6 Arbeiten mit dem QtDesigner

Eine GUI-Anwendung zu entwickeln ist nicht nur die Arbeit eines Software-Entwicklers. Es gibt GUI-Designer, welche sich auf diese Aufgabe spezialisiert haben. Auch das Verändern vom GUI (z.B. neuanordnen von GUI-Elementen) sollte – gerade bei grösseren Projekten - nicht abhängig von Python-Code sein. Ein weiterer Aspekt ist die Mehrsprachigkeit von Applikationen: Es sollte auch für einen nicht-Programmierer recht einfach sein eine Applikation in eine andere Sprache zu übersetzen, also z.B. von Englisch auf Chinesisch.
PyQt hat zahlreichen Tools, welche diese Aufgaben einfacher lösen können. In diesem Kapitel sehen wir uns diese ein wenig an.

Der **Qt-Designer** ist eine grafische Anwendung zum Erstellen von Benutzeroberflächen. Das Tool erlaubt das „zusammenklicken“ einer GUI. Mit dem **uic Tool** wandeln wir die Benutzeroberfläche, welche wir mit dem Qt-Designer erstellt haben in Python-Code um.

6.1 Starten des Qt-Designer

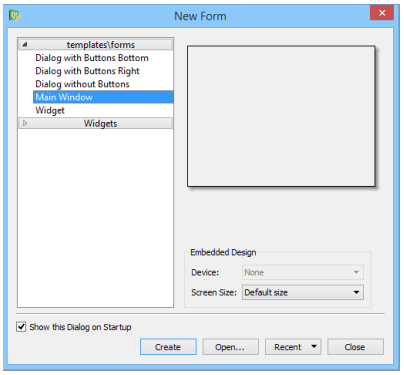
Der Qt-Designer befindet sich unter Windows im Python Ordner unter Anaconda3/Library/bin
Wo genau designer.exe ist kann mit diesem Programm herausgefunden werden:

```
import sys
import os
s = os.path.split(sys.executable)
os.startfile(s[0] + "/Library/bin")
```

6.2 Das erste Projekt mit Qt-Designer

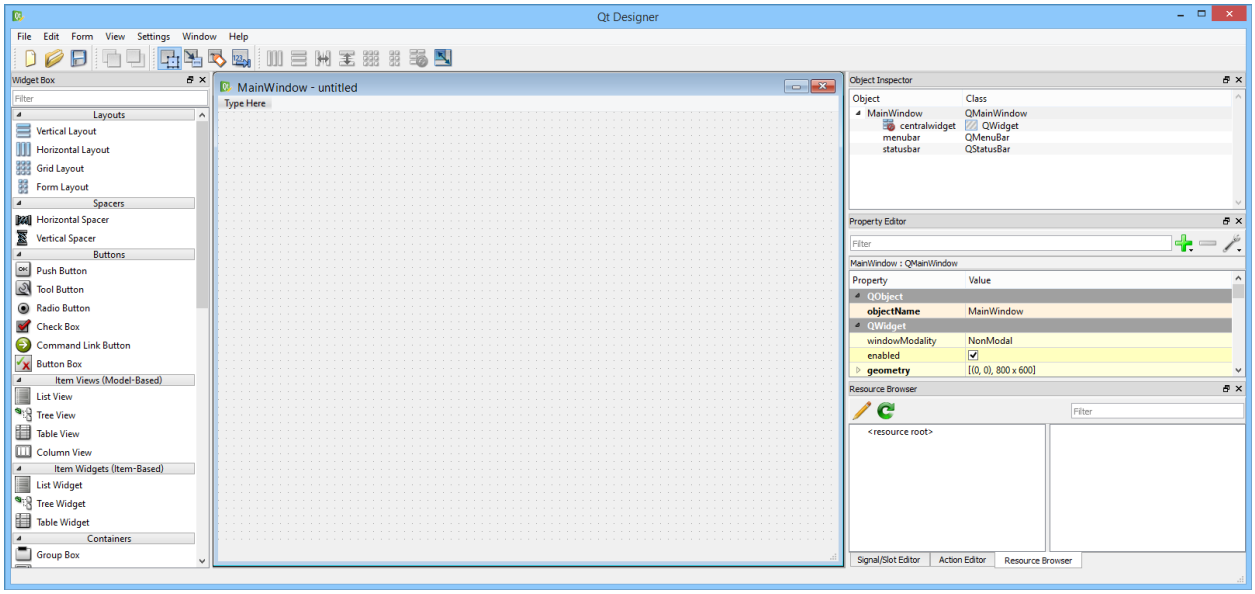
Zunächst erstellen wir mit PyCharm ein neues Projekt. Danach öffnen wir den Qt-Designer.

Nach dem Öffnen des Qt-Designer kann im Startup Dialog ein „Main Window“ gewählt werden. Mit diesem erstellen wir ein Hauptfenster.



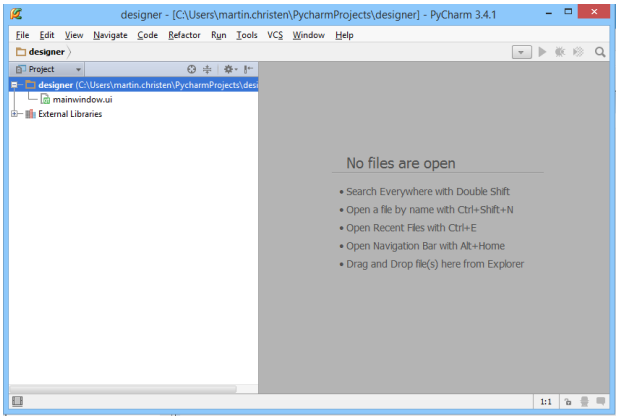
Der Startup Dialog von Qt-Designer

Es erscheint ein leeres Fenster, welches unter Windows so aussehen sollte:



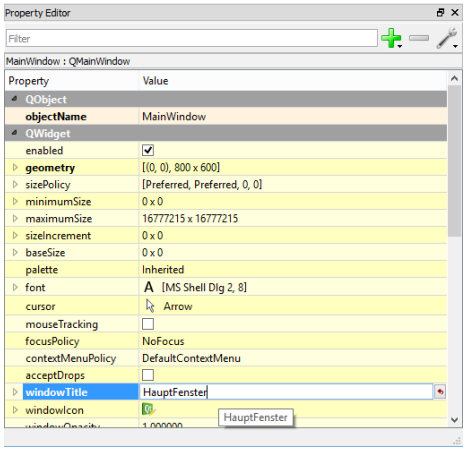
Der Qt-Designer mit einem leeren QMainWindow

Nun können wir das Projekt bereits abspeichern:
Das *.ui File - wir nennen es `mainwindow.ui` - soll nun beim zuvor erstellen PyCharm Projekt gespeichert werden.



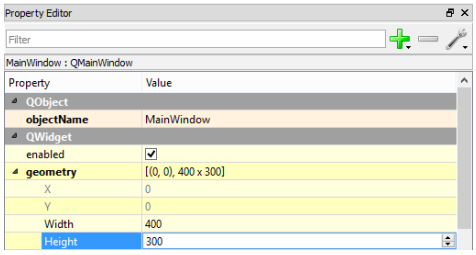
PyCharm mit einem ui-File

Nach Klick auf das „MainWindow“ können wir den Namen des Fensters editieren. Dies geschieht durch Ändern von `windowTitle` im **Property Editor**:



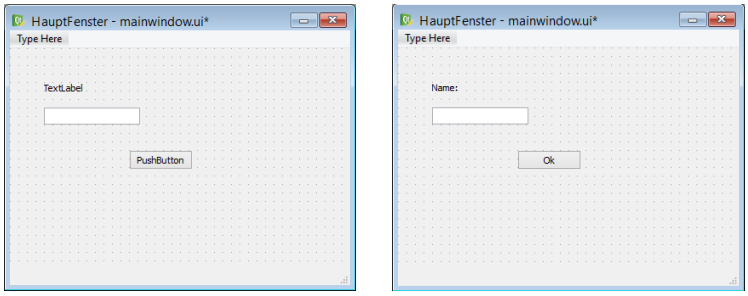
Setzen des Fenster-Titels

Danach Ändern wir die Fenstergrösse auf 400x300. Dies geschieht durch Ändern von `geometry`:



Ändern der Fenstergrösse

Nun können wir im MainWindow einen „Push Button“, ein „Label“ und ein „Text Edit“ hinzufügen. Dies geschieht indem wir die entsprechenden Widgets mit Drag&Drop von der Widget Box in das Hauptfenster ziehen. Das Resultat kann zum Beispiel so aussehen:



Widgets auf dem Main-Window vor und nach der Umbenennung

Durch Doppelklick auf den Label oder auf den PushButton kann der Name geändert werden. Wir ändern den Label auf „**Name**“ und den Push Button auf „**Ok**“.

6.3 Ui-File in Code umwandeln

Mit dem Tool pyuic5 können wir aus einem UI-File direkt Code erstelleb.

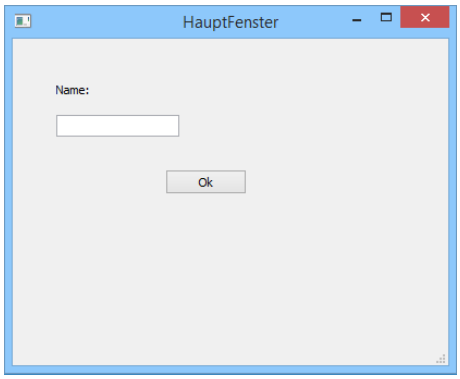
```
pyuic5 -x uifile.ui -o mypythoncode.py
```

6.4 Ui-Files direkt im Code verwenden

Ui files können auch direkt verwendet werden.

```python
import sys
from PyQt5.QtWidgets import *
from PyQt5.uic import *

# Eine Qt-Applikation erstellen:
app = QApplication(sys.argv)

window = loadUi("uifile.ui") # GUI aus ui-File laden:
window.show()  # Fenster anzeigen

# Application-Loop starten
app.exec()
```

Und wir sehen, dass unser zuvor erstelltes GUI nun korrekt dargestellt wird:



Das im Qt-Designer erstellte GUI als Python Programm

Nun können wir das GUI mit dem Qt-Designer einfach verändern, und die Änderungen sind im Programm direkt sichtbar!

6.5 Verwenden von Layouts und Signals/Slots

Die geladenen UI-Files agieren wie die in den vorherigen Kapiteln erzeugten Klassen. Mit „connect“ können Interaktionen erstellt werden und mit den im UI-Designer definierten Namen kann direkt auf die Widget-Instanz zugegriffen werden.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.uic import *

def buttonClick():
    print("Button Clicked")

# Eine Qt-Applikation erstellen:
app = QApplication(sys.argv)

mainwindow = loadUi("uifile.ui")

# Nun kann auf die im UI-Designer gesetzten Namen zugegriffen werden
mainwindow.MeinLabel.setText("Dies ist ein Label!")
mainwindow.pushButtonOk.clicked.connect(buttonClick)

mainwindow.show()  # Fenster anzeigen

# Application-Loop starten
app.exec()
```

6.6 Objektorientiert arbeiten

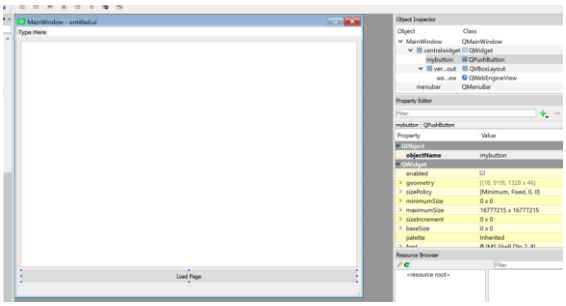
Wir können auch loadUi innerhalb des Konstruktors ausführen, und so ganz gewohnt objektorientiert arbeiten. Danach können alle connects wie gewohnt gemacht werden.

```
from PyQt5 import uic
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
import sys

class UiWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('uifile.ui', self)
        self.show()

app = QApplication(sys.argv)
window = UiWindow()
app.exec()
```

6.7 Erstellen eines Webbrowsers

Dies erfordert eventuell die installation von PyQtWebEngine (danach restart Designer)

```
conda install pyqtwebengine
```



```
from PyQt5.uic import *
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtWebEngineWidgets import QWebEngineView

class UiWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi('uifile.ui', self)
        self.show()

        self.mybutton.clicked.connect(self.loadPage)


    def loadPage(self):
        #self.webEngineView.setHtml("<h1>Hello World</h1>")
        self.webEngineView.load(QUrl("http://www.fhnw.ch/"))

app = QApplication([])
window = UiWindow()
app.exec()
```

Die QWebEngine View hat einige wichtige Signale, wie z.B.

```
loadFinished(ok : bool)
loadProgress(progress : int)
urlChanged(url: QUrl)
```

Es ist u.a. auch möglich JavaScript auszuführen (am besten nach loadFinished):

```
self.webEngineView.page().runJavaScript("document.getElementsByName('bla')[0].value", self.store_value)
```