

Abdelmalek Essaâdi University

National School of Applied Sciences of Tangier

**Master CyberSecurity and cyberCriminality (MCSC)**

**Department of Mathematics and Computer Science**

System programming Project

Creating a GUI using pyqt5 implementing the different labs of System Programming in Python.

**Author:Achraf Chliyah Supervisor: Pr Khalid AMECHNOUE**

19/02/2022

MCSC 2

**Table Of Contents**

[INTRODUCTION](#_mtse5bia7qih)

[Technology used](#_vdsmlamqhejc)

[python](#_xg9tr4q8hs7k)

[PyQt](#_78fsf7lcut46)

[Qt Creator](#_jjnte9ak0ew4)

[Install Qt Creator on Linux](#_z2ymdlpzyz63)

[1) Install prerequisites](#_q0ql7j9b7gnk)

[2) Download the Qt installer](#_hak1r7j74a95)

[3) Run the Qt installer](#_nxqchxl9xr3z)

[Converting your .ui file to Python](#_qfic2pbwfe5y)

[Building Main Windows](#_7qbdn34ehepj)

[Laying out the Main Window](#_93l6nqlhfjgl)

[QPushButton widget](#_wu2631bgnd12)

[QPlainTextEdit widget](#_966xoxrwqha0)

[QTextEdit widget](#_txfe7366igqi)

[QTabWidget widget](#_73miy0hzerih)

[list of system calls implemented](#_1iz5pbeqzw6g)

[fork()](#_4os548im5xzh)

[wait()](#_u81voeao0xie)

[getpid()](#_ux9mxid161q8)

[getppid()](#_h5fdhylwtfpa)

[exec() family](#_ikos69demf8f)

[system(cmd)](#_x6tbmts6n90r)

[dup(fd)](#_s12zwelzg9l9)

[dup2(fd, fd2, inheritable=True)](#_cajmtrp8b6sm)

[pipe()](#_g966ajpyfy73)

[pipe2(flags)](#_hqmn9k32meyz)

[mkfifo(path, mode=438, \*, dir\_fd=None)](#_z29p5cmbyvn1)

[signal.signal(signalnum, handler)](#_3f3bx7gvs4n7)

[lseek(fd, pos, how)](#_8taxs6wnp94z)

[signal.alarm(time)](#_8r0dw2ymtna5)

[kill(pid,sig)](#_czfoanuk7pgg)

[CONCLUSION](#_6abhef3vzg2)

# INTRODUCTION

This project is a realization of a graphical user interface GUI using the pyqt5 library to implement the various system calls relating to the linux operating system. The programming language used is Python. An operating system like Linux has over 380 distinct system calls so we can't cover all the system calls. for more information on all linux system calls visit the link https://man7.org/linux/man-pages/man2/syscalls.2.html

# Technology used

## python

**Python** is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

## PyQt

**PyQt** is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt is free software developed by the British firm Riverbank Computing. It is available under similar terms to Qt versions older than 4.5; this means a variety of licenses including GNU General Public License (GPL) and commercial license, but not the GNU Lesser General Public License (LGPL). PyQt supports Microsoft Windows as well as various flavours of UNIX, including Linux and MacOS (or Darwin).

PyQt implements around 440 classes and over 6,000 functions and methods[including:

* a substantial set of GUI widgets
* classes for accessing SQL databases (ODBC, MySQL, PostgreSQL, Oracle, SQLite)
* QScintilla, Scintilla-based rich text editor widget
* data aware widgets that are automatically populated from a database
* an XML parser
* SVG support
* classes for embedding ActiveX controls on Windows (only in commercial version)

To automatically generate these bindings, Phil Thompson developed the tool SIP, which is also used in other projects.

In August 2009, Nokia, the then owners of the Qt toolkit, released PySide, providing similar functionality, but under the LGPL after failing to reach an agreement with Riverbank Computing to change its licensing terms to include LGPL as an alternative license

## Qt Creator

**Qt Creator** is a cross-platform C++, JavaScript and QML integrated development environment which simplifies GUI application development. It is part of the SDK for the Qt GUI application development framework and uses the Qt API, which encapsulates host OS GUI function calls.[5] It includes a visual debugger and an integrated WYSIWYG GUI layout and forms designer. The editor has features such as syntax highlighting and autocompletion. Qt Creator uses the C++ compiler from the GNU Compiler Collection on Linux. On Windows it can use MinGW or MSVC with the default install and can also use Microsoft Console Debugger when compiled from source code. Clang is also supported.

# Install Qt Creator on Linux

## 1) Install prerequisites

If you're running Ubuntu, Debian, Mint, or any other Debian derivative, you can install the needed prerequisites using the apt package manager. Issue the following commands below in your shell.

This command ensures your package manager is up-to-date:

**sudo apt-get update && sudo apt-get upgrade**

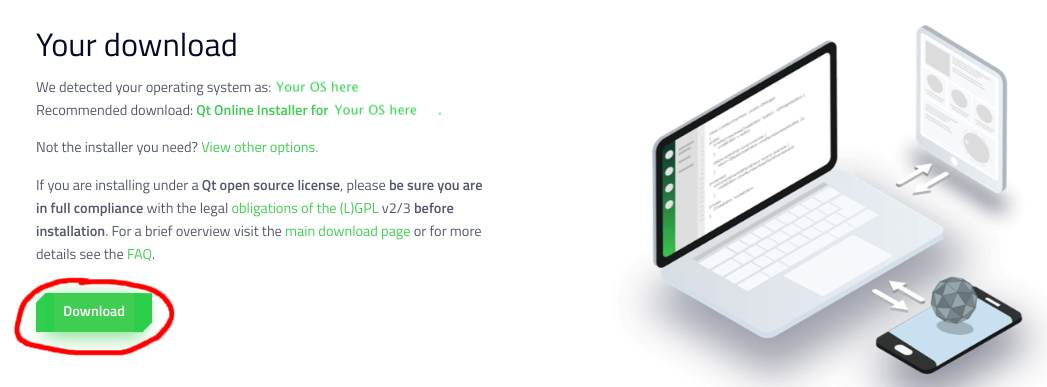
This command installs the tools and libraries needed for Qt

**sudo apt-get -y install build-essential openssl libssl-dev libssl1.0 libgl1-mesa-dev libqt5x11extras5**

If you're running some other variant of Linux, figure out a way to install the tools make, g++, and gdb. (For example, on Fedora / Red Hat systems, you may be able to use the yum package manager.)

## 2) Download the Qt installer

Download the Qt installer from its official download site at **https://www.qt.io/download-qt-installer**. The site should detect that your computer is running linux and recommend "Qt Online Installer for linux". Click the green **"Download"** button to download the installer.

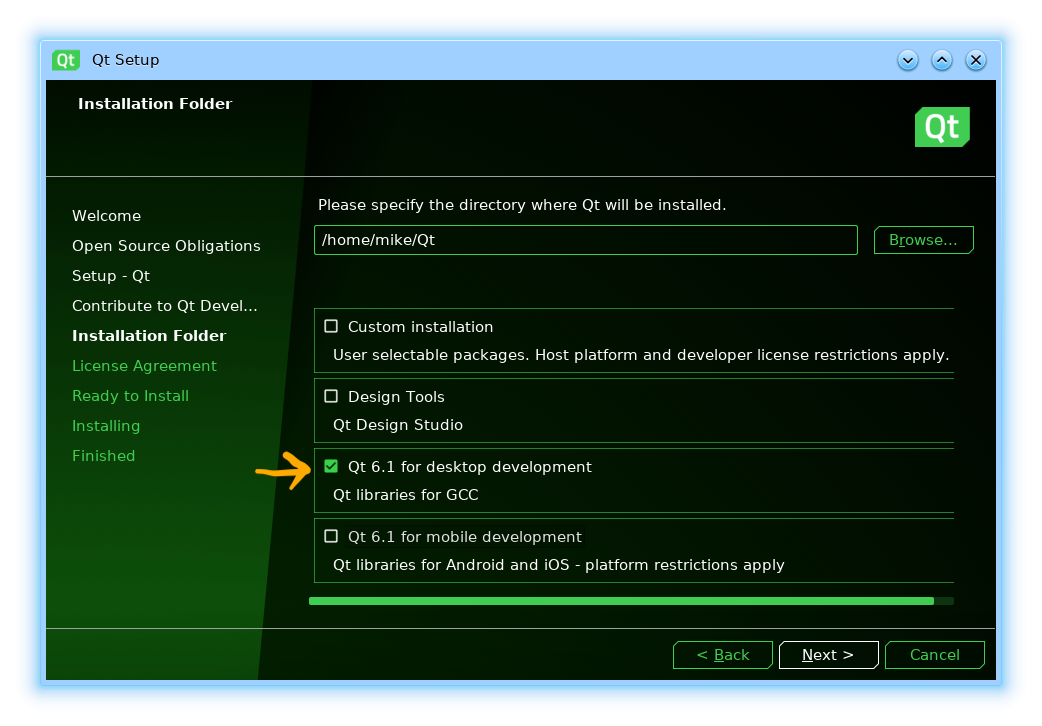


## 3) Run the Qt installer

The dowloaded installer is named something like qt-unified-linux-x64-version-online.run and and is likely located in your ~/Downloads directory. Use chmod +x qt\*.run to make the file executable, then run it by typing ./qt\*.run

The Qt installer will walk you through a set of steps. For most steps, you can use the default settings and simply click **"Next"** or **"Agree"** to move on, with the following exceptions:

* At the Welcome step, sign up for your own Qt Account. Go ahead and put in your email and verify your account via email. When setting up your account, you do not have to put in your phone number or city.
* At the Installation Folder step (see screenshot below),select the option **Qt 6.x for desktop development Qt libraries for GCC**. Do not change the name or location of the directory where Qt will be installed.



# Converting your .ui file to Python

To generate a Python output file run pyuic5 from the command line, passing the .ui file and the target file for output, with a -o parameter. The following will generate a Python file named MainWindow.py which contains our created UI.

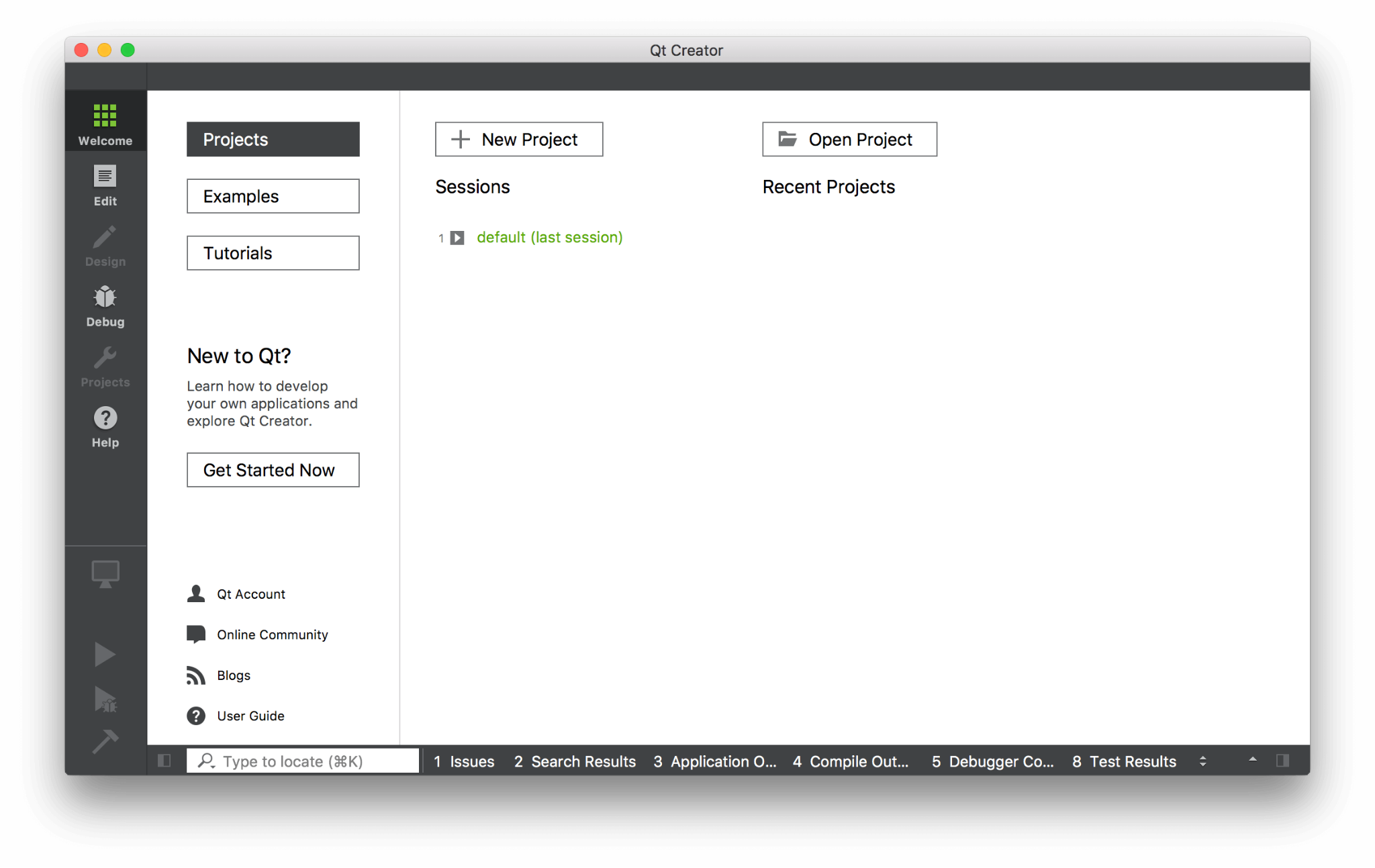
**pyuic5 mainwindow.ui -o MainWindow.py**

You can open the resulting MainWindow.py file in an editor to take a look, although you should *not* edit this file. The power of using Qt Creator is being able to edit, tweak and update your application while you develop. Any changes made to this file will be lost when you update it. However, you *can* override and tweak anything you like when you import and use the file in your applications.

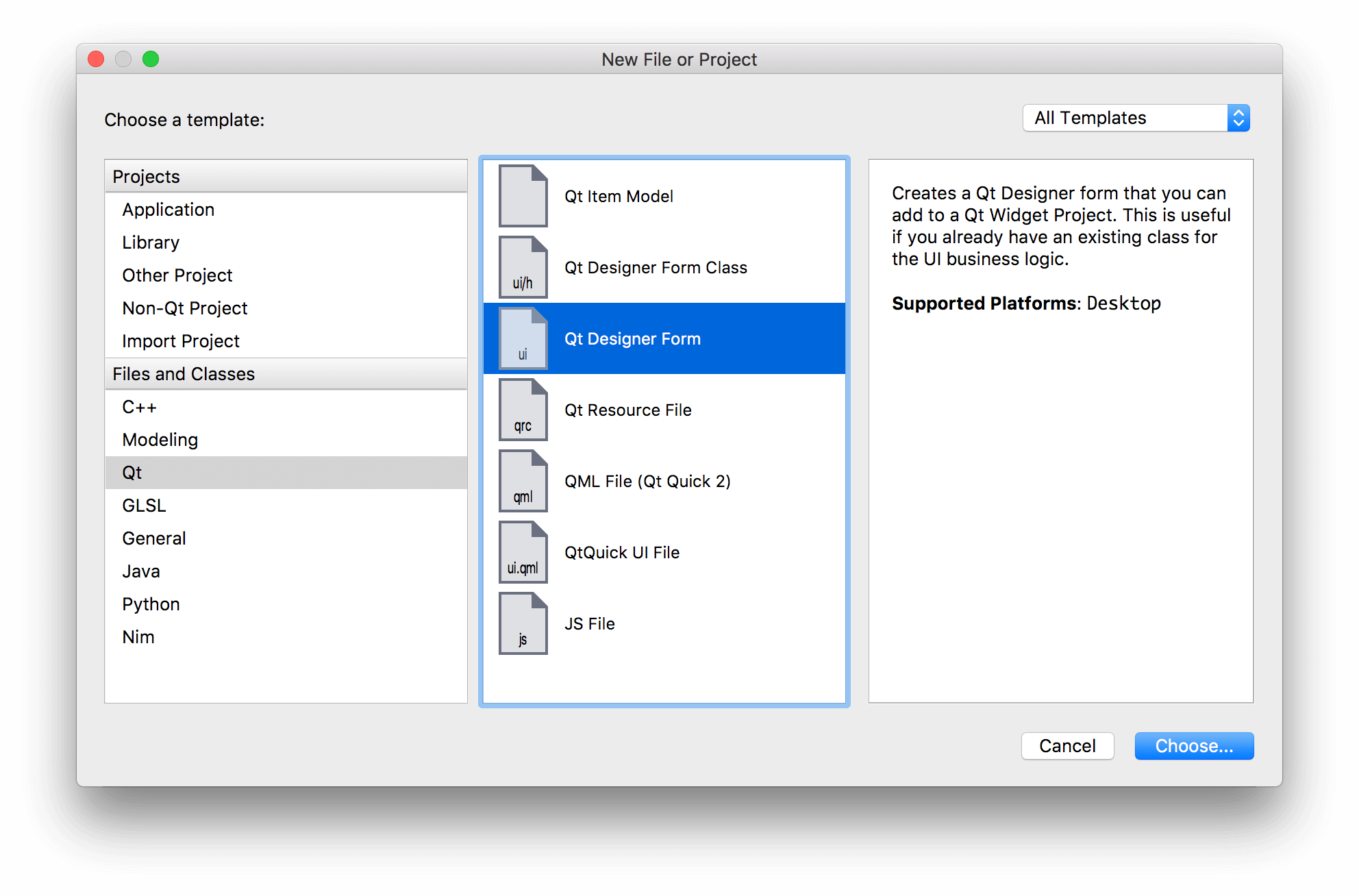
# 

# Building Main Windows

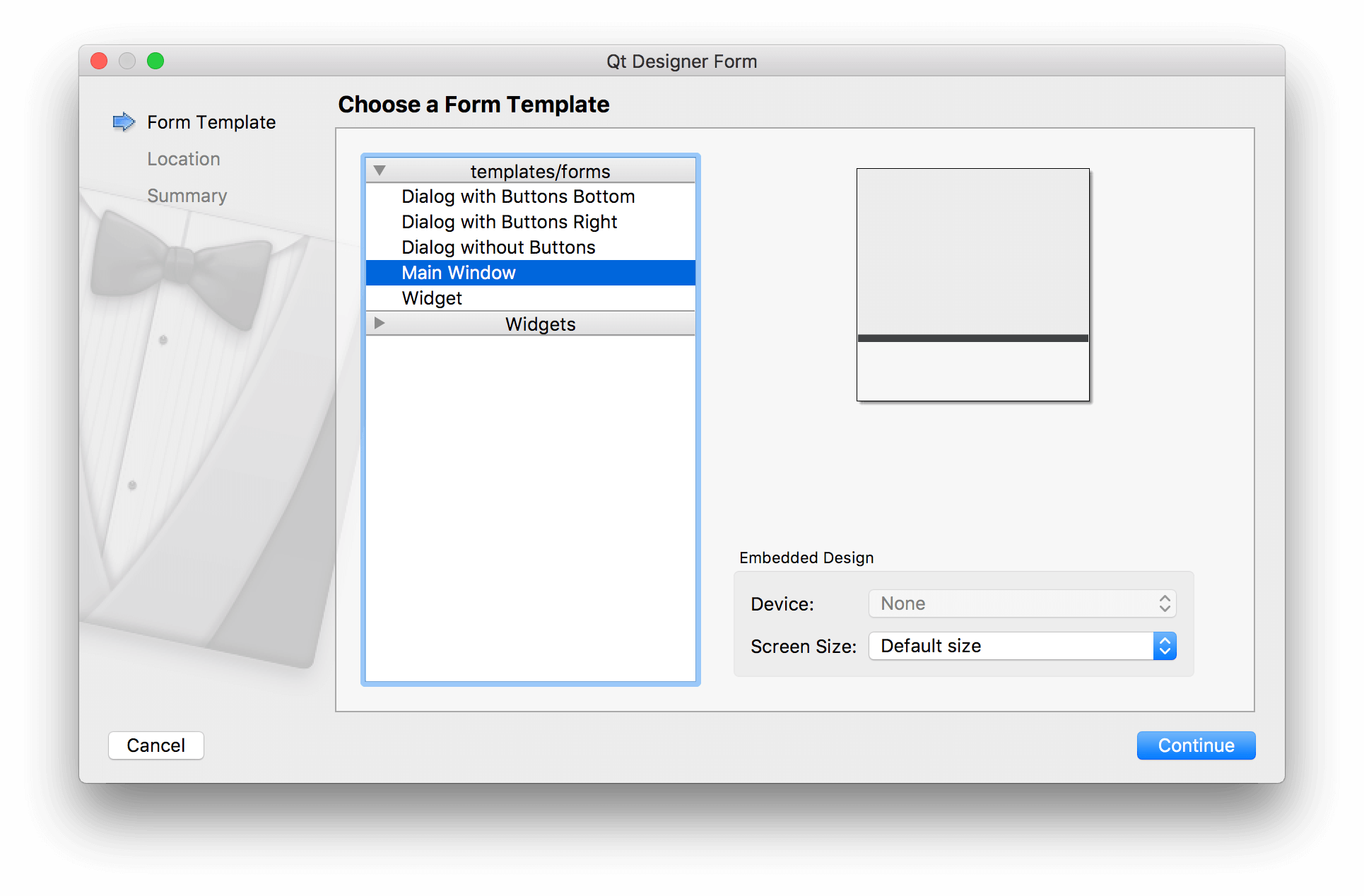
Open up Qt Creator and you will be presented with the main window. The designer is available via the tab on the left hand side. However, to activate this you first need to start creating a .ui file.



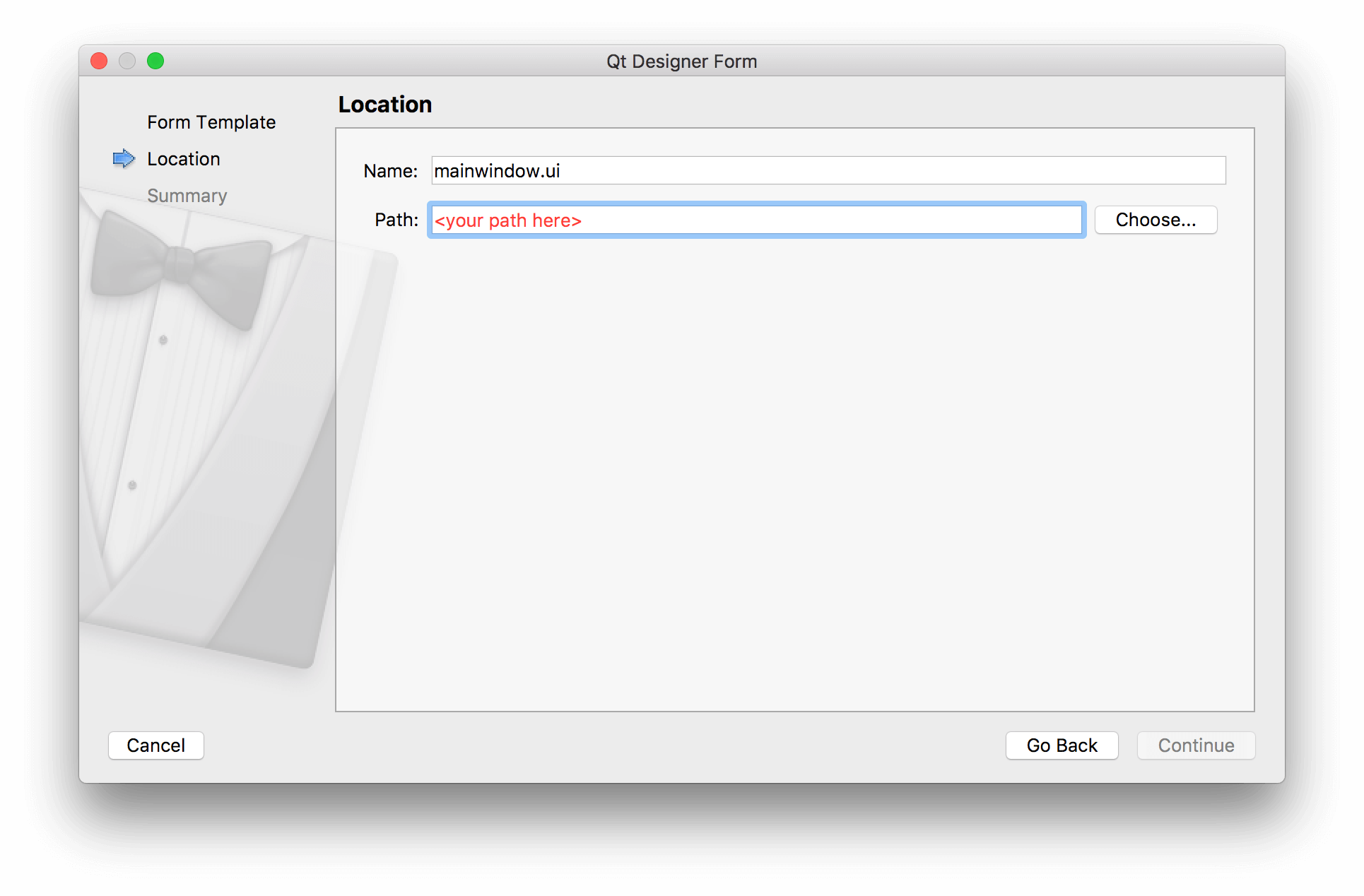
To create a .ui file go to File -> New File or Project... In the window that appears select *Qt* under *Files and Classes* on the left, then select *Qt Designer Form* on the right. You'll notice the icon has "ui" on it, showing the type of file you're creating.



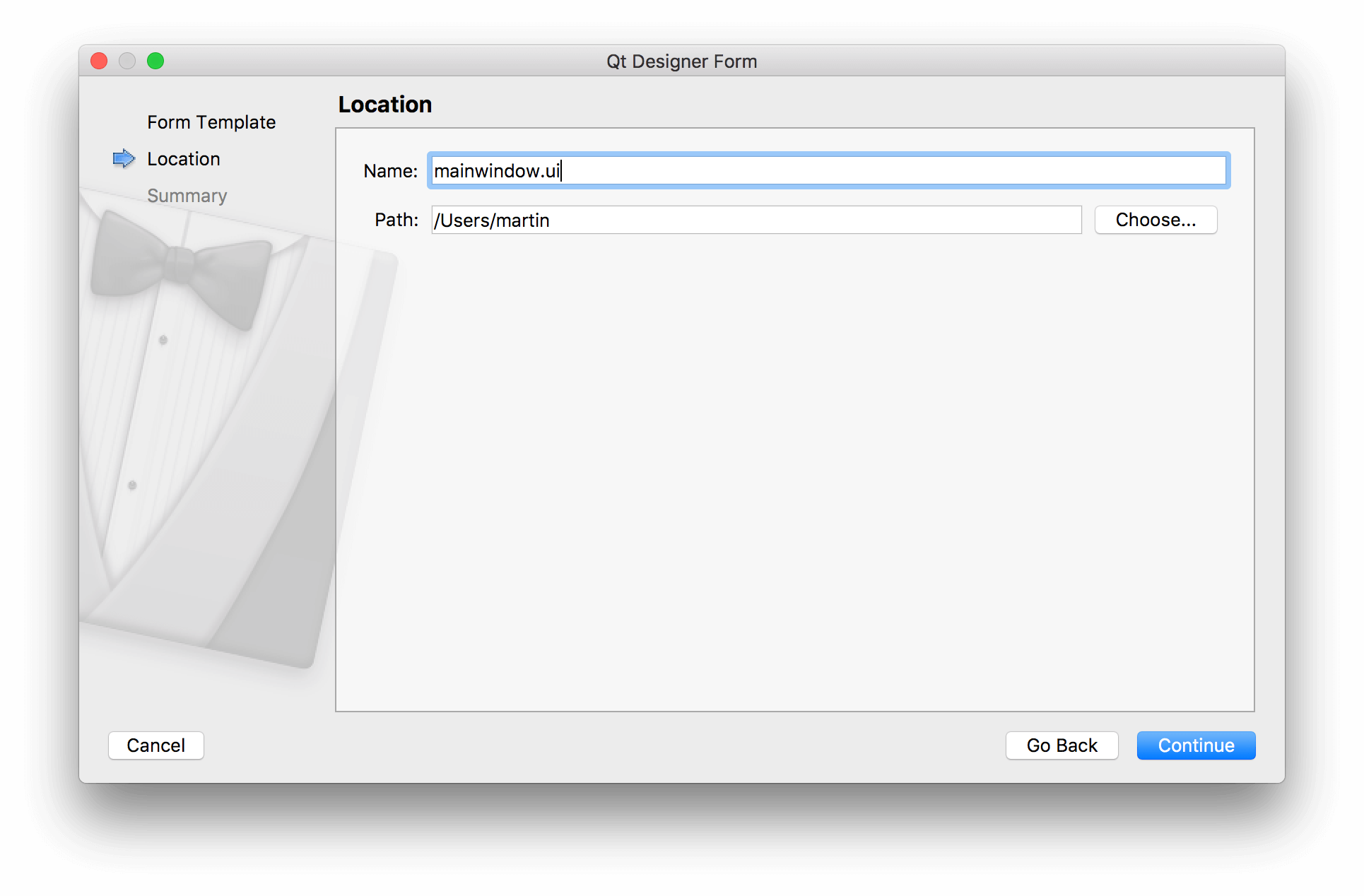
In the next step you'll be asked what type of widget you want to create. If you are starting an application then *Main Window* is the right choice. However, you can also create .ui files for dialog boxes, forms and custom compound widgets.



Next choose a filename and save the folder for your file. Save your .ui file with the same name as the class you'll be creating, just to make subsequent commands simpler.



Finally, you can choose to add the file to your version control system if you're using one. Feel free to skip this step — it doesn't affect your UI.



# Laying out the Main Window

in my application i used four widgets which is **QPushButtonwidget** to launch the program that use the system call and tab widget that contain **QTextEdit** widget to display the program and it output and a **QPlainTextEdit** widget to display a definition of the system call in case.

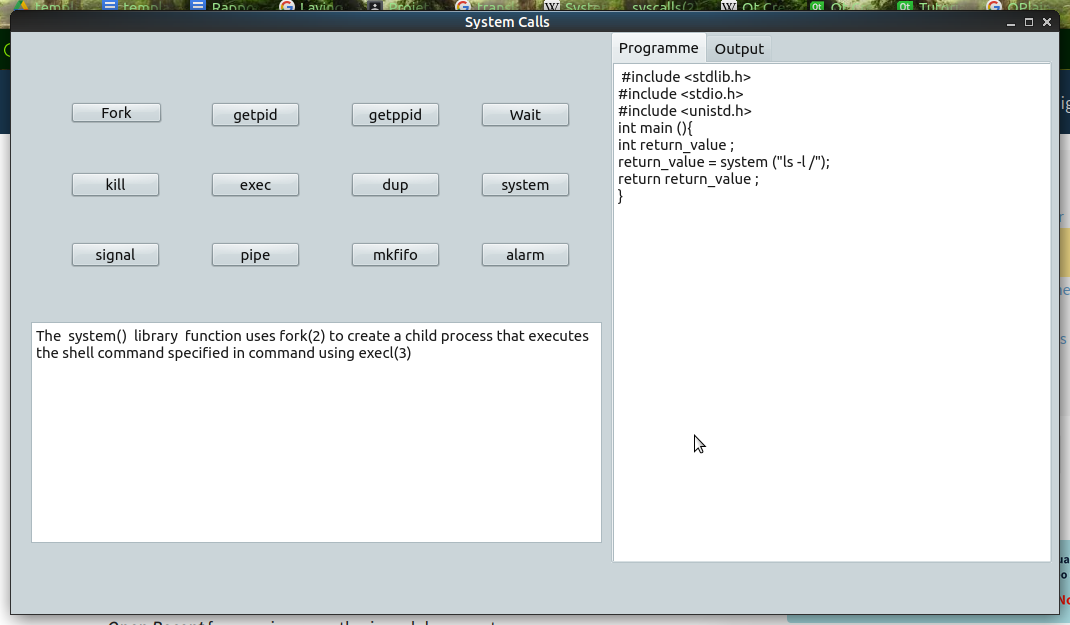
I used the **subprocess** module to execute the C code of different system calls like:

for displaying the program code:

**prog = subprocess.run(["cat", "system.c"], capture\_output=True)**

for displaying the output of the program:

**output = subprocess.run("./system", capture\_output=True)**



## 

## QPushButton widget

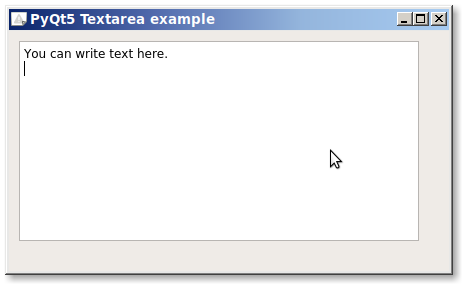


The push button, or command button, is perhaps the most commonly used widget in any graphical user interface. Push (click) a button to command the computer to perform some action, or to answer a question. Typical buttons are OK, Apply, Cancel, Close, Yes, No and Help.

A command button is rectangular and typically displays a text label describing its action. A shortcut key can be specified by preceding the preferred character with an ampersand in the text.

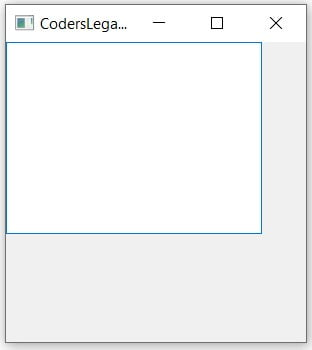
## QPlainTextEdit widget

QPlainTextEdit is an advanced viewer/editor supporting plain text. It is optimized to handle large documents and to respond quickly to user input.

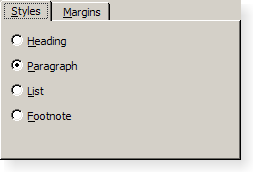


## QTextEdit widget

**QTextEdit** is an advanced WYSIWYG viewer/editor supporting rich text formatting using HTML-style tags, or Markdown format. It is optimized to handle large documents and to respond quickly to user input.



## QTabWidget widget

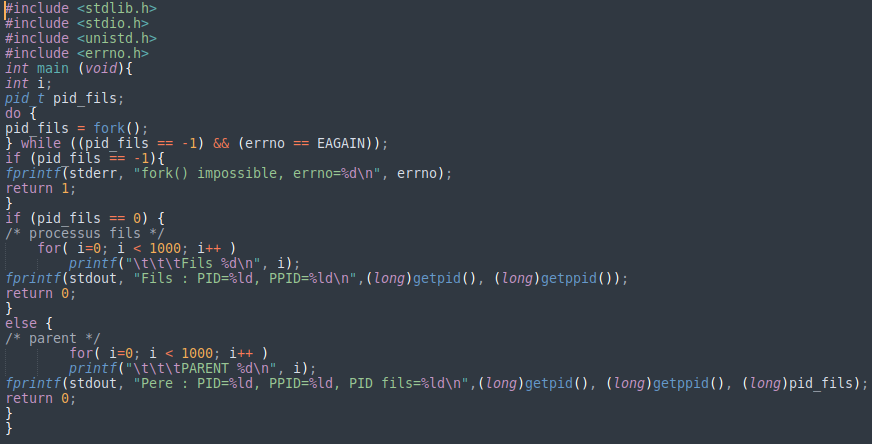


A tab widget provides a tab bar and a "page area" that is used to display pages related to each tab. By default, the tab bar is shown above the page area, but different configurations are available. Each tab is associated with a different widget (called a page). Only the current page is shown in the page area; all the other pages are hidden. The user can show a different page by clicking on its tab or by pressing its Alt+*letter* shortcut if it has one.

# list of system calls implemented

## fork()

Fork a child process. Return 0 in the child and the child’s process id in the parent. If an error occurs **OSError** is raised.



## wait()

Wait for completion of a child process, and return a tuple containing its pid and exit status indication: a 16-bit number, whose low byte is the signal number that killed the process, and whose high byte is the exit status (if the signal number is zero); the high bit of the low byte is set if a core file was produced.

## getpid()

Return the current process id.

## getppid()

Return the parent’s process id. When the parent process has exited, on Unix the id returned is the one of the init process (1), on Windows it is still the same id, which may be already reused by another process.

## 

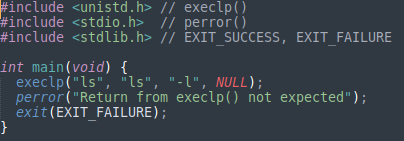
## 

## 

## exec() family

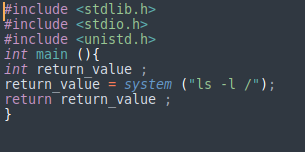
* **execl**(path, arg0, arg1, ...)
* **execle(**path, arg0, arg1, ..., env)
* **execlp**(file, arg0, arg1, ...)
* **execlpe**(file, arg0, arg1, ..., env)
* **execv**(path, args)
* **execve(**path, args, env)
* **execvp(**file, args)
* **execvpe**(file, args, env)

These functions all execute a new program, replacing the current process; they do not return. On Unix, the new executable is loaded into the current process, and will have the same process id as the caller. Errors will be reported as **OSError** exceptions.



## system(cmd)

Execute the command (a string) in a subshell. This is implemented by calling the Standard C function system(), and has the same limitations.



## 

## 

## 

## 

## 

## 

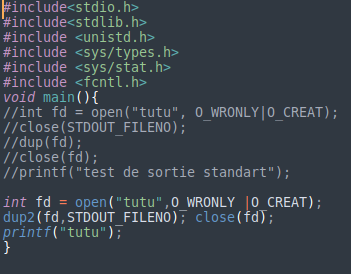
## 

## 

## 

## dup(fd)

Return a duplicate of file descriptor fd. The new file descriptor is non-inheritable

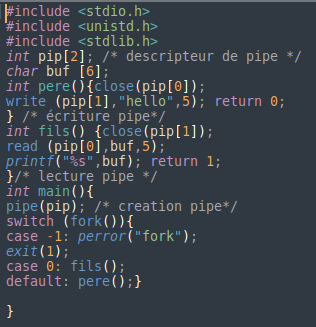


## dup2(fd, fd2, inheritable=True)

duplicate file descriptor fd to fd2, closing the latter first if necessary. Return fd2. The new file descriptor is inheritable by default or non-inheritable if inheritable is False.

## pipe()

Create a pipe. Return a pair of file descriptors (r, w) usable for reading and writing, respectively. The new file descriptor is non-inheritable.

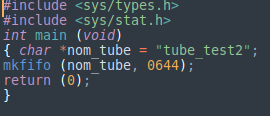


## pipe2(flags)

create a pipe with flags set atomically. flags can be constructed by ORing together one or more of these values: **O\_NONBLOCK**, **O\_CLOEXEC**. Return a pair of file descriptors (r, w) usable for reading and writing, respectively.

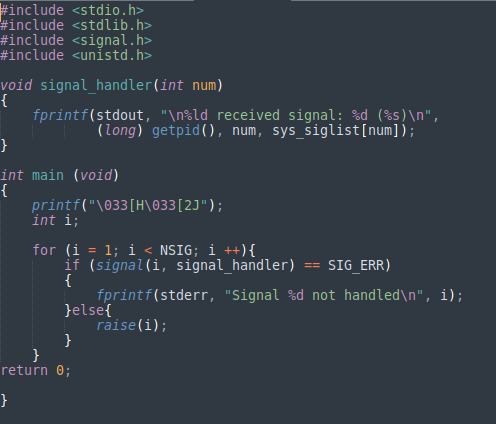
## mkfifo(path, mode=438, \*, dir\_fd=None)

Create a FIFO (a named pipe) named path with numeric mode mode. The current umask value is first masked out from the mode.



## signal.signal(signalnum, handler)

Set the handler for signal **signalnum** to the function handler. handler can be a callable Python object taking two arguments ,or one of the special values **signal.SIG\_IGN** or **signal.SIG\_DFL**

****

## 

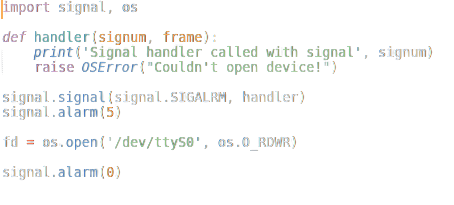
## 

## lseek(fd, pos, how)

Set the current position of file descriptor fd to position pos, modified by how: **SEEK\_SET** or 0 to set the position relative to the beginning of the file; **SEEK\_CUR** or 1 to set it relative to the current position; S**EEK\_END** or 2 to set it relative to the end of the file. Return the new cursor position in bytes, starting from the beginning.

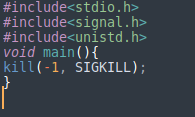
## signal.alarm(time)

if time is non-zero, this function requests that a **SIGALRM** signal be sent to the process in seconds. Any previously scheduled alarm is canceled (only one alarm can be scheduled at any time). The returned value is then the number of seconds before any previously set alarm was to have been delivered. If time is zero, no alarm is scheduled, and any scheduled alarm is canceled. If the return value is zero, no alarm is currently scheduled.



## kill(pid,sig)

Send signal **sig** to the process **pid**. Constants for the specific signals available on the host platform are defined in the signal module.

****

# 

# 

# CONCLUSION

When you create applications in PyQt, you commonly build a main window and several dialogs. Building the GUI of those windows and dialogs can take a lot of time if you hand code them. Luckily, Qt provides Qt Designer, which is a powerful tool intended to create GUIs fast and productively using a user-friendly graphical interface.

With Qt Designer, you can drag and drop all the required widgets onto an empty form, lay them out, and create your GUIs in almost no time. Those GUIs are saved in .ui files that you can translate into Python code and use in your applications.