Interactive Programming with PyQt5

Canberra Python User Group
February 2019

Outline

- Qt Background
- Installing Qt and PyQt5
- API Break PyQt4 vs PyQt5
- GUI Design
- Interacting with Widgets
- Signals and Slots
- Demonstration

Qt Background

Qt is a set of C++ libraries and development tools that includes platform independent abstractions for graphical user interfaces, networking, threads, regular expressions, SQL databases, SVG, OpenGL, XML, user and application settings, positioning and location services, short range communications (NFC and Bluetooth), web browsing, 3D animation, charts, 3D data visualisation and interfacing with app stores. PyQt5 implements over 1000 of these classes as a set of Python modules.

Called Qt because the letter Q looked appealing in Emacs typeface, and "t" was inspired by Xt, the X toolkit.

Qt Background

Qt was originally developed by Haavard Nord and Eirik Chambe-Eng at TrollTech in Finland in 1990, with an open source version for X11/Unix and a proprietary version for Windows.

In 2000, Qt/X11 2.2 was released under the GPL v2 and for Windows under the GPL in June 2005

Qt Background

Nokia acquired TrollTech in 2008 and focused resources into their Symbian mobile OS with Qt as the main development platform.

In 2011 Nokia dropped Qt in favour of Windows Phone. This had made many people very angry and has been widely regarded as a bad move. They sold Qt to Digia, who carved it off into the Qt Company in 2014.

Qt Background

Qt is currently being developed by The Qt Company, a publicly listed company, and the Qt Project under open-source governance, involving individual developers and organizations working to advance Qt. Qt is available under both commercial licenses and open source GPL 2.0, GPL 3.0, and LGPL 3.0 licenses.

Qt Background: Examples

Qt in the wild:

- Google Earth
- KDE Plasma
- Tesla Model S in-car UI
- Mathematica
- TeamViewer
- VLC Media Player
- Blizzard Entertainment.
- Dreamworks
- Lucasfilm
- Valve Corporation

Qt Background: Platforms

Qt Platforms:

Linux

- X11, Android, Wayland

Microsoft

- Windows (XP, Vista, 7, 8, 10)

Apple

- iOS, MacOS (via Cocoa API)

Qt Background: Languages

Language bindings

- C++
- C#
- Go
- Haskell
- JavaScript
- Java
- Pascal
- **PYTHON**
- Ruby
- Rust

Qt Background: Python Bindings

PyQt is python bindings developed by the British firm Riverbank Computing using the SIP binding generator

PySide is python bindings developed by Finnish firm The Qt Company using the shiboken binding generator

Similar functionality

Different developers, licensing terms, documentation, platforms, binding generators

Installing Qt (14GB installed)

Windows:

online or offline installers: https://www.qt.io/download-qt-installer

Ubuntu:

sudo apt install build-essential, qtcreator, qt5-default, pyqt5-dev-tools, qttools5-dev-tools

Installing PyQt5

pip3 install pyqt5

Only Python v3.5 and later are supported.

Qt uses SIP to generate python bindings (originally developed in 1998 for PyQt), not BOOST, not SWIG

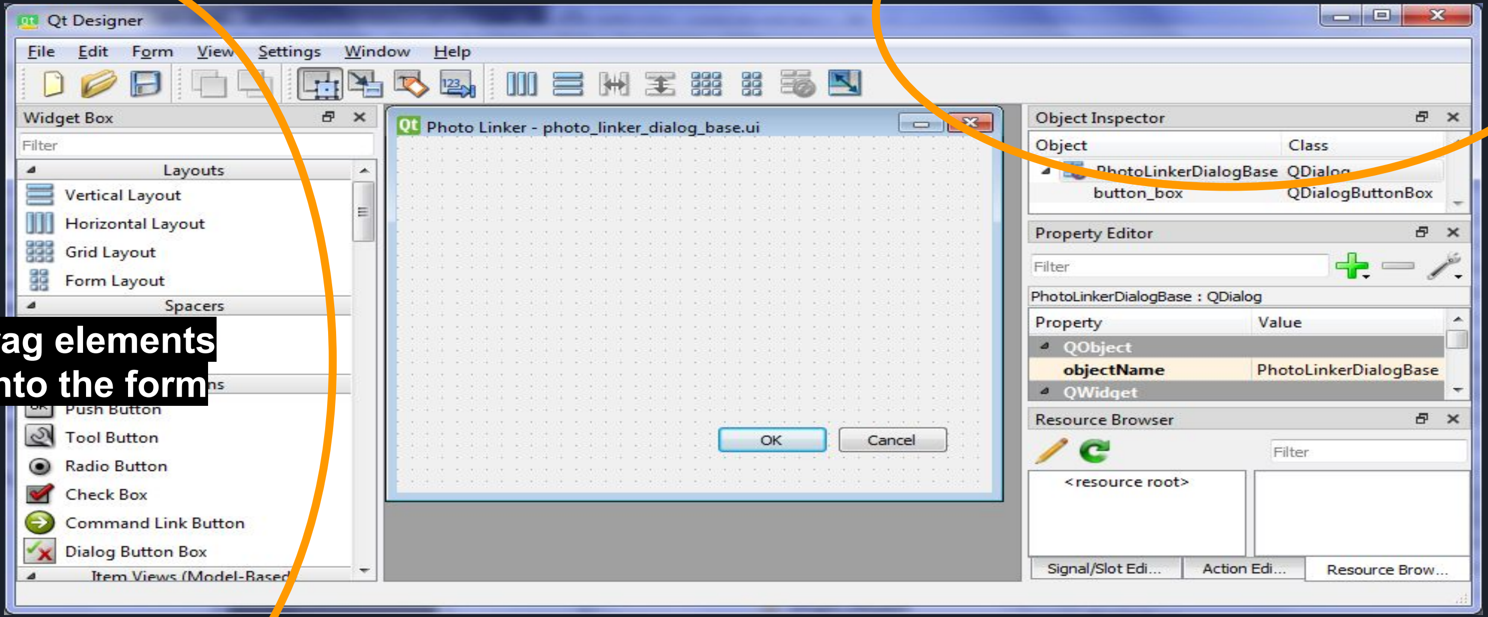
PyQt4 and PyQt5

The PyQt5 is not backward compatible with PyQt4; there are several significant changes in PyQt5.

- Python modules have been reorganized. Some modules have been dropped (QtScript), others have been split into submodules (QtGui, QtWebKit).
- New modules have been introduced, including QtBluetooth, QtPositioning, or Enginio.

- PyQt5 supports only the new-style signal and slots handling. The calls to SIGNAL() or SLOT() are no longer supported.

GUI Design: Qt Designer



Object	Class
PhotoLinkerDialogBase	QDialog
button_box	QDialogButtonBox

Property Editor

PhotoLinkerDialogBase : QDialog

Property	Value
QObject	
objectName	PhotoLinkerDialogBase
QWidget	

Element Name

Drag elements onto the form

GUI Design: Qt Resources

http://pyqt.sourceforge.net/Docs/PyQt5/resources.html

PyQt5 supports Qt's resource system. This is a facility for embedding resources such as icons and translation files in an application. This makes the packaging and distribution of those resources much easier.

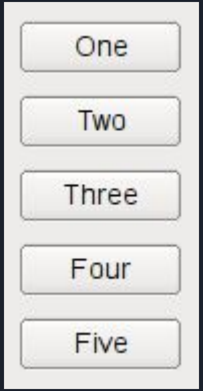
A .qrc resource collection file is an XML file used to specify which resource files are to be embedded.

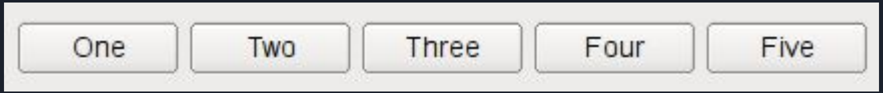
pyrcc5 reads the .qrc file, and the resource files, and generates a Python module that only needs to be imported by the application in order for those resources to be made available just as if they were the original files.

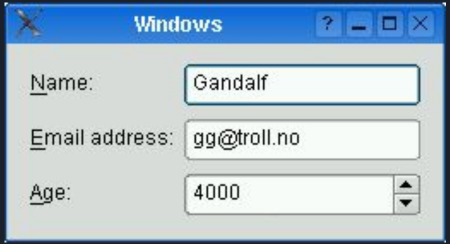
GUI Design: Layouts

Automatically position elements on your form

- QVBoxLayout - 1 Vertical column
- QHBoxLayout - 1 Horizontal row
- QGridLayout - rows and columns with grid reference system
- QFormLayout - 2 columns (label, then input widget)

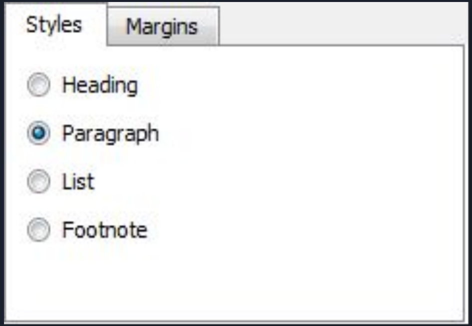




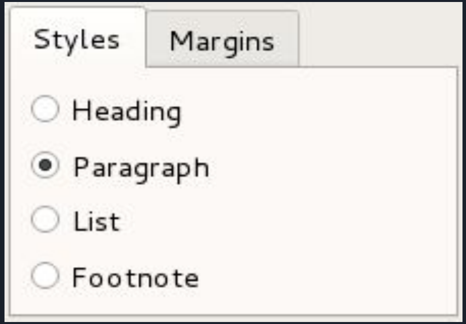


Name: Gandalf
Email address: gg@troll.no
Age: 4000

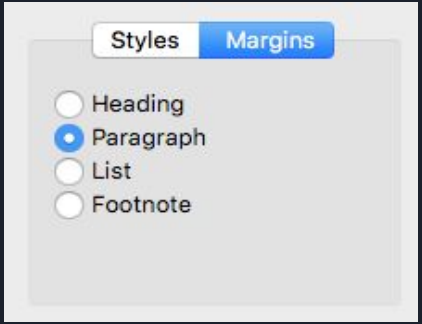
GUI Design: Styles



QWindowsVistaStyle



QFusionStyle



QMacStyle

The QApplication

There can be only one QApplication, it manages:

- user desktop settings (fonts, doubleclick interval)
- event handling from underlying window system
- parse command line arguments (style, debugging)
- defines look and feel (style)
- localisation (translation)
- clipboard
- mouse cursor

Querying Widgets: getting information

Importing the QT Designer ui file as a class.

```
# import the dialog form as a class

from dialogScript import DialogClass

# instantiate the dialog form class

def __init__(self):

   self.dialog = DialogClass()

self.dialog.show()

# query the dialog class radioButton element’s status

self.dialog.radioButton.isChecked()
```

Querying Widgets: setting information

```
# populate a comboBox element with sequential values
for i in range(10):
    self.dialog.comboBox.addItem(str(i))
# set a checkBox to checked
self.dialog.checkBox.setChecked(True)
# set the text for a textLabel
self.dialog.textLabel.setText(“sample text”)
```

Event Loops are Dead

Game event loop pseudo-code:

```
while not ExitCondition:
    check for user input
    run AI
    move enemies
    resolve collisions
    draw graphics
    play sounds
```

Event Loops are Dead

Event Loops block until an event has arrived

This is bad.

Asynchronous signal handlers are much better

Event Loops are (not) DEAD

QApplication.exec() ???

This command enters the main event loop to start event handling (user interaction)

When a signal calls a slot it is executed immediately, independent of the GUI event loop.

Signals and Slots (user interaction)

Interactive Programming relies on signals:

A signal is emitted when something of potential interest happens. A slot is a Python callable. If a signal is connected to a slot then the slot is called when the signal is emitted. If a signal isn’t connected then nothing happens. The code (or component) that emits the signal does not know or care if the signal is being used.

PyQt5 Signals and Slots

```
# slot - executes an action

def printValue():

   print(self.textToPrint)

# signal - initiates an action

self.dialog.spinBox.valueChanged.connect(printValue)
```

Notice there are no arguments passed?

Demonstration: Hello, World!

- **Hello, World!**, first step in interactive python

A GUI with a button that emits a signal to change a label's text to "Hello, World!"

Topics not covered in this presentation

- Internationalisation (translations)
- Documentation (sphinx: HTML, LaTeX, epub, man, QtHelp)
- Testing
  - unit testing
  - assertions
  - doctest
  - property-based testing
  - code profiling
- Remote debugging
- Security (sql injection, user input sanitation, web security)
- Compiling an executable or installer
- PySide

Topics not covered in this presentation

- UI Design

Jonah Sullivan,

e jonah.sullivan@ga.gov.au

t +61 (0) 2 6249 9516