneway"~>"..."
59403885	Van Inhoff	residential	NO	NO	NO	"bicycle"~>"yes","oneway"~>"..."

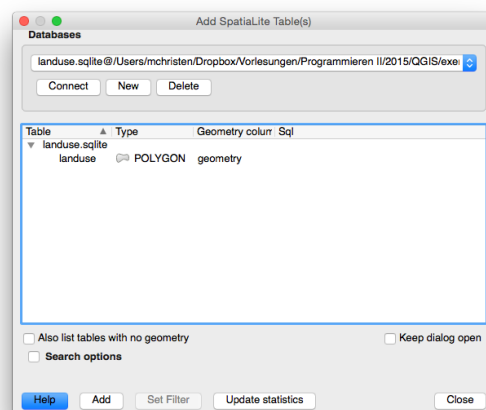
Nun laden wir Daten aus einer Datenbank. Wir fügen Vektordaten aus einer „Spatialite“ Datenbank hinzu. Dazu klicken wir auf das Icon .



Im Dialog wird dann mit „New“ eine neue Datenbankverbindung aufgebaut. In unserem Fall ist dies folgendes File:

```
exercise_data/epsg4326/landuse.sqlite
```

Danach wird die Datenbank mittels dem „Connect“-Button verbunden:



Danach wird „landuse“ (Landnutzung) gewählt und mittels „Add“ hinzugefügt. Die Karte sollte danach etwa so aussehen:



Wir sehen, dass der „landuse“ Layer den „roads“ Layer überdeckt. Wir können dies im „Layers“ Tool einfach mittels Drag & Drop anpassen, so dass „landuse“ der erste Layer ist.

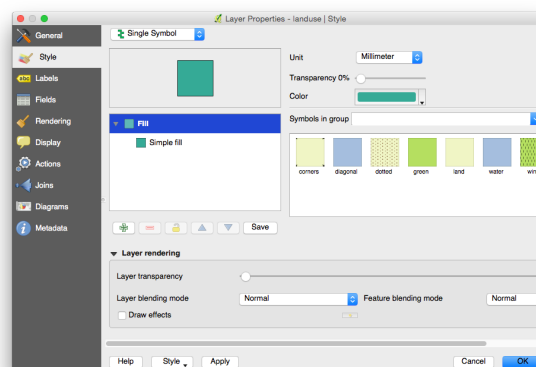
Nun können wir auch noch weitere Vektor-Layer hinzufügen:

`places.shp, buildings.shp, rivers.shp, water.shp`

Die Reihenfolge der Layer sollte dann: landuse, water, rivers, roads, buildings, places (von unten nach oben) sein. Unsere Karte sieht dann etwa so aus:



Diese Karte ist natürlich alles andere als schön, aber wir können dies über die Layereigenschaften anpassen. Zunächst machen wir einen Rechtsklick auf den „landuse“ Layer und wählen „Eigenschaften“. Dann erscheint dieses Fenster:



Danach können wir „Einfache Füllung“ wählen und die Farbe beispielsweise auf grau setzen.


Nun öffnen wir die Eigenschaften des „landuse“ Layers wieder und setzen den „Randstil“ auf „Kein Stift“.

Nun ändern wir die Farben der anderen Layers, und zwar so dass:

- Der waters-Layer dunkelblau wird
- Der rivers-Layer hellblau wird

Es ist auch möglich gewisse Layer anhand des Massstabs darzustellen. Beispielsweise macht es wenig Sinn alle Gebäude schon von weit entfernt darzustellen – auch aus Performancegründen.

Dazu öffnen wir die Eigenschaften des „buildings“ Layer. Im Tab „Allgemein“ kann „Massstabsabhängige Sichtbarkeit“ aktiviert werden. Dort setzen wir Minimum auf 1:10000 und das Maximum auf 1:1.

Nun gehen wir zurück auf die Eigenschaften des landuse-Layers und fügen mit  einen Symbollayer hinzu und setzen dabei folgendes:

- Den Randstil auf „Kein Stift“.
- Den Füllstil auf „Dense 6“.

Nun ändern wir den style des roads-Layer. Wir fügen einen neuen Symbollayer hinzu, setzen die Basis Stiftbreite auf 0.3 und den Stiftstil auf „Gestrichelte Linie“. Den neuen Symbollayer setzen wir auf Strichdicke 1.3. Das Resultat sieht dann etwa so aus:




Dies ist allerdings nicht das gewünschte Result, da wir die Reihenfolge der Symbolelayer umdrehen müssen!

Dies geschieht mittels Drag & Drop.
Die Strassentypen können auch klassifiziert werden.

10.3 Arbeiten mit Rasterdaten

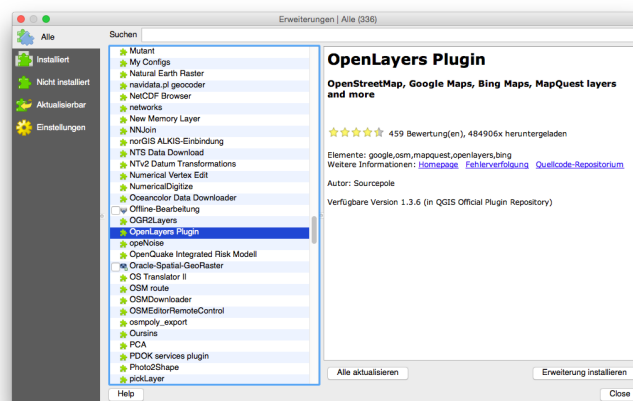
QGIS verwendet die GDAL Bibliothek (<http://www.gdal.org>) um Rasterdatenformate zu lesen und schreiben. Es werden weit mehr als 100 Rasterformate unterstützt, wie z.B. Arc/Info Binary Grid, Arc/Info ASCII Grid, GeoTIFF, Erdas Imagine.

Rasterdaten werden ganz einfach mit dem Symbol  hinzugefügt. Wir werden Rasterdaten nur am Rande betrachten.

10.4 Plugins

In QGIS können wir eine Vielzahl an Plugins installieren. Wir installieren nun das „OpenLayers Plugin“. Dies geschieht über das Menu „Erweiterungen/Erweiterungen verwalten und installieren“. Wir werden hier nur zwei Erweiterungen ansehen.

10.4.1 OpenLayers Plugin



Im „Web“ Menu ist nun der Eintrag „OpenLayers Plugin“ zu sehen. Dort wählen wir „OpenStreetMap/OpenStreetMap“ aus. Danach sehen wir einen neuen Layer „OpenStreetMap“ und die Karte aus OpenStreetMap. Auch das Referenzsystem (Bezugssystem) hat geändert. Wir sehen nun „EPSG:3857“ in der Statusbar.

10.5 Programmieren mit QGIS

In diesem Kapitel lernen wir innerhalb von QGIS zu programmieren. Als Testdaten verwenden wir das GeoPackage von natural-earth:

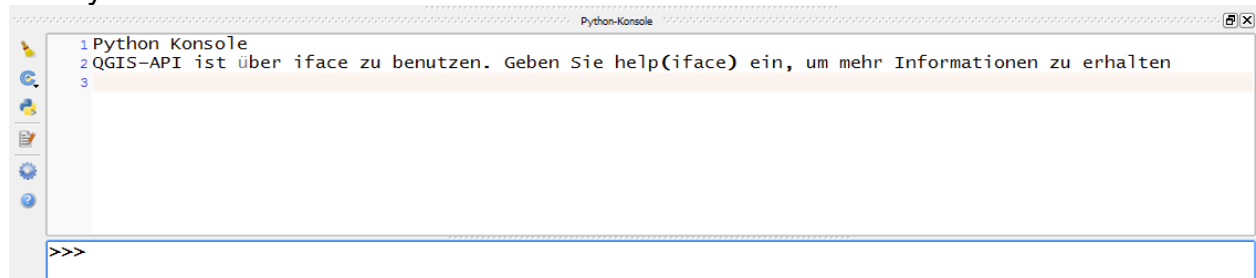
<https://www.naturalearthdata.com/downloads/>

10.5.1 Verwendung der Python Konsole

Soll etwas innerhalb von QGIS entwickelt werden, so kann dazu die Python Konsole verwendet werden. Meistens sind es kleine Skripts die über diesen Weg ausgeführt werden.

Die Python Konsole kann über das Menu „Erweiterungen/Python Konsole“ geöffnet werden oder auch ganz einfach mit Klick auf das Python-Konsole Icon.

Die Python Konsole erscheint:



Wenn die Konsole geöffnet wird, so wird automatisch folgender Python Code ausgeführt:

```
from qgis.core import *
import qgis.utils
```

Das heisst Teile des qgis Moduls werden automatisch importiert. Dies ist nützlich, da es mühsam wäre dies jedesmal immer wieder neu einzutippen.

10.5.2 Projekte Laden und Speichern

Wir starten QGIS und fügen aus dem Natural Earth GeoPackage den Layer ne_50m_airports zu.

In der Python Konsole schreiben wir nun:

```
project = QgsProject.instance()
project.write("C:/data/QGIS_Projekt/projekt.qgs")
```


Nun könnten wir Änderungen vornehmen und das Projekt über Python speichern:

```
project.write()
```

oder um das Projekt unter einem anderen Namen zu speichern rufen wir auf:

```
project.write("C:/data/QGIS_Projekt/neues_projekt.qgs")
```

Die Methoden `read()` und `write()` geben übrigens jeweils `True` oder `False` zurück, falls überprüft werden soll, ob das Laden resp. Speichern funktioniert hat.

Um ein Projekt zu laden rufen wir einfach folgendes auf:

```
project.read("C:/data/QGIS_Projekt/projekt.qgs")
```

10.5.3 Arbeiten mit Layern

Um alle Layer-Objekte zu erhalten kann das über das `mapCanvas()` Objekt geschehen. Die `MapCanvas` Klasse ist dafür verantwortlich alle GIS-Objekte darzustellen. Wir greifen auf den `MapCanvas` und die Layer folgendermassen zu:

```
canvas = iface.mapCanvas()  
layers = canvas.layers()
```

Um die Namen aller Layers auszugeben können wir folgendes definieren:

```
for l in layers:  
    print(l.name())
```

Um zu überprüfen, ob es sich wirklich um Vektor-Layer handelt, können wir folgendes schreiben:

```
for l in layers:  
    if l.type() == QgsMapLayer.VectorLayer:  
        print(l.name())
```

Im Falle eines Rasters, wäre der Layer-Typ dann einfach
`QgsMapLayer.RasterLayer`

Die Dokumentation, inklusive dem Klassendiagramm zum Map-Layer finden wir auf:
<http://qgis.org/api/classQgsMapLayer.html>

Diese Dokumentation ist allerdings für C++, kann aber leicht auch für Python verstanden werden.

Sehen wir uns mal die Features des ersten Layers an („ne_10m_airports“):

```
airports = layers[0]
features = airports.getFeatures():
for f in features:
    print(f.id())
```

Der Einfachheit halber betrachten wir mal das erste Feature in der Liste:

```
f = list(airports.getFeatures())[0]

id = f.id()           # die ID des Features erhalten
geometry = f.geometry() # auf die Geometrie des Features zugreifen
```

Nun können wir überprüfen, um was für eine Geometrie es sich handelt. Bei den Flughäfen sind es jedoch alles Punkte (<https://qgis.org/pyqgis/3.0/core/Wkb/QgsWkbTypes.html>)

```
if geometry.type() == QgsWkbTypes.PointGeometry:
    print("Typ ist Punkt")
elif geometry.type() == QgsWkbTypes.LineString:
    print("Typ ist Linie")
elif geometry.type() == QgsWkbTypes.Polygon:
    print("Typ ist Polygon")
```

Wir können auch von jedem Feature auf die Attribute zugreifen, dies geschieht folgendermassen:

```
attrib = f.attributes()
print(attrib)
```

Attrib ist eine Liste mit sämtlichen Attributen des Features.

Um die Definition der Attribute zu erhalten können wir im Layer auf die „pendingFields“ zugreifen:

```
for field in airports.fields():
    print(field.name() + ", " + field.typeName())
```

Also das Feld 3 wäre in unserem Beispiel der Name des Flughafens.

Die Geometrie erhalten wir folgendermassen:

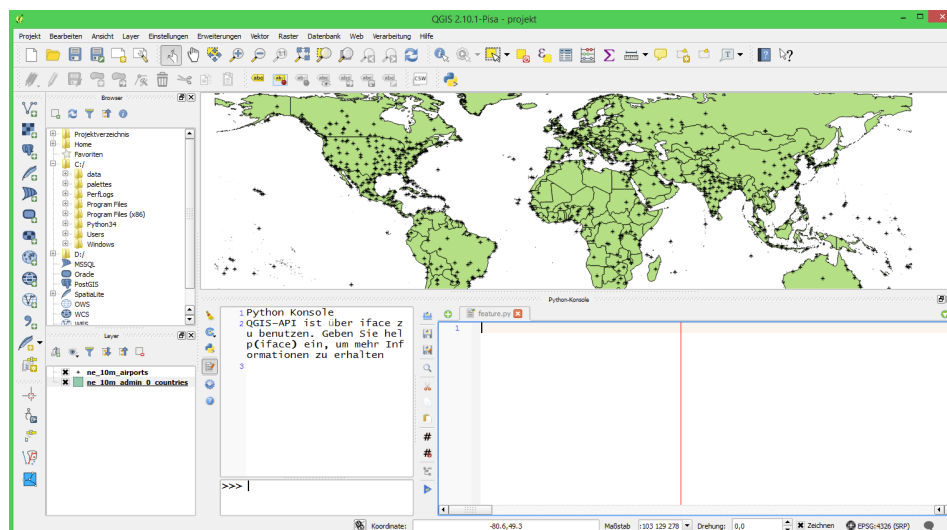
```
wkt = geometry.asWkt()
```

```
point = geometry.asPoint()
print(point.x(), point.y())
```

Nun müssen wir zurück zur QGIS Oberfläche: In QGIS haben wir einen aktiven Layer – das ist der Layer der gerade gewählt ist. Dies geschieht mit der `activeLayer` Methode. Falls kein Layer gewählt ist, so wird `None` zurückgegeben.

```
layer = iface.activeLayer()
```

Wir fügen den admin0 layer aus dem GeoPackage hinzu. Dann selektieren wir im GUI den Layer „ne_10m_admin_0_countries“. Wir wählen in der Python Konsole nun den Button „Editor Anzeigen“ und speichern das File als „feature.py“ ab:



Nun schreiben wir ein kleines Programm, bei dem die Schweiz rot eingefärbt werden soll. Zunächst müssen wir die „SelectionColor“ des Map Canvas auf rot setzen. Dies geschieht folgendermassen (<https://qgis.org/pyqgis/3.0/core/Vector/QgsVectorLayer.html>):

```
from PyQt5.QtGui import QColor

canvas = iface.mapCanvas()
canvas.setSelectionColor(QColor("red"))
```

Dann können wir mit der Methode `setSelectedFeatures` der Layer-Klasse die Features über dessen id wählen:

```
layer = iface.activeLayer()
layer.select(93)
layer.select(92)
layer.select(51)

layer.deselect(51)
```

10.5.4 Einen Vektor Layer erzeugen

Die Klasse „QgsVectorLayer“ erzeugt einen Vector Layer. Standardmässig können in QGIS mindestens folgende Layer erstellt werden:

- Shapefile-Layer – ein ESRI Shapefile
- Spatialite-Layer – ein Datenbanklayer mit Spatialite
- Temporärlayer – ein Layer welcher im Speicher (temporär) gehalten wird

Zunächst importieren wir alle Klassen von PyQt5.QtCore, da wir auf Funktionalität von Qt zurückgreifen müssen:

```
from PyQt5.QtCore import *
```

Der einfachste Vektorlayer ist ein Temporärlayer. Dieser wird z.B. folgendermassen erstellt:

```
layer = QgsVectorLayer("Point?crs=epsg:4326",
                       "temporary_points",
                       "memory")
```

Jeder Layer verfügt um einen sogenannten „Data-Provider“, welcher über gewisse Eigenschaften verfügt, welche formatspezifisch sind. So hat z.B. ein Shapefile andere Fähigkeiten als ein PostGIS Layer. Der Data-Provider kann über die Layer-Klasse erhalten werden:

```
provider = layer.dataProvider()
```

Die Fähigkeiten des Data-Providers können über die „Capabilities“ abgefragt werden:

```
caps = provider.capabilities()

if caps & QgsVectorDataProvider.AddFeatures:
    print(u"Juhuuu!! Es können neue Features hinzugefügt werden!")
```

Es können u.a. folgende Fähigkeiten des Data-Providers abgefragt werden:

```
QgsVectorDataProvider.
    AddFeatures
    DeleteFeatures
```

```
ChangeAttributeValues
AddAttributes
DeleteAttributes
SaveAsShapefile
CreateSpatialIndex
SelectAtId
ChangeGeometries
SelectGeometryAtId
SelectEncoding
```

Nun muss der Vektor-Layer „editierbar“ gemacht werden, dies geschieht mit:

```
layer.startEditing()
```

Um die Felder der Attribute zu definieren wird über den Data-Provider die Methode „addAttribute“ verwendet. Als Parameter wird eine Liste von QgsField verwendet. Ein QgsField besteht aus einem Namen und einem Datentyp (QVariant.String, QVariant.Int, QVariant.Double).

```
provider.addAttributes([QgsField("name", QVariant.String),
                        QgsField("gewichtung", QVariant.Int)])
```

Nun können die Features erstellt und hinzugefügt werden. Ein Feature wird mit der Klasse QgsFeatures erstellt. Dem Feature kann dann die Geometrie und die Attribute übergeben werden und über den Data-Provider hinzugefügt werden:

```
feature = QgsFeature()
feature.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(7.6386, 47.5336)))
feature.setAttributes(["Muttentz", 5])
provider.addFeatures([feature])
```

Um das editieren zu beenden wird beim Layer die Methode „commitChanges“ aufgerufen:

```
layer.commitChanges()
```

Nun muss noch der Umfang des Layers berechnet werden, respektive angepasst werden. Dies geht mit der Methode „updateExtents()“ des Layers.

```
layer.updateExtents()
```

Um den Layer in QGIS in der Legende hinzuzufügen, wird noch folgendes aufgerufen:

```
project = QgsProject.instance()
project.addMapLayer(layer)
```

Der Temporärlayer kann nun gespeichert werden. Ein Shapefile kann folgendermassen erzeugt werden:

```
QgsVectorFileWriter.writeAsVectorFormat(layer,
                                         "myshape.shp",
```

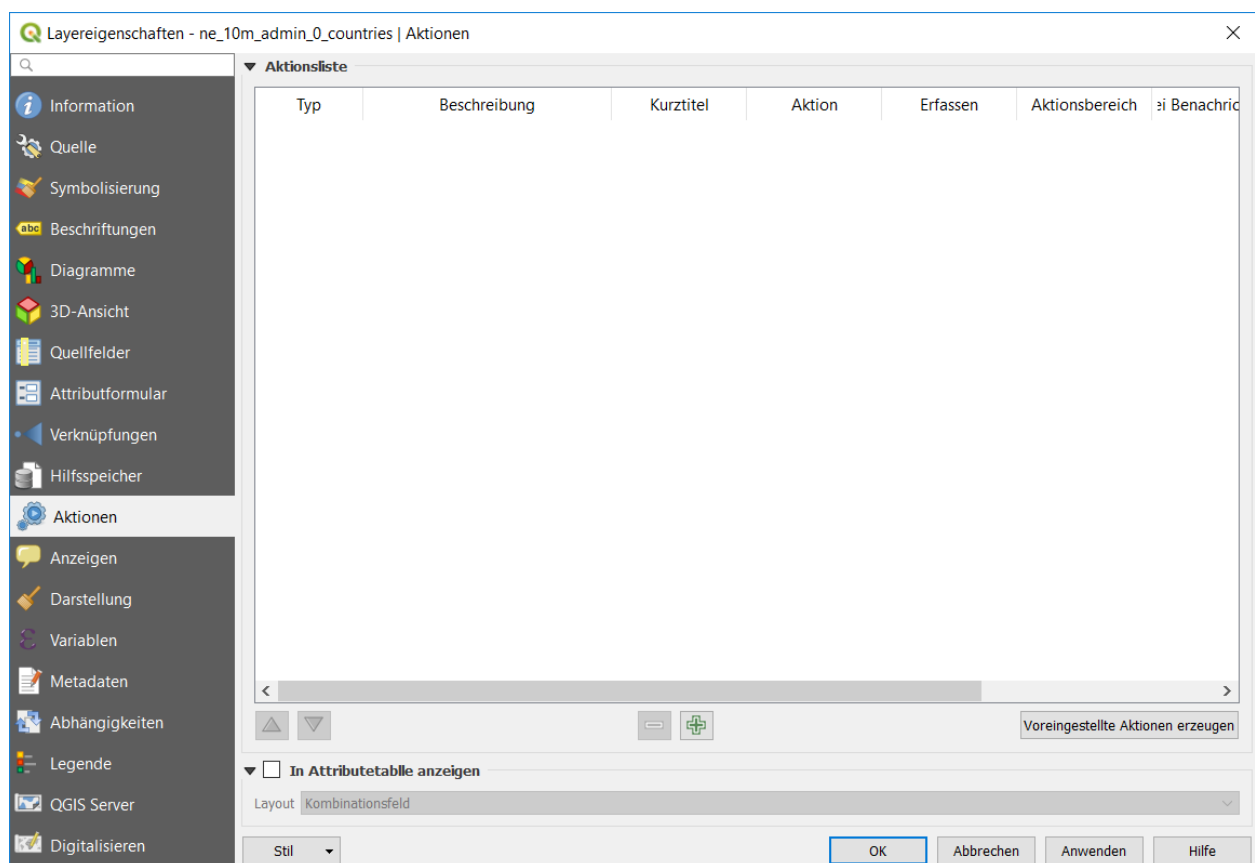
```
provider.encoding(),  
provider.crs()
```

10.6 Python Actions in QGIS

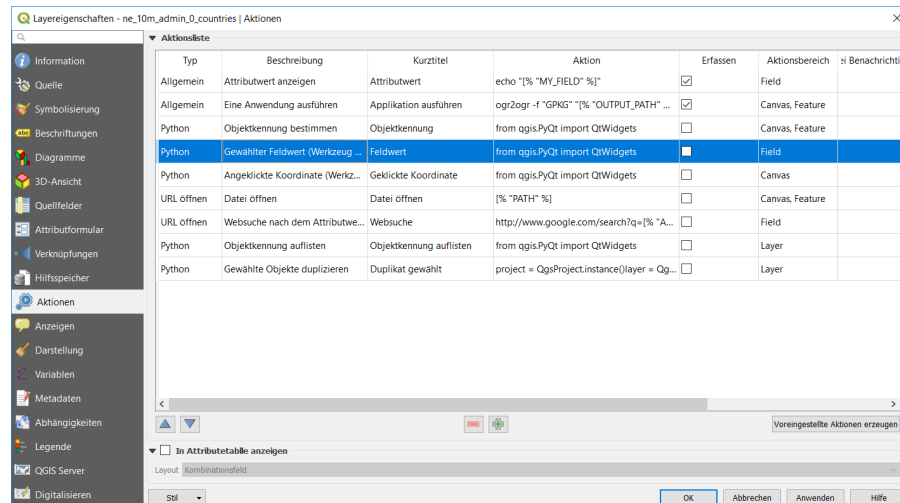
In diesem Kapitel lernen wir wie wir QGIS neue Funktionalität hinzufügen mit Python & Actions.

10.6.1 Erstellen einer Python Action

Wir erstellen ein neues QGIS Projekt und fügen den Layer «ne_110m_admin_0_countries» aus dem GeoPackage von Natural Earth hinzu. Mit einem Rechtsklick auf den Layer wählen wir im Pop-up-Menu «**Eigenschaften...**». Dort wählen wir den «Aktionen» Tab:



Dort klicken wir auf den Button «Voreingestellt Aktionen erzeugen»

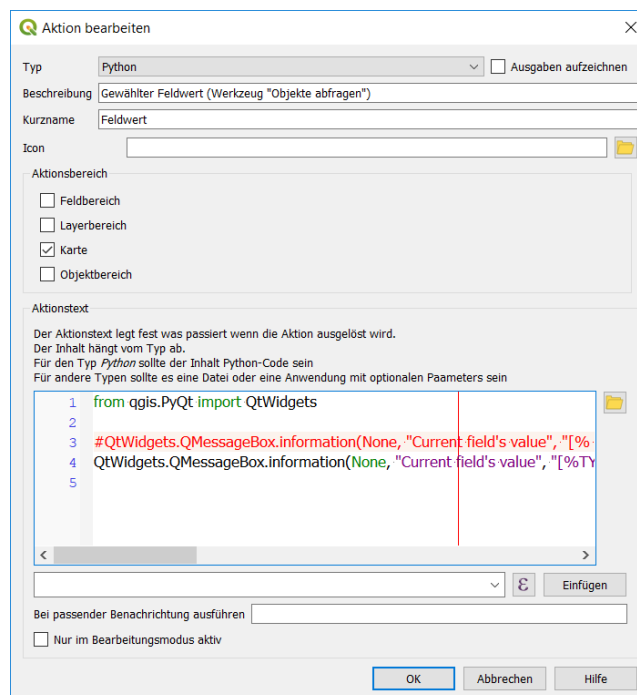


Wir wählen «Gewählter Feldwert» mit einem Doppelklick. Dort ändern wir den code zu folgendem:

```
from PyQt5 import QtWidgets

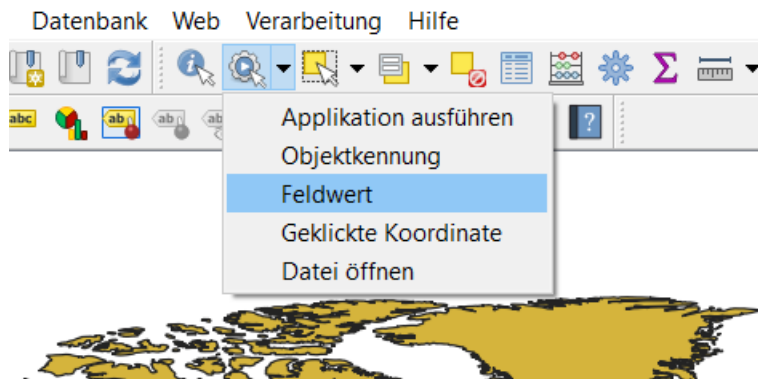
QtWidgets.QMessageBox.information(None, "Current field's value", "[%TYPE%]")
```

Wir ändern den Aktionsbereich auf «Karte» und deselektieren «Feldbereich».



Wir bestätigen 2x mit Ok.

Nun wählen wir die Aktion «Feldwert»



Wir klicken auf die Länder und das Python-Script wird jeweils ausgeführt!

10.6.2 Eine neue Python Action erstellen

Wir erstellen nun eine komplett neue Action, welche jeweils die Geometrie in die Python Konsole ausgibt. Dieses Beispiel zeigt, wie wir Layer Actions mit den QGIS Python Funktionalität kombinieren lassen.

```
from qgis.core import *
from qgis.utils import *

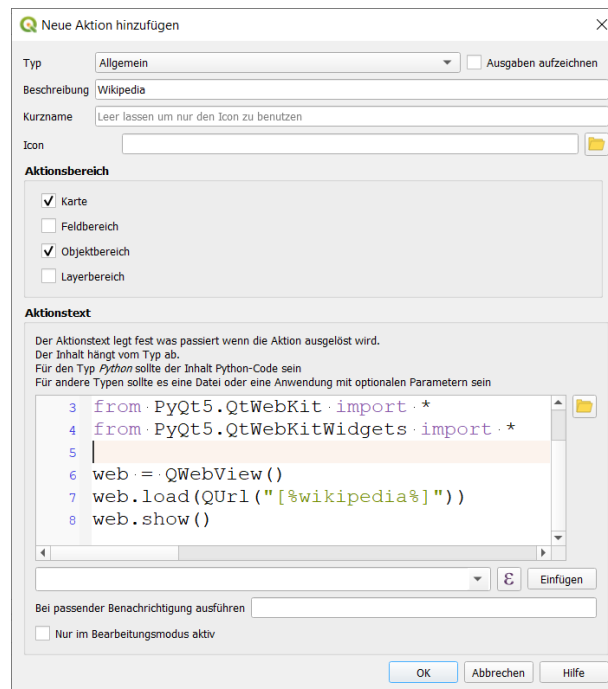
clicked = "[% $id %]"

if clicked != "":
    clickedid = int(clicked)
    print(clicked, type(clicked))
    layer = iface.activeLayer()
    features = layer.getFeatures()
    for feature in features:
        if feature.id() == clickedid:
            print("found!")
            print(feature.geometry().asWkt())
```


10.6.3 Beispiel: URL öffnen

Im Airport Datensatz ist eine Spalte «wikipedia». Diese URL soll im Webbrowser geöffnet werden.

Dazu erstellen wir eine Python Action:

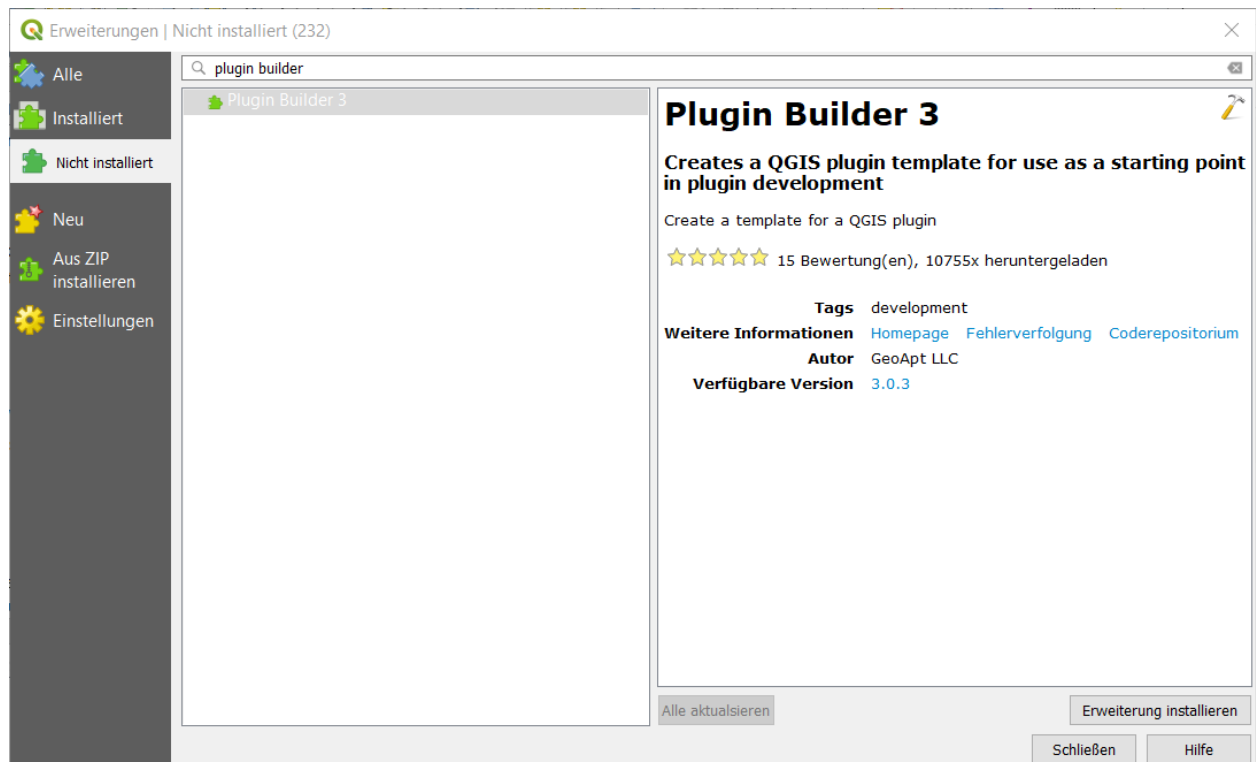


```
from PyQt5 import QtCore
from PyQt5.QtCore import *
from PyQt5.QtWebKit import *
from PyQt5.QtWebKitWidgets import *
web = QWebView()
web.load(QUrl("[%wikipedia%]"))
web.show()
```

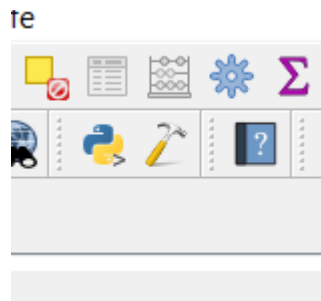
10.7 Erstellen von QGIS Plugins

Wir haben nun gesehen wie Python-Skripte in QGIS verwendet werden können. Nun werden wir nun lernen wie **Plugins** erstellt werden.

Es gibt in QGIS ein Plugin, welches die Erstellung von neuen Plugins erleichtert. Dieses Plugin heisse «Plugin Builder 3» und kann ganz einfach installiert werden:



Nach erfolgter Installation kann der Plugin Builder (Hammer-Symbol) gestartet werden:

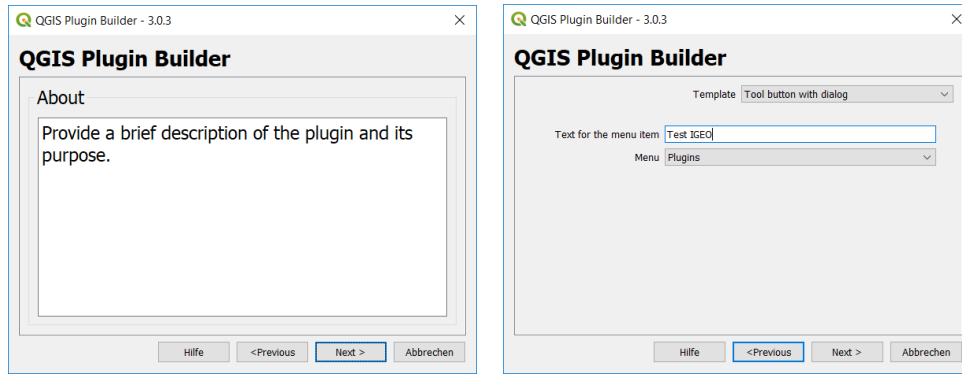


Nach dem Start des Plugin Builders wird ein Fenster geöffnet, in dem mehrere Informationen abgefragt werden:

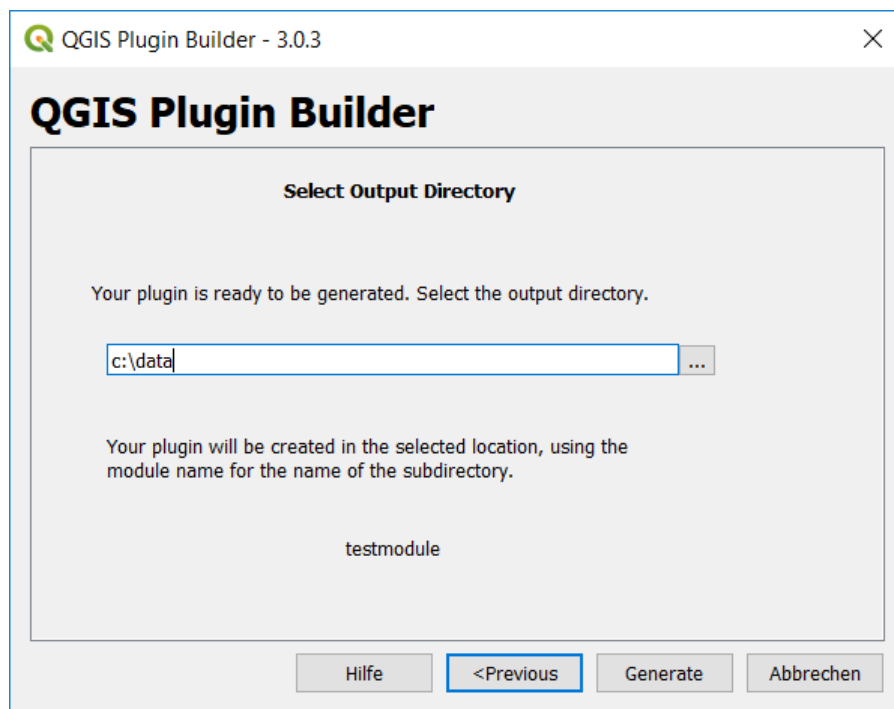
The screenshot shows the 'QGIS Plugin Builder - 3.0.3' window. The title bar includes the QGIS logo and the text 'QGIS Plugin Builder - 3.0.3' with a close button. The main area is titled 'QGIS Plugin Builder' in large bold letters. Below the title is a form with several input fields: 'Class name' (TestPlugin), 'Plugin name' (TestPlugin), 'Description' (Dies ist ein Tet-Plugin), 'Module name' (TestModule), 'Version number' (0.1), 'Minimum QGIS version' (3.0), 'Author/Company' (FHNW IGEO), and 'Email address' (martin.christen@fhnw.ch). At the bottom of the form are four buttons: 'Hilfe', '<Previous', 'Next >', and 'Abbrechen'.

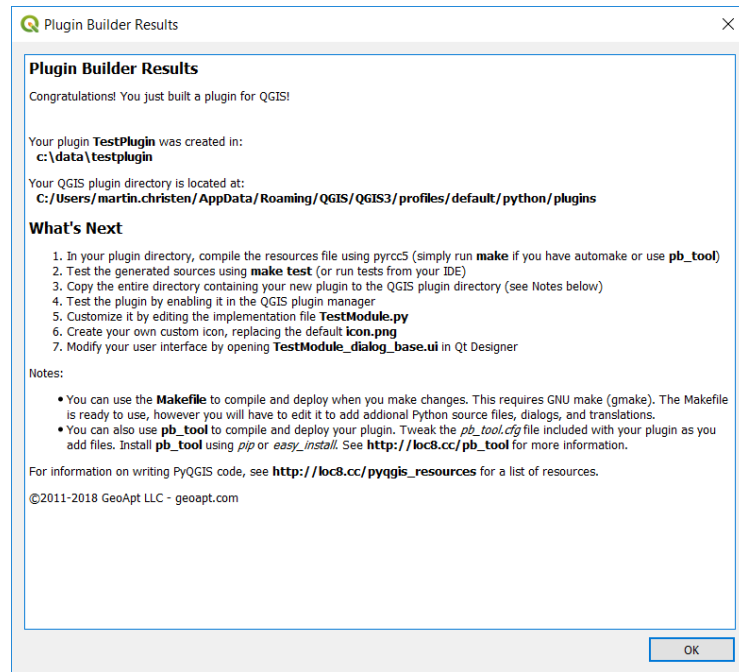
Wir füllen nun den Dialog auf. Der Klassenname und das Plugin soll «TestPlugin» heissen.

Im Feld «About» wird danach einfach eine Beschreibung des Plugins angegeben, und danach wird «Tool button with dialog» ausgewählt und ein Menu Eintrag definiert. In unserem Fall «Test IGEO»



Danach wird die URL für den Bug-Tracker und das Respositry (wo der Source Code sich befindet) angegeben. Da wir dies nicht unterstützten schreiben wir n/a (not available). Bei einem richtigen Plugin ist dies jedoch erforderlich. Dann geben wir an, wo das Plugin installiert werden soll:





In C:\Programme\QGIS ein File «OSGeo4W_New.bat» erstellen mit folgendem Inhalt:

```
@echo off
rem Root OSGEO4W home dir to the same directory this script exists in
call "%dp0\bin\o4w_env.bat"
@echo off
call "%OSGEO4W_ROOT%\bin\o4w_env.bat"
call "%OSGEO4W_ROOT%\apps\grass\grass-7.4.2\etc\env.bat"
path %PATH%;%OSGEO4W_ROOT%\apps\qgis\bin
path %PATH%;%OSGEO4W_ROOT%\apps\grass\grass-7.4.2\lib
path %PATH%;%OSGEO4W_ROOT%\apps\Qt5\bin
SET PYTHONPATH=
SET PYTHONHOME=%OSGEO4W_ROOT%\apps\Python37
PATH %OSGEO4W_ROOT%\apps\Python37;%OSGEO4W_ROOT%\apps\Python37\Scripts;%PATH%

rem List available o4w programs
rem but only if osgeo4w called without parameters
@echo on
@if [%1]==[] (echo run o-help for a list of available commands & cmd.exe /k) else (cmd /c "%*")
```

Zunächst installieren wird das pb_tool:

```
pip install pb_tool
```

Dort geben wir «cd» (Current directory) das das Verzeichnis ein, wo das Plugin gespeichert wurde.

Also z.B:

```
cd c:\data\testplugin
```

Dann deployen wir das Plugin

```
pb_tool deploy
```

Nach Verändern des GUIs (designer.exe) geben wir jeweils ein:

```
pyrcc5 -o resources.py resources.qrc
```

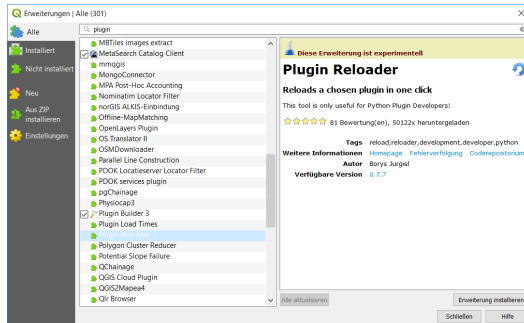
Nun starten wir QGIS erneut (es muss vorher geschlossen worden sein!).

Das Plugin kann gestartet werden!

10.8 QGIS Plugin Entwicklung mit PyCharm

10.8.1 Installation Plugin Reloader

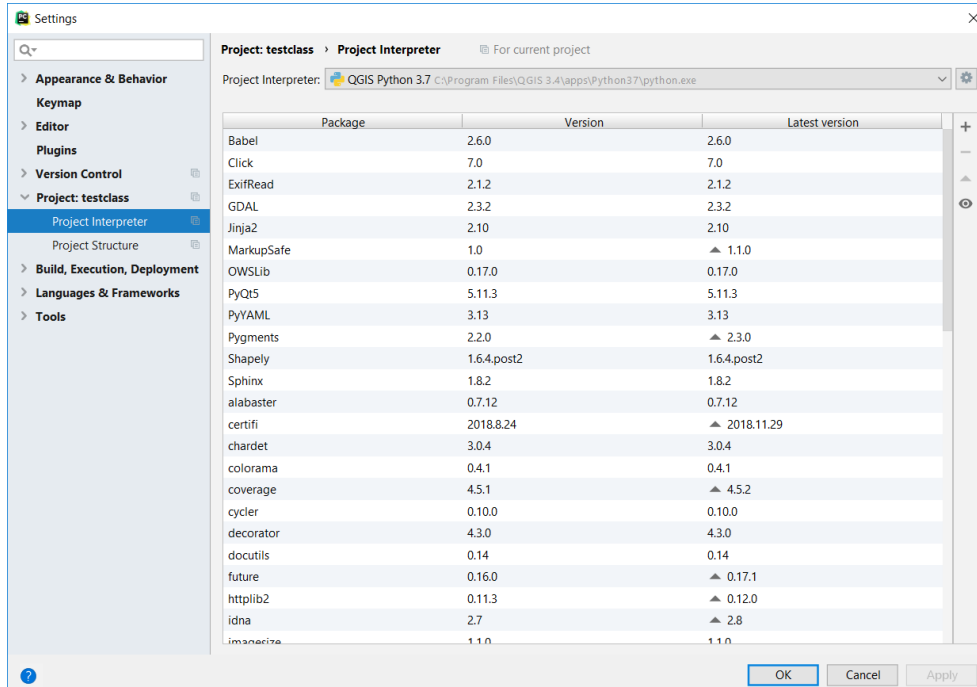
Zunächst installieren wir das QGIS Modul: «Plugin Reloader». Mit diesem müssen wir QGIS nicht jedesmal neu starten wenn das Plugin geändert wird.



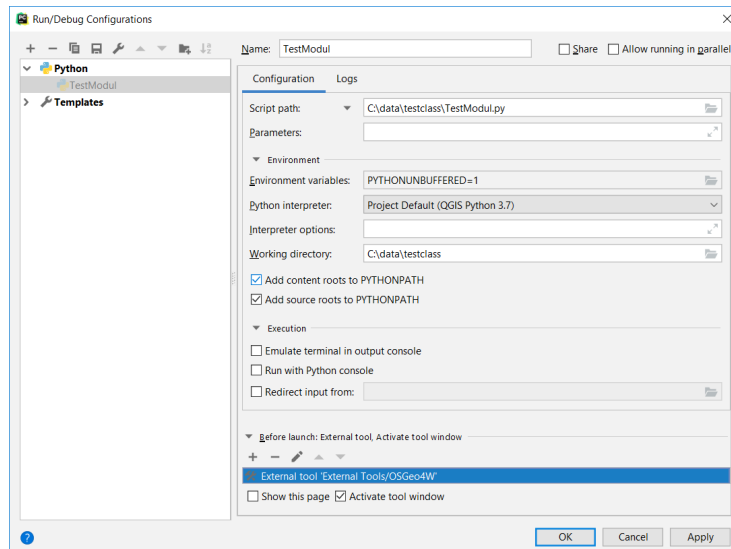
Um den Plugin Reloader zu installieren muss unter Einstellungen die Checkbox «Auch experimentelle Erweiterungen anzeigen» aktiviert sein.

10.8.2 PyCharm Konfigurieren

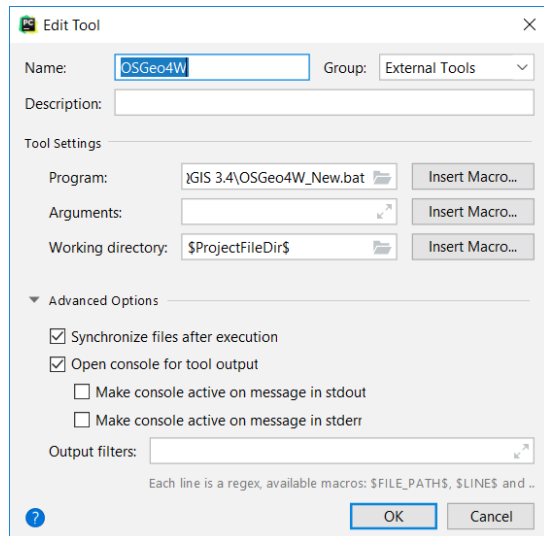
Zunächst Setzen wir den Python-Interpreter auf QGIS:



Dann die Run/Debug Konfiguration editieren. Externes Tool hinzufügen:
C:\Programme\QGIS 3.x\OSGeo4W_New.bat



Das externe Tool wird folgendermassen konfiguriert:

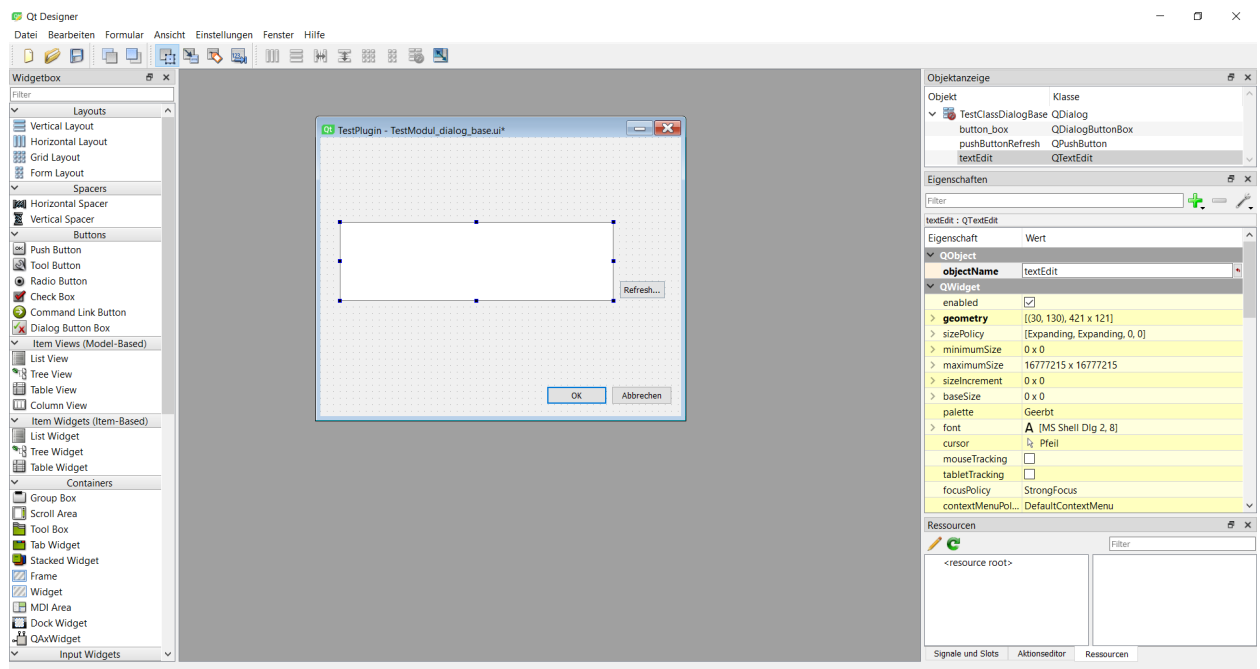


Hinweis: Mit PyCharm Professional kann man auch «Remote Debugging» betreiben. Dies sehen wir hier nicht an.

10.8.3 Entwicklung eines Plugins

Wir editieren das ui-File. Fügen ein QTextEdit (read-only) hinzu mit Objektnamen "textEdit".

Und wir fügen ein Button "refresh" hinzu mit Objektnamen "pushButtonRefresh":



Wir können auch das icon.png editieren. Dies ist das Symbol unseres Plugins innerhalb QGIS.

Wir fügen im UI auch noch einen QgsMapLayerComboBox (ganz unten) hinzu (Objektnamen: mapComboBox)

Nun erstellen wir folgenden Code im «Modulname_diglog.py» hinzu:

```
import os

from PyQt5 import uic
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QMessageBox

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'TestModul_dialog_base.ui'))

class TestClassDialog(QtWidgets.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(TestClassDialog, self).__init__(parent)
        self.setupUi(self)

        self.pushButtonRefresh.clicked.connect(self.refresh)

    def refresh(self):
        QMessageBox.information(self, "Info", "Refresh Button!")
```

Wir führen aus:

```
pb_tool deploy -y
```

Und wird starten das Plugin. Wenn keine Fehler im Code sind, funktioniert es.
Ansonsten: fix & reload!

Wir passen den Code an, so dass der aktuelle Layer aus der Dropbox angezeigt wird:

```
def refresh(self):
    layername = self.mapComboBox.currentText()
    QMessageBox.information(self, "Info", f"Refresh Button! {layername}")
```

Wir können auch weitere Dinge entwickeln:

```
def refresh(self):
    layername = self.mapComboBox.currentText()
    foundlayer = None
```

```
layers = QgsProject.instance().layerTreeRoot().children()
for i in layers:
    if i.name() == layername:
        foundlayer = i.layer()

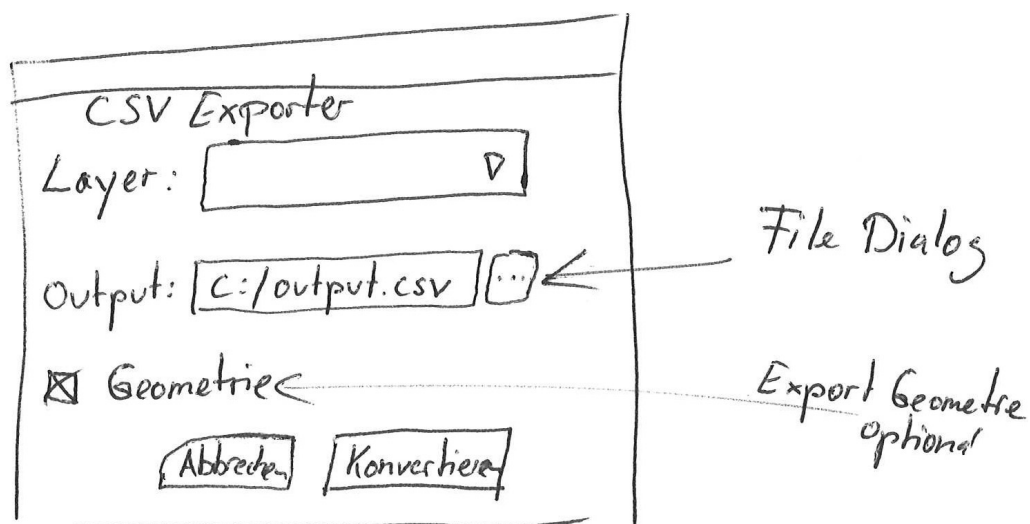
if foundlayer:
    features = foundlayer.getFeatures()
    s = ""
    for feature in features:
        s = s + feature["Name"] + "\n"

    self.textEdit.setText(s)
else:
    QMessageBox.information(self, "Info", f"ERROR!!!! LAYER GONE!!!")
    self.textEdit.setText("ERROR!!!")
```

10.8.4 Entwicklung eines komplexeren Plugins

QGIS Plugins in einem Texteditor zu entwickeln ohne den Code zu starten/testen ist oft sehr schwierig bis unmöglich, weil Fehler nicht leicht erkennbar sind.

Wir implementieren nun ein einfaches CSV-Exporter-Plugin, welches folgendermassen spezifiziert wird:



Zum Glück ist es möglich ein solches Plugin auch mit PyCharm zu entwickeln. Dazu kann folgendes .bat file verwendet werden, um PyCharm mit QGIS integration zu starten.

```
@echo off
SET OSGE04W_ROOT="C:\Program Files\QGIS 2.18\"
call "%OSGE04W_ROOT%\bin\o4w_env.bat"
call "%OSGE04W_ROOT%\apps\grass\grass-7.2.2\etc\env.bat"
@echo off
path %PATH%;%OSGE04W_ROOT%\apps\qgis\bin
path %PATH%;%OSGE04W_ROOT%\apps\grass\grass-7.2.2\lib
set PYTHONPATH=%PYTHONPATH%;%OSGE04W_ROOT%\apps\qgis\python;
set PYTHONPATH=%PYTHONPATH%;%OSGE04W_ROOT%\apps\Python27\Lib\site-packages
set QGIS_PREFIX_PATH=%OSGE04W_ROOT%\apps\qgis
start "PyCharm QGIS" /B "C:\Program Files (x86)\JetBrains\PyCharm Community Edition 5.0.4\bin\pycharm.exe" %*
```

Wir erstellen zunächst in QGIS ein Plugin „CSV Exporter“:

QGIS Plugin Builder

Class name: CSVExporter

Plugin name: CSV Exporter

Description: This plugin exports a layer as CSV

Module name: csv_exporter

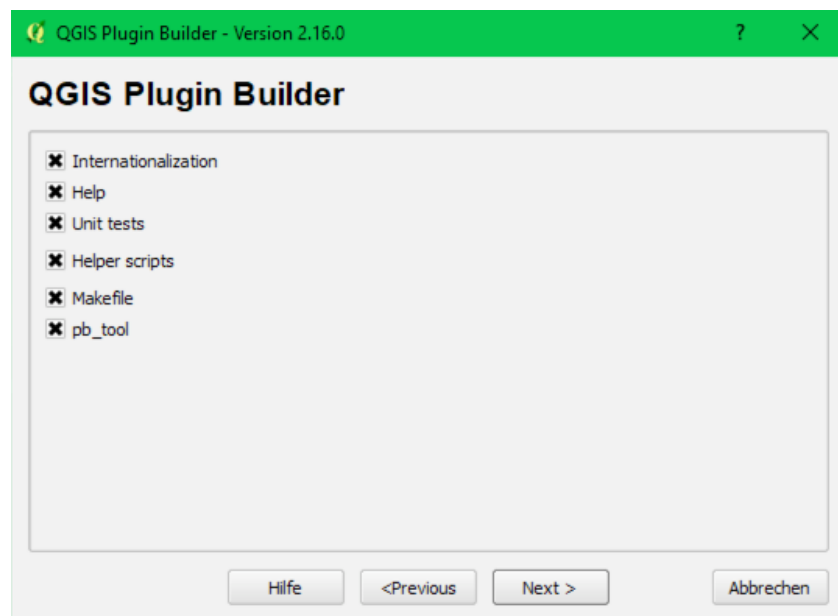
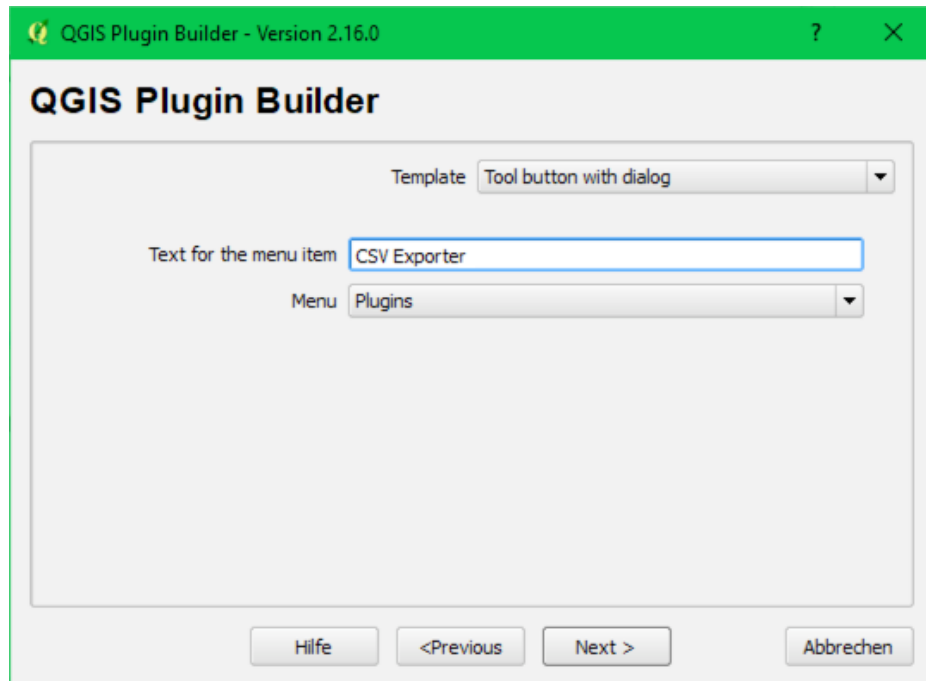
Version number: 1.0

Minimum QGIS version: 2.0

Author/Company: FHNW

Email address: martin.christen@fhnw.ch

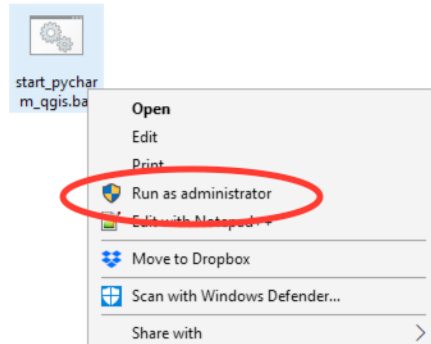
Hilfe <Previous Next > Abbrechen



wichtig ist, das „pb_tool“ aktiviert ist“

Das Plugin wird nun gespeichert (z.B. unter C:/Develop/CSV_Exporter)

Danach wird PyCharm über das .bat File **als Administrator gestartet**. Dann öffnen wir das gerade erstellte Projekt (CSVExporter). Dies ist notwendig, da wir neue Python Module installieren wollen. Später können wir das .bat File auch ohne Administrator Rechte ausführen.



Ist das Projekt geladen, gehen wir in die „Python Console“:
Dort sollte jetzt “Python 2.7.5” stehen.

Danach wechseln wir in das **Terminal**.

Nun müssen wir pip installieren. Dies geschieht mit:

```
pip install requests==2.5.3
python -m pip install --upgrade pip setuptools wheel
```

Im Terminal wird nun das pb_tool (Publish Tool) installiert (dauert eine Weile)

```
python -m pip install pb_tool
```

Remote Debugging:

```
python -m pip pydevd
```

Nun wird mit dem Qt Designer das GUI erstellt:



Die Objektnamen für Checkbox, Combobox und LineEdit sind: geometrie, layer, filename

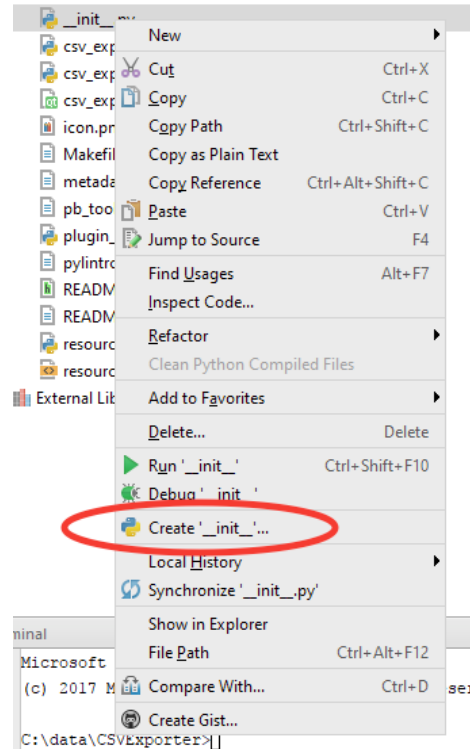
Nun starten wir PyCharm neu (bat-File ohne Administrator-Rechte)

Wir machen einen ersten deploy. Im Terminal geben wir ein:

```
pb_tool deploy
```

Dies installiert das Plugin.

Ein „Rechtsklick“ auf das File `__init__.py` öffnet das Popupmenu, bei dem wir „Create `__init__`“ wählen:



Und dort erstellen wir bei „Before launch: External Tool“ (mit +) zwei neue externe Tools:

Program: `pb_tool`
Parameters: `deploy -y`
Working Directory: `$ProjectFileDir$`

Program: `qgis`
Parameters: `--nologo -project c:/data/projekt.qgs`
Working Directory: `$ProjectFileDir$`

Nun starten wir das Programm.

Bei Start des Plugins sehen wir das Fenster...

Im Fuke csv_exporter.py fügen wir bei den imports folgendes hinzu (wir benötigen iface)

```
from qgis.utils import iface
```

In der Methode “run()” ergänzen wir:

```
layers = iface.legendInterface().layers()
layer_list = []
for layer in layers:
    layer_list.append(layer.name())

self.dlg.layer.clear()
self.dlg.layer.addItem(layer_list)
```

Im csv_exporter_dialog können wir dann die Logik für die Fileauswahl implementieren und danach das CSV File speichern (vgl. Kapitel 14.1.3)