Урок QtDesigner

QtDesigner, pyuic, два способа подключения uic-файла

Аннотация

В этом уроке разбирается популярный способ создания графических интерфейсов — с помощью программы QtDesigner.

1. Установка QtDesigner и первый запуск

Когда на прошлом занятии мы создавали интерфейсы «руками» и размещали виджеты «на глазок», вы наверняка подумали, что есть какой-то более простой способ. И он действительно есть. Это программа **QtDesigner**, которая включена в сборку PyQT5. Но для ее использования необходимо установить библиотек**QtDesigner**у `pyqt5-tools`.

```
pip install pyqt5-tools
```

Теперь программа находится на вашем компьютере по адресу:

```
Путь_к_папке_где_установлен_Python\Lib\site-packages\qt5_applications\Qt\bin\designer.exe
```

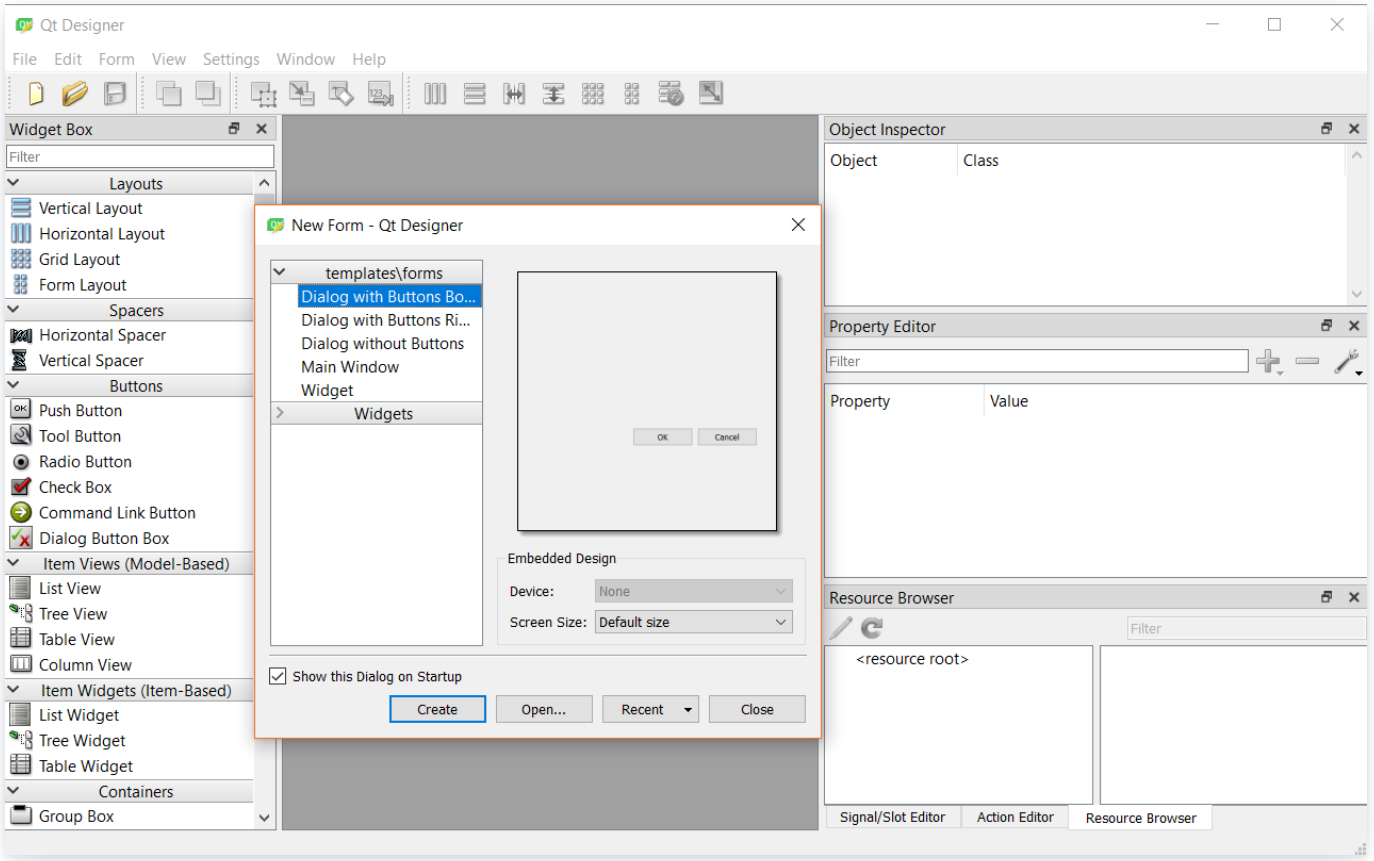
или (для старых версий):

```
Путь_к_папке_где_установлен_Python\Lib\site-packages\pyqt5_tools\designer.exe
```

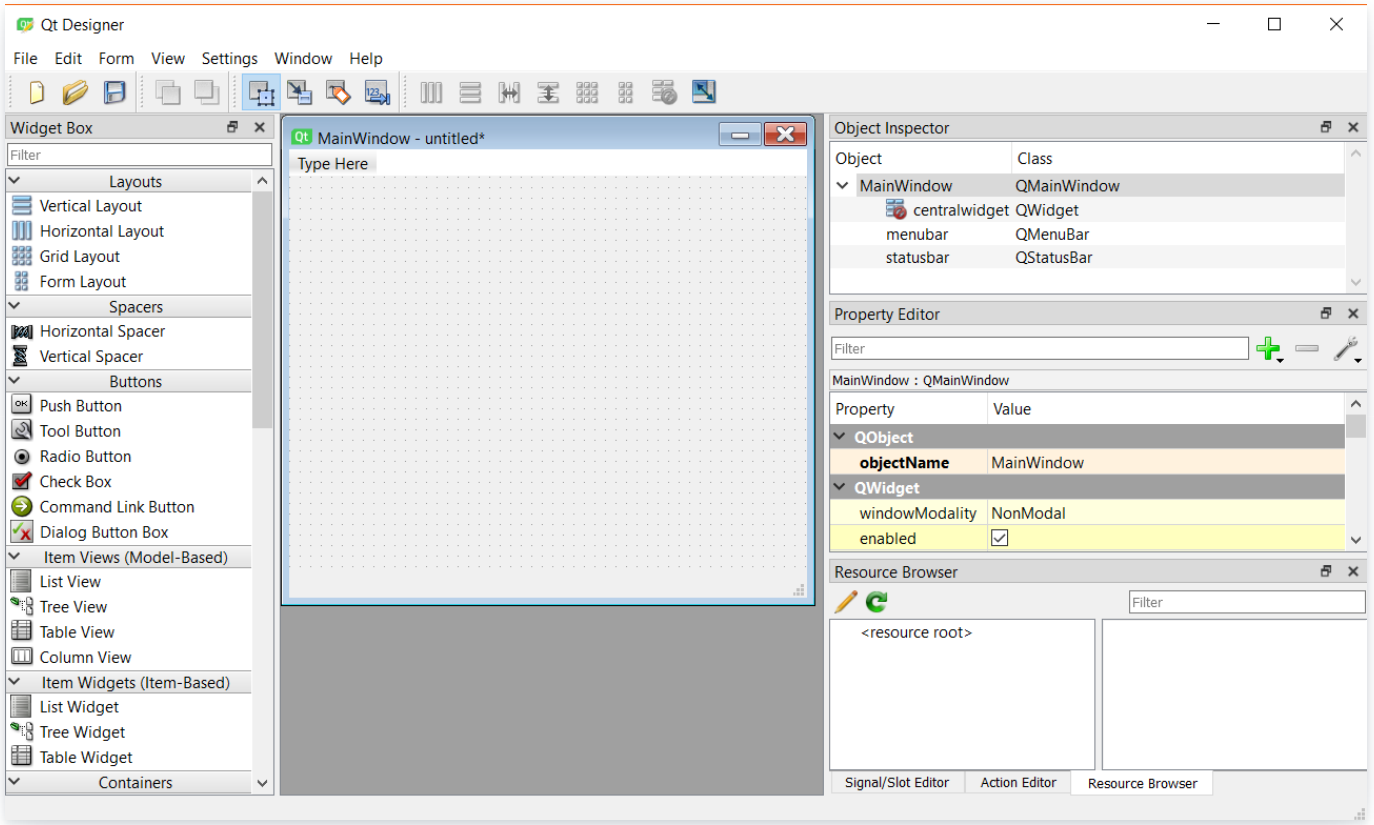
Создайте ярлык к этой программе, поскольку использовать ее придется часто.

Так же можно скачать и установить программу **QtDesigner** в виде отдельного приложения. Для этого перейдите по **ссылке** и скачайте установщик приложения дла вашей операционной системы.

Давайте запустим программу и посмотрим, что в ней можно сделать.



При запуске открывается окно с предложением выбрать шаблон для формы или виджет, на основе которого мы будем делать свой интерфейс. Выберем **Main Window**. Откроется пустое окно.



Теперь рассмотрим, что у нас есть, кроме пустой формы, которую мы будем заполнять.

Слева — меню виджетов, **Widget Box**. В нем они сгруппированы в зависимости от их функциональности. Отдельно кнопки, отдельно виджеты для ввода данных и так далее.

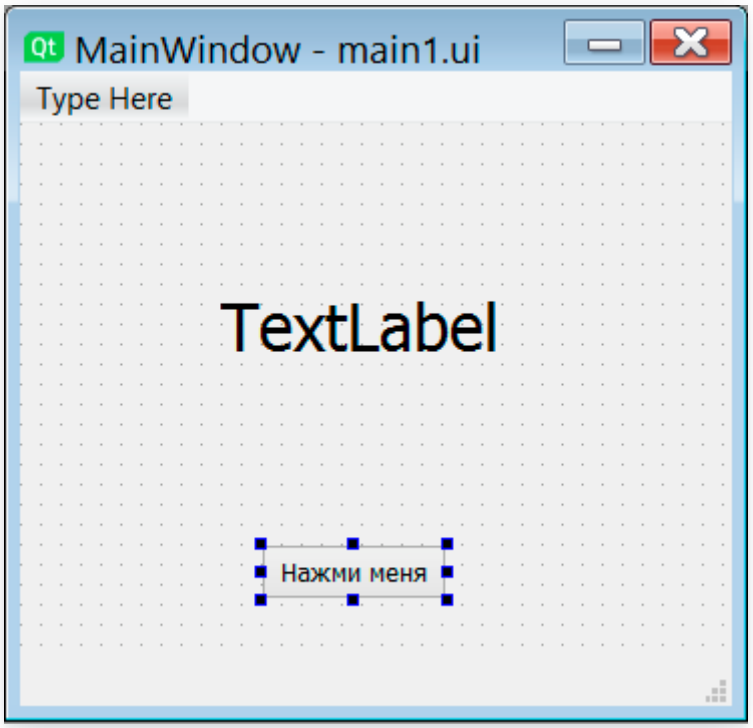
Справа — Инспектор объектов (**Object Inspector**), Редактор свойств (**Property Editor**) и Браузер ресурсов (**Resourse Browser**). Остановимся на первых двух. В **Инспекторе объектов** отображается информация об используемых виджетах. Для каждого виджета указывается его имя и класс. Кроме того, можно увидеть иерархическую структуру всего интерфейса.

Чтобы разместить виджет на форме, его надо просто перетащить из меню виджетов. При этом информация об этом виджете автоматически появится в Инспекторе объектов.

Редактор свойств помогает изменять значения тех или иных атрибутов виджета (например, текст или размер). Расположение свойств меняется кнопкой с изображением гаечного ключа.

Попробуйте создать простейший интерфейс из кнопки (`PushButton`) и текстового поля (`TextLabel`), поиграйте с **Редактором свойств**, разберитесь, где поменять название и размеры кнопки, как изменить шрифт в `TextLabel`. После того как у вас вышло что-то похожее на картинку, ниже сохраните полученный дизайн. Для сохранения в меню **Файл** выберите вкладку **Сохранить как**, найдите папку проекта и впишите имя.

Помните: название объекта (атрибут `objectName`) и текст, который может быть на нем показан (атрибут `text`), — это разные вещи.



Давайте посмотрим, как выглядит наш дизайн с точки зрения компьютера. Для этого откроем созданный нами файл с помощью любого текстового редактора. Лучше использовать не просто Блокнот, а, например, SublimeText, Notepad++ или VS Code. Содержимое файла будет примерно вот таким:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>554</width>
    <height>379</height>
   </rect>
  </property>
```

```
<property name="windowTitle">
 <string>MainWindow</string>
</property>
<widget class="QWidget" name="centralwidget">
 <widget class="QPushButton" name="pushButton">
  <property name="geometry">
   <rect>
    <x>210</x>
    <y>200</y>
    <width>93</width>
    <height>28</height>
   </rect>
  </property>
  <property name="text">
   <string>Нажми меня</string>
  </property>
  <property name="checkable">
   <bool>true</bool>
  </property>
 </widget>
 <widget class="QLabel" name="label">
  <property name="geometry">
   <rect>
    <x>130</x>
    <y>80</y>
    <width>281</width>
    <height>61</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <pointsize>24</pointsize>
   </font>
  </property>
  <property name="text">
   <string>Текст на метке</string>
  </property>
  <property name="textFormat">
   <enum>Qt::AutoText</enum>
  </property>
 </widget>
</widget>
<widget class="QMenuBar" name="menubar">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>554</width>
   <height>26</height>
```

```
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Если вы послушали наш совет и посмотрели в течение лета материалы по HTML, то наверняка увидите знакомые конструкции. Конечно, это не HTML, а язык разметки XML. Если присмотреться, мы увидим, что внутри этого документа описаны все наши виджеты и их свойства, а также показана их вложенность друг в друга.

2. Подключение дизайна к программе

Теперь у нас есть дизайн, но нам надо подключить его к программе. Для этого есть два способа.

Способ первый: загрузка ui-файла

```
import sys

from PyQt5 import uic  # Импортируем uic
from PyQt5.QtWidgets import QApplication, QMainWindow


class MyWidget(QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('01.ui', self)  # Загружаем дизайн
        self.pushButton.clicked.connect(self.run)
        # Обратите внимание: имя элемента такое же как в QTDesigner

    def run(self):
        self.label.setText("OK")
        # Имя элемента совпадает с objectName в QTDesigner


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = MyWidget()
    ex.show()
    sys.exit(app.exec_())
```

Для загрузки ui-файла импортируйте класс `uic`, а затем в конструкторе вызовите метод `loadUi`, где одним из параметров указывается файл с интерфейсом. В нашем примере он называется 01.ui и лежит в той же папке, что и запускаемый нами скрипт.

После выполнения метода `loadUi` все виджеты становятся полями класса, имена для которых мы задали в редакторе свойств. Затем можно работать с ними точно так же, как в предыдущем уроке. Что мы и делаем, подключая обработчик нажатия кнопки.

Способ второй: использование pyuic

Второй способ — конвертирование ui-файла в класс Python. Для этого нужна консольная утилита **pyuic5**.

Чтобы ею воспользоваться, нужно открыть командную строку (терминал), перейти в ту папку, где лежит ваш ui-файл, и выполнить следующую команду:

```
pyuic5 ui_file.ui -o ui_file.py
```

Давайте посмотрим, что находится внутри получившегося файла. (В зависимости от того, что вы сделали в QtDesigner, содержимое файла будет отличаться, но смысл останется неизменным.)

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

# Form implementation generated from reading ui file '01.ui'
#
# Created by: PyQt5 UI code generator 5.15.1
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(554, 379)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(180, 200, 111, 28))
        self.pushButton.setCheckable(True)
        self.pushButton.setObjectName("pushButton")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(130, 80, 281, 61))
        font = QtGui.QFont()
        font.setPointSize(24)
        self.label.setFont(font)
        self.label.setTextFormat(QtCore.Qt.AutoText)
        self.label.setObjectName("label")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 554, 28))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
```

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

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton.setText(_translate("MainWindow", "Нажми меня"))
        self.label.setText(_translate("MainWindow", "Текст на метке"))
```

Очень похоже на то, что мы делали на прошлом уроке, не правда ли? Только весь код написан не в инициализаторе, а в методе `setupUi()`. Утилита pyuic5 конвертирует XML-описание разметки из ui-файла в класс Python с кодом, создающим точно такой же интерфейс.

В получившемся файле нет кода для запуска приложения и никаких обработчиков событий. Есть соблазн дописать код в получившийся файл, но делать этого не стоит, и вот почему: никто из нас не идеален и не может написать интерфейс сразу так, чтобы в него никогда потом не пришлось вносить изменения. Поэтому если мы перемешаем код интерфейса и наш код с логикой, а потом поправим дизайн, после конвертации вся наша работа пропадет. К счастью, есть простой выход из ситуации.

В блоке по ООП мы говорили про множественное наследование, и говорили, что использовать его надо аккуратно и далеко не всегда. Ситуация с разделением интерфейса и логики исполнения — как раз звездный час для множественного наследования.

Давайте создадим новый py-файл рядом с классом, который получился после конвертации интерфейса, вот с таким кодом:

```
import sys

from PyQt5.QtWidgets import QApplication, QMainWindow
from ui_01 import Ui_MainWindow


# Наследуемся от виджета из PyQt5.QtWidgets и от класса с интерфейсом
class MyWidget(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        # Вызываем метод для загрузки интерфейса из класса Ui_MainWindow,
        # остальное без изменений
        self.setupUi(self)
        self.pushButton.clicked.connect(self.run)

    def run(self):
        self.label.setText("OK")


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = MyWidget()
    ex.show()
    sys.exit(app.exec_())
```

Предками нашего класса `MyWidget` являются и `QMainWindow`, и `Ui_MainWindow`. От первого унаследованы методы, а от второго — дизайн. В остальном работа схожа: мы вызываем метод `setupUi` из `Ui_MainWindow`, а затем просто работаем с полями.

Какой способ лучше?

Так какой же способ лучше? Ответ на этот вопрос неожиданный — оба метода уместны в определенных ситуациях.

Загрузка ui-файла очень удобна, когда наше приложение находится на стадии разработки и нам постоянно надо вносить какие-либо изменения в наш интерфейс. В таких случаях постоянная конвертация файла только замедляет процесс разработки. На самом деле конвертация происходит и в этом случае: метод `uic.loadUi()` выполняет эту конвертацию каждый раз при запуске приложения (а точнее каждый раз, когда выполняется эта строчка кода), что может сильно снизить производительность приложения.

Поэтому хорошим советом будет следующий:

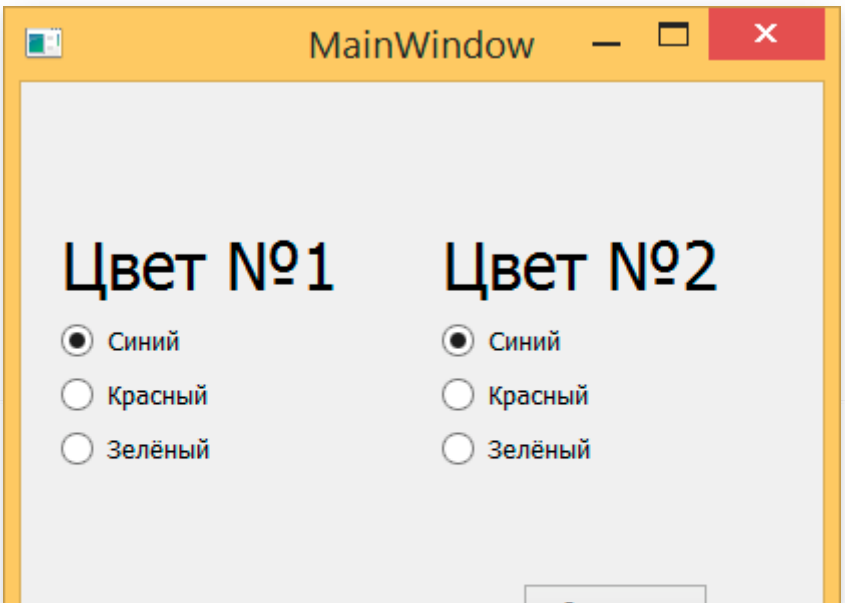
- На этапе разработки подключайте интерфейс с помощью ui-файлов
- Для релиза сконвертируйте весь интерфейс в классы Python

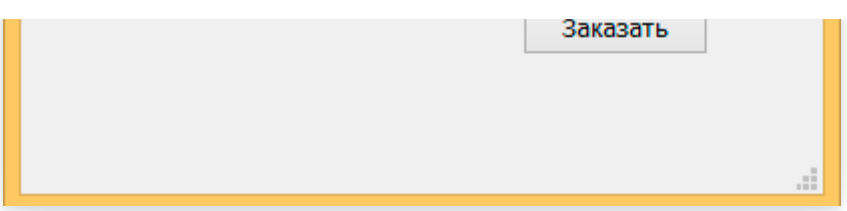
3. Размещение виджетов

Простое «накидывание» виджетов в QtDesigner или из кода работает неплохо. Проблемы начинаются, когда мы попытаемся каким-либо образом изменять размеры окна нашего приложения. В этом случае какие-то виджеты перестают частично или полностью попадать в поле зрения пользователя, и приложением становится неудобно пользоваться. Однако и с этой проблемой мы можем легко справиться, как программно, так и при помощи QtDesigner.

В QtDesigner можно размещать виджеты на экране не хаотично, а упорядоченно. Этот процесс называется разметкой. Для этого есть виджеты, которые называются **Layout** (Разметка): Vertical, Horizontal, Grid и Form Layout.

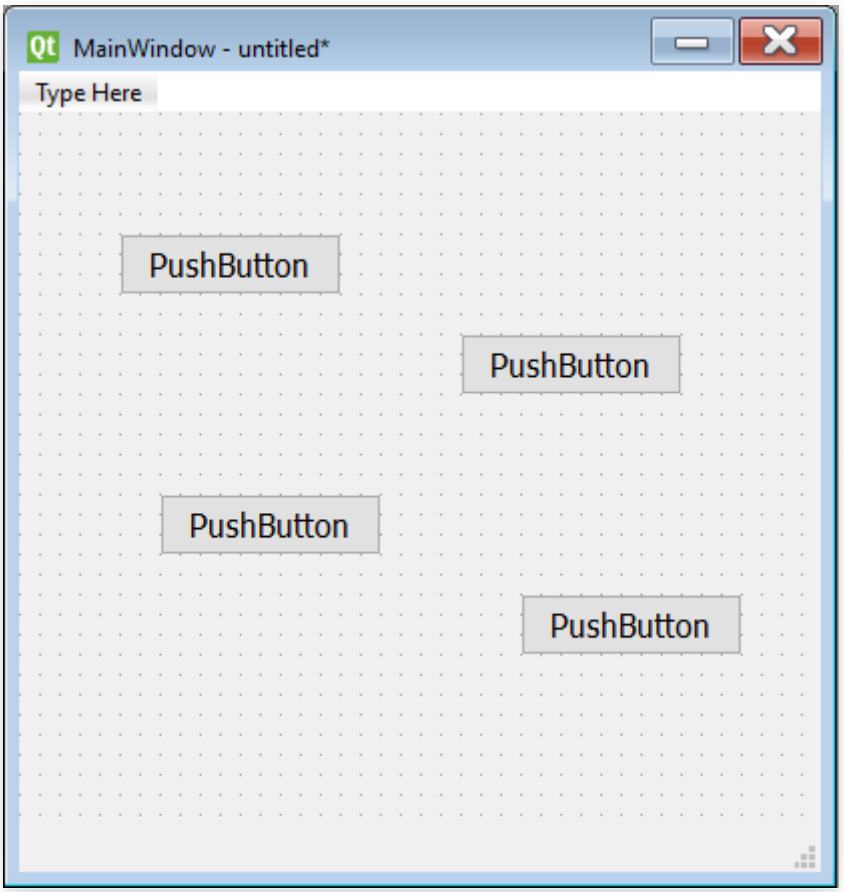
Но они нужны не только для красивого размещения элементов интерфейса, но и для создания групп из виджетов, например, для Radio Button (элемент интерфейса, который позволяет пользователю выбрать одну опцию (пункт) из предопределенного набора (группы)). Когда мы работаем с радиокнопками, можем выбрать только одну из них. А как поступить в том случае, когда у нас несколько логических групп, в каждой их которых нужно сделать выбор? Если мы просто разместим все Radio Button на нашем виджете, то никак не сможем выбрать два. А вот если часть из них поместить в какой-нибудь Layout — легко.



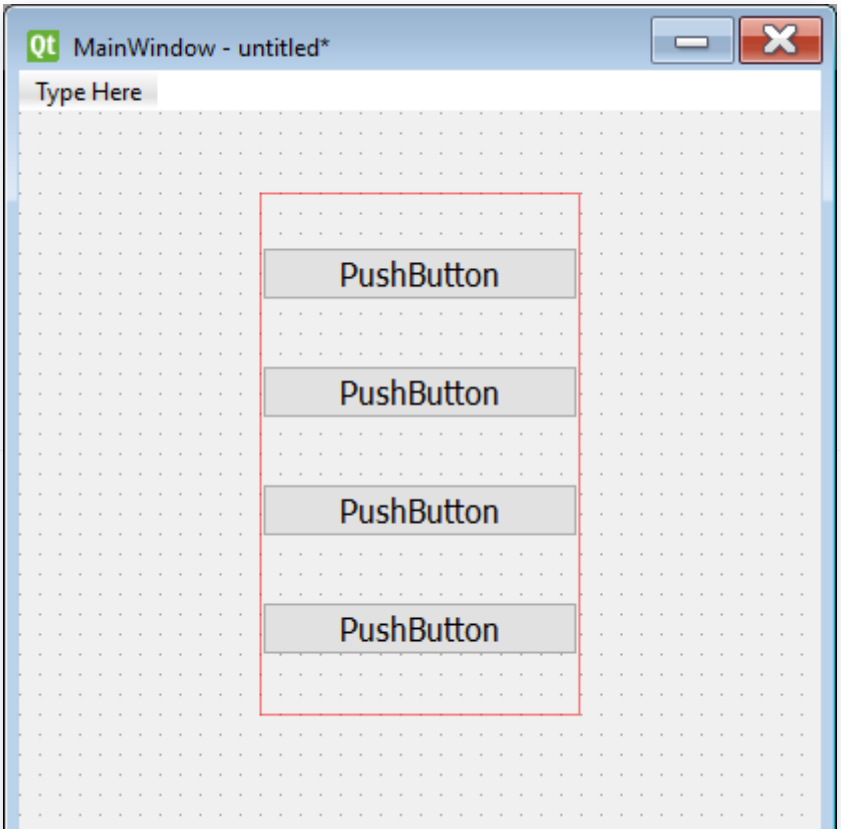


Давайте посмотрим на то, как будут выглядеть привычные нам `PushButton` при применении к ним различных layout-ов.

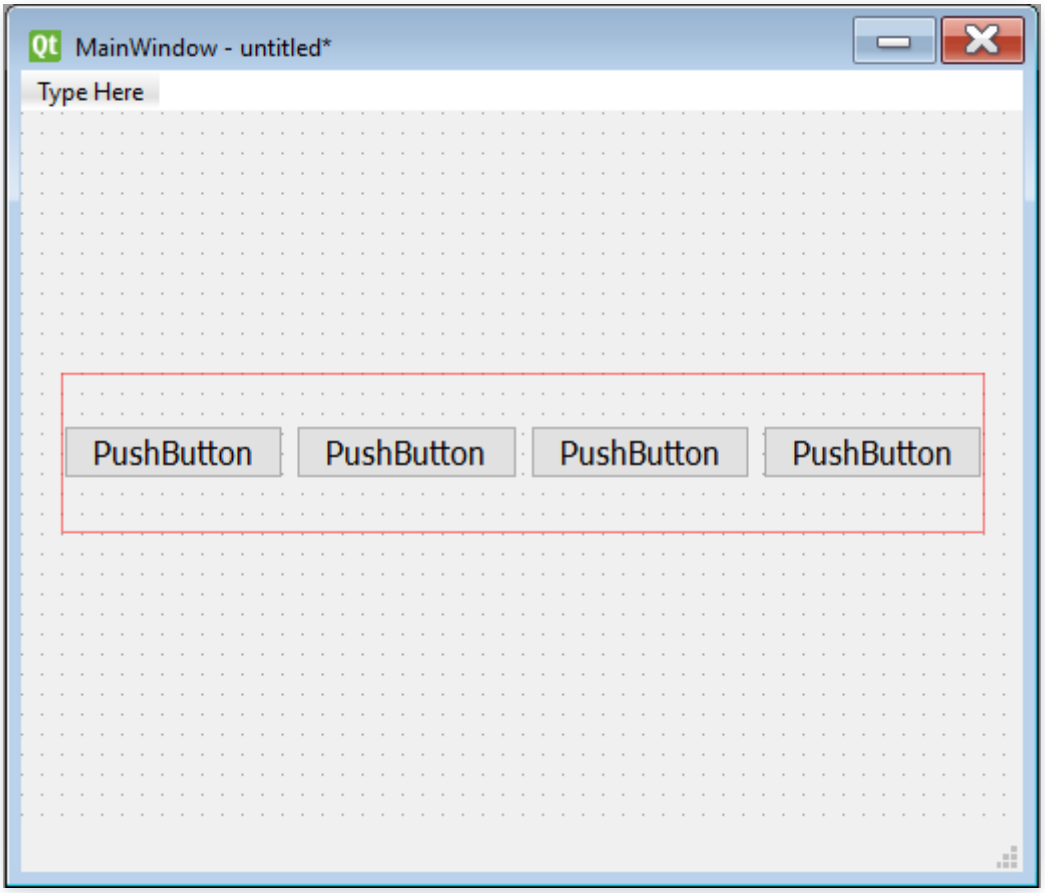
Разместим изначально наши кнопки на форме в случайном порядке.



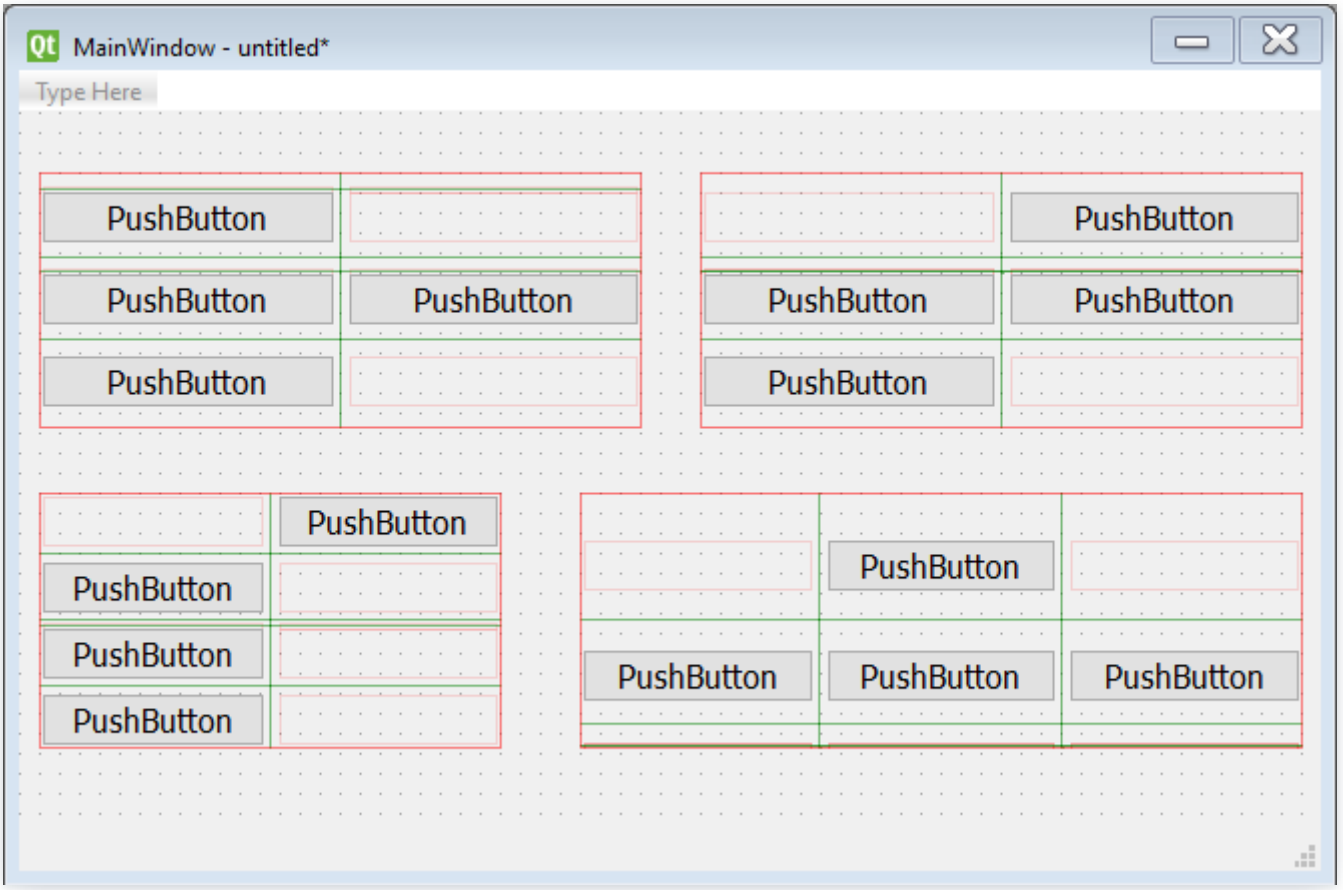
Затем поместим их внутрь **Vertical Layout**.



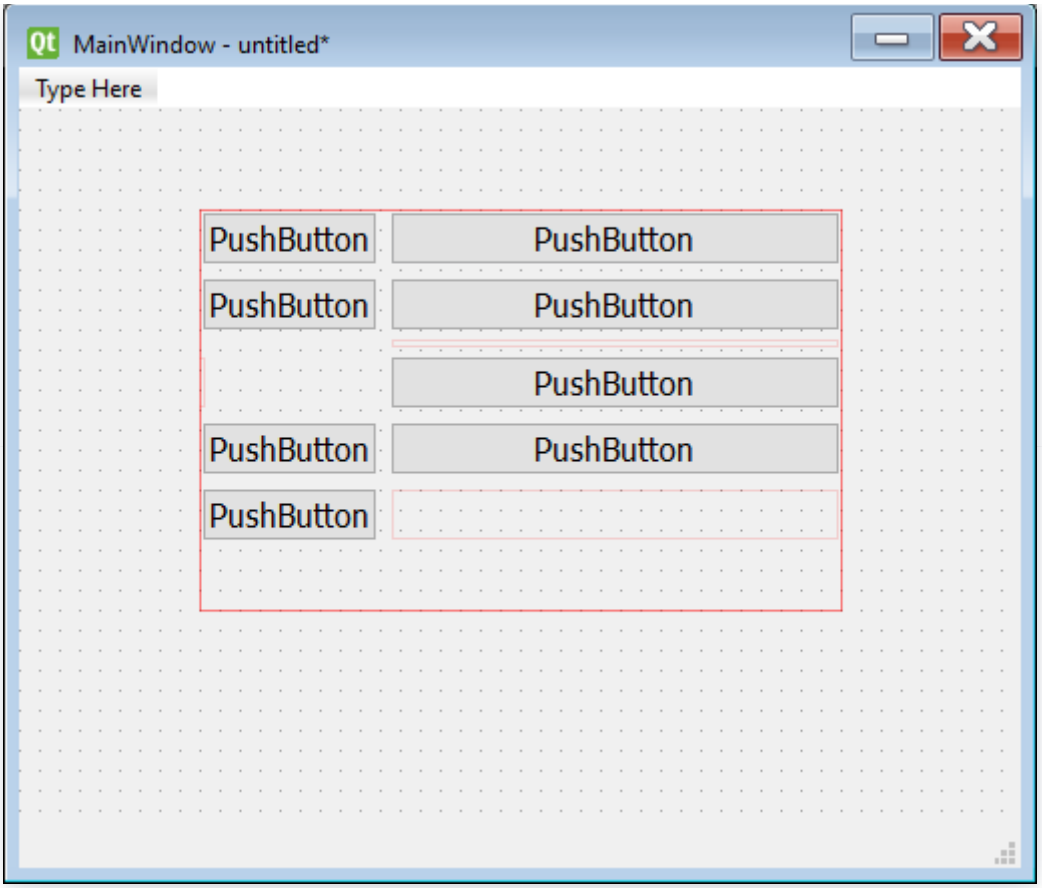
Теперь — в **Horizontal Layout**.



В случае использования **Grid Layout** появляются различные варианты:



А **Form Layout** является подвидом **Grid Layout**. У него фиксированное число столбцов, равное двум. Как можно догадаться по названию, такой layout удобнее всего использовать для создания форм, где в первый столбец размещаются подсказки для полей, а во второй — виджеты для ввода значения.



Создавать Layout можно и из кода напрямую, точно также, как мы это делали с другими виджетами. Для добавления элементов в Layout у них всех существует метод `addWidget()`, который принимает виджет, который надо разместить, и дополнительные настройки размещения. Например, чтобы добавить виджет в Grid Layout, можно написать такой код: `grid.addWidget(elem, x, y)`. В качестве параметров метода выступают виджет, а также координаты ячейки, по которым его нужно расположить. Подробнее можно почитать в **документации**.

Кроме Layout, существуют и виджеты для создания виртуальных групп (вроде `QButtonGroup`). Чаще всего они нужны, когда у нас есть много виджетов с одинаковыми функциями. Например, у нас много Radio Button, но они все отвечают за один и тот же параметр — цвет. Чтобы объединить Radio Button в группу в QtDesigner, необходимо выделить их, нажать на правую кнопку мыши, кликнуть пункт меню **Assign to button group** и выбрать либо пункт создания новой группы (**New button group**), либо уже созданную. Чтобы «повесить» на всю группу обработчик событий, нужно вызвать уже знакомый нам метод `connect`.

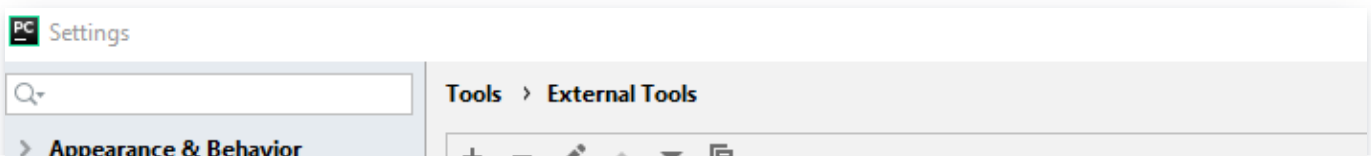
```
self.buttonGroup.buttonClicked.connect(self.run)
```

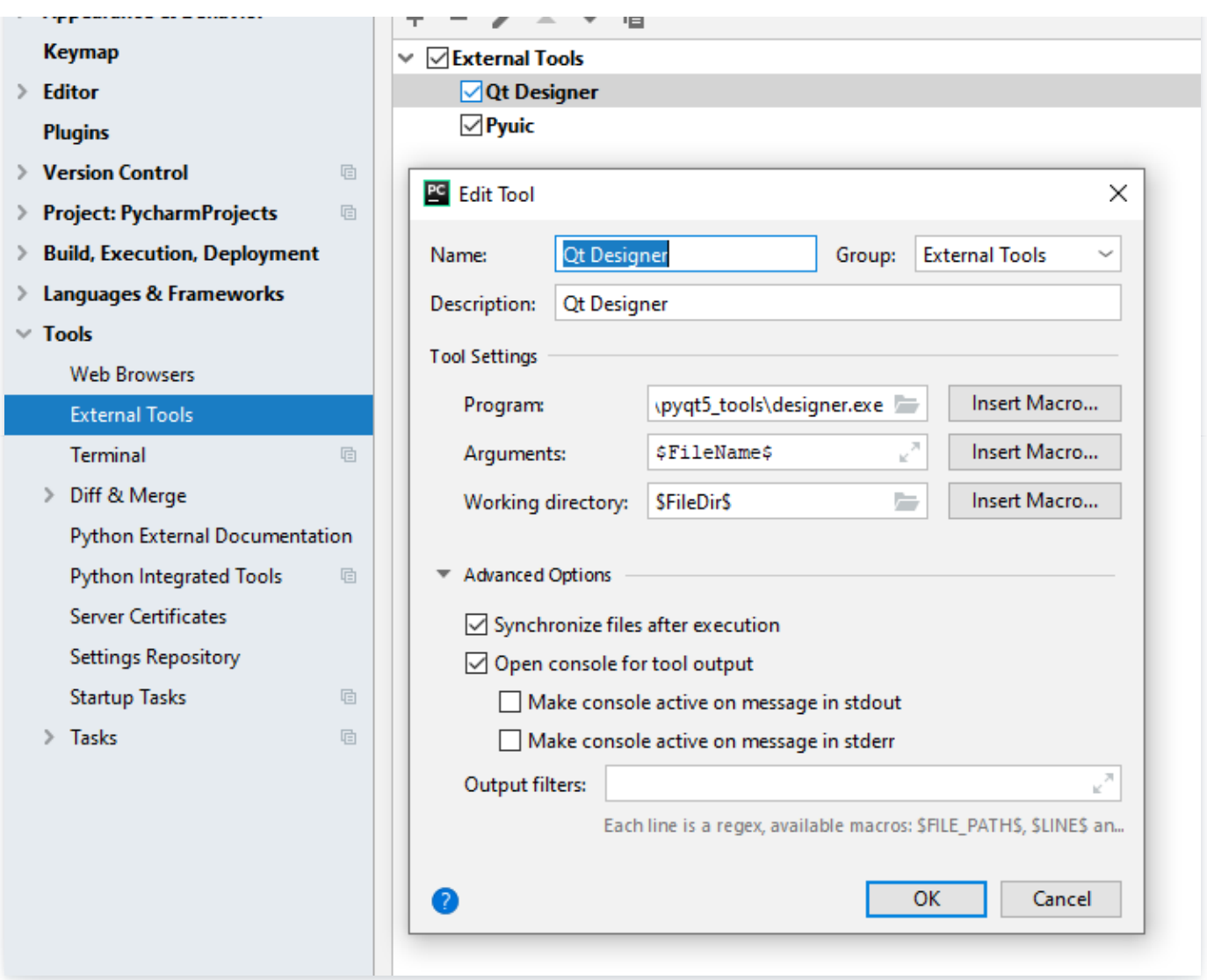
А чтобы получить список всех кнопок группы, примените метод `.buttons()`.

4. Настройка PyCharm для работы с графическим интерфейсом

Как вы, наверное, уже могли заметить, PyCharm «из коробки» не очень-то дружит с ui-файлами, а конвертация файлов, пусть и не сложная, но требует некоторых манипуляций в командной строке. Давайте попробуем немного облегчить себе жизнь.

Сначала добавим открытие ui-файлов в QtDesigner непосредственно из PyCharm. Для этого зайдем в настройки PyCharm, перейдем по пути Settings → Tools → External Tools и нажмем плюсик.

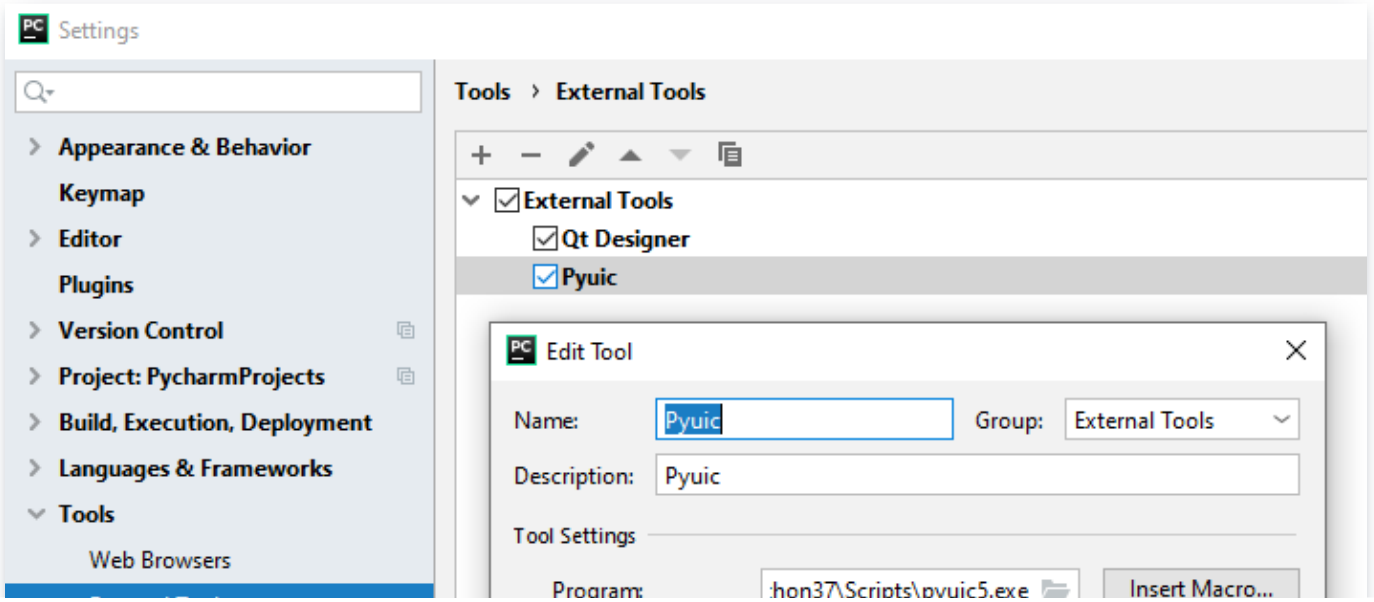


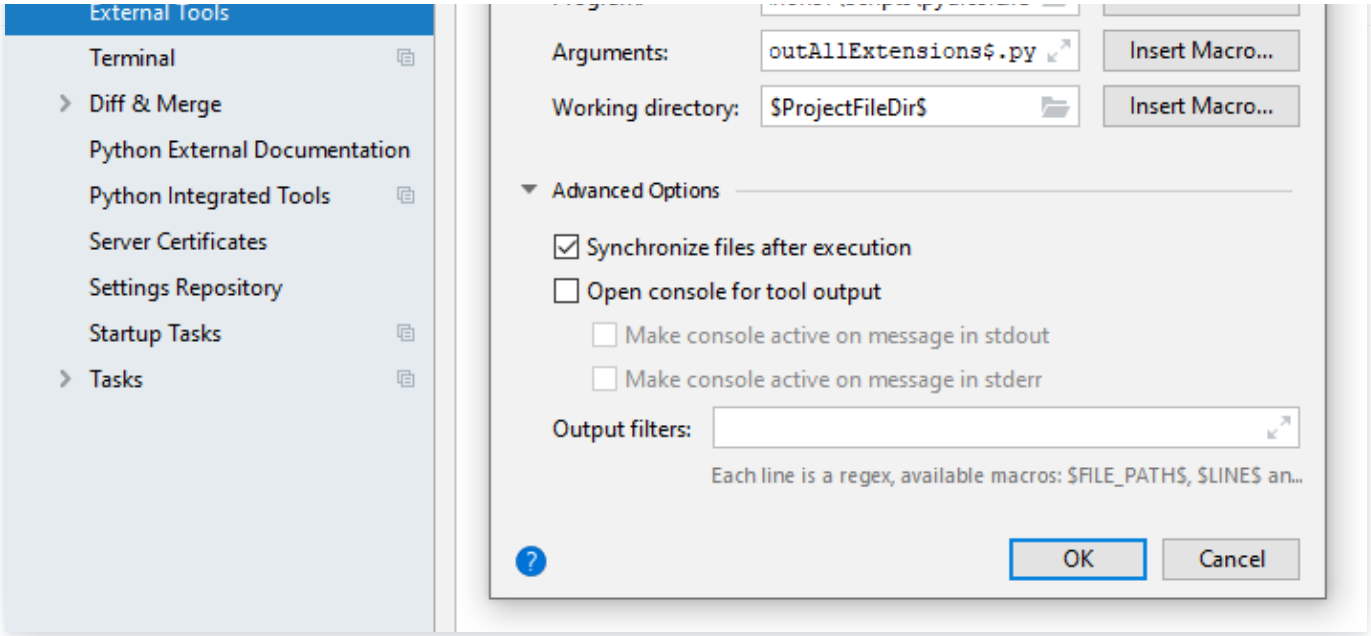


В появившейся форме введем следующую информацию:

- Name: **Qt Designer**
- Program: **Путь_к_папке_где_установлен_Python\Lib\site-packages\qt5_applications\Qt\bin\designer.exe** (Например: C:\Python37\Lib\site-packages\qt5_applications\Qt\bin\designer.exe)
- Arguments: **$FileName$**
- Working Directory: **$FileDir$**

Теперь добавим возможность быстрого конвертирования ui-файл в py-файл. Снова жмем плюсик, находясь в там же настройках (по пути Settings → Tools → External Editor):

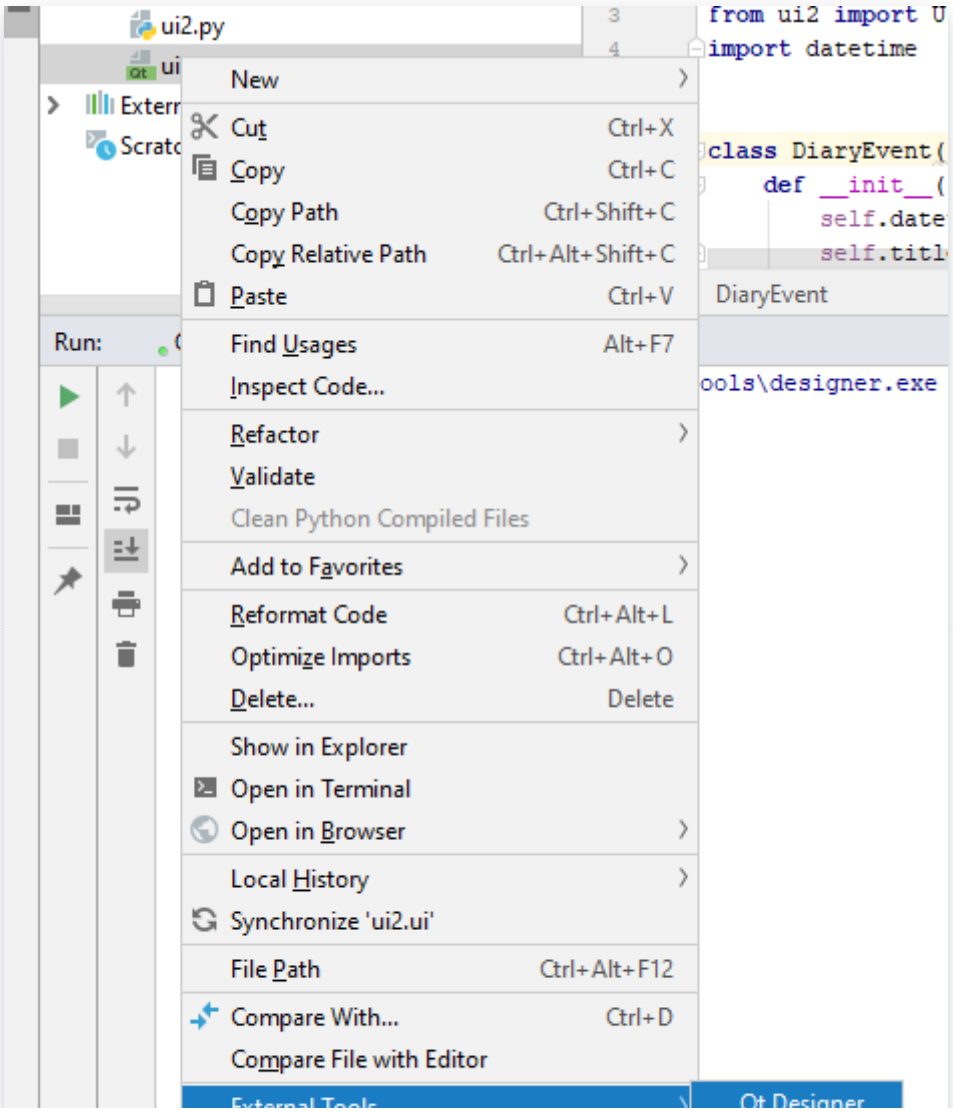


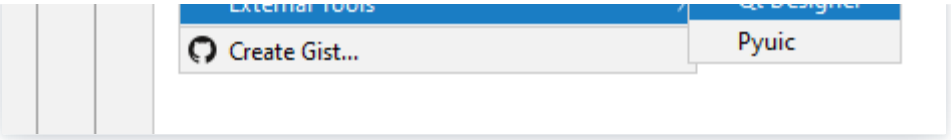


В появившейся форме введем следующую информацию:

- Name: **Pyuic**
- Program: **<Папка с питоном>\Scripts\pyuic5.exe** (Например: C:\Python37\Scripts\pyuic5.exe)
- Arguments: **$FileName$ -o $FileNameWithoutAllExtensions$.py**
- Working Directory: **$ProjectFileDir$**

Теперь мы в проекте можем нажать правой кнопкой на ui-файл и, выбрав в меню External Tools, выбрать нужную опцию: либо открыть его для редактирования в QtDesigner, либо конвертировать в py-файл.





5. Экраны с высоким разрешением (HiRes)

На экранах с высоким разрешением некоторые интерфейсы, разработанные для стандартного разрешения, могут выглядеть не корректно или мелко. Есть много способов решения этой проблемы. Вот один из них. До начала запуска основного приложения Qt и до описания своих виджетов добавьте вот такие строки:

```
from PyQt5 import QtCore, QtWidgets

if hasattr(QtCore.Qt, 'AA_EnableHighDpiScaling'):
    QtWidgets.QApplication.setAttribute(QtCore.Qt.AA_EnableHighDpiScaling, True)

if hasattr(QtCore.Qt, 'AA_UseHighDpiPixmaps'):
    QtWidgets.QApplication.setAttribute(QtCore.Qt.AA_UseHighDpiPixmaps, True)
```

6. Как сдавать задачи с интерфейсом, созданным в QtDesigner

Решения всех задач, которые проверяются автоматически, должны сдаваться в виде одного файла "*.py". Дизайн приложения QT будем встраивать в файл с основной программой так:

```
import io
import sys

from PyQt5 import uic  # Импортируем uic
from PyQt5.QtWidgets import QApplication, QMainWindow

template = """<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>554</width>
    <height>379</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>180</x>
```

```
      <y>200</y>
      <width>111</width>
      <height>28</height>
     </rect>
    </property>
    <property name="text">
     <string>Нажми меня</string>
    </property>
    <property name="checkable">
     <bool>true</bool>
    </property>
   </widget>
   <widget class="QLabel" name="label">
    <property name="geometry">
     <rect>
      <x>130</x>
      <y>80</y>
      <width>281</width>
      <height>61</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>24</pointsize>
     </font>
    </property>
    <property name="text">
     <string>Текст на метке</string>
    </property>
    <property name="textFormat">
     <enum>Qt::AutoText</enum>
    </property>
   </widget>
  </widget>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>554</width>
     <height>28</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
"""

class MyWidget(QMainWindow):
```

```
    def __init__(self):
        super().__init__()
        f = io.StringIO(template)
        uic.loadUi(f, self)  # Загружаем дизайн
        self.pushButton.clicked.connect(self.run)
        # Обратите внимание: имя элемента такое же как в QTDesigner

    def run(self):
        self.label.setText("OK")
        # Имя элемента совпадает с objectName в QTDesigner


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = MyWidget()
    ex.show()
    sys.exit(app.exec_())
```

- Всё содержимое ui-файла помещаем в многострочную строковую константу, в нашем примере `template`.
- Далее при помощи строк

```
f = io.StringIO(template)
uic.loadUi(f, self)
```

подключаем дизайн.

Справка