

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa

Melksham

Wiltshire

UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

Python and the Qt GUI

The Qt GUI (Graphic User Interface) is supported in Python via the PyQt module. With PyQt and Qt, you can develop applications with a graphic look and feel under Python. This module introduces the concepts involved through some programs and demonstration.

<i>Introduction to Qt and other Python GUIs</i>	<i>238</i>
<i>The Component parts of using Python with Qt</i>	<i>238</i>
<i>Hello Python GUI world.</i>	<i>240</i>
<i>Widgets, geometry, events, signals, slots...</i>	<i>240</i>
<i>A simple but practical control</i>	<i>244</i>
<i>Where to learn the use of Python/Qt applications</i>	<i>244</i>
<i>BlackAdder.</i>	<i>245</i>

20.1 Introduction to Qt and other Python GUIs

Qt is a Graphic User Interface library that provides C++ programmers with the ability to write portable GUIs. PyQt lets you glue the Qt library to Python so that you can similarly write portable GUI applications in Python.

Once you have a platform-independent GUI all sorts of exciting possibilities open up. One is the Blackadder environment¹ where you use code based on Python with Qt to develop further Python applications.

There are a number of other GUIs available for Python.

- **Tkinter** is the oldest Python GUI toolkit. It is based on tcl/tk, and has neither the real platform GUI look and feel, nor a real Python programming style. A good resource is John Grayson's book, Python and Tkinter programming.
- **wxPython** is based on the wxWindows toolkit. wxWindows is a crossplatform wrapper around a native toolkit of each platform: the standard Win32 controls on Windows; and GTK on Unix/X11.
- **FxPy** is one of the smaller toolkits² and is based on the FOX toolkit. FxPy's main feature is execution speed.
- **PyGTK** is based on GTK (formerly known as the Gimp Toolkit). Not really intended for cross-platform work, it has recently been ported (more or less) to Windows.
- **Pythonwin** is the (rather underdocumented) binding to Microsoft's MFC library. It's not portable, of course.

So what of PyQt? It's documentation says:

"PyQt is a set of Python bindings for the Qt toolkit. The bindings are implemented as a set of Python modules: qt, qtcanvas, qtgl, qtnetwork, qtsql, qtable, qtui and qtxml, and contains 300 classes and over 5,750 functions and methods.

"PyQt also implements the qtext Python module. This contains bindings for QScintilla, the Qt port of the Scintilla programmer's editor class.

"PyQt is licensed under the GNU GPL (for UNIX/Linux and MacOS/X), under the Qt Educational License (for use with the educational edition of Qt for Windows), and under a commercial license (for Windows, UNIX/Linux and MacOS/X). You can purchase the commercial version of PyQt from <http://www.riverbankcomputing.co.uk/pyqt/index.php>.

"PyQt supports Qt versions 1.43 to 3.2.3 and Python versions 1.5 to 2.3.2. PyQt has been ported to Windows, MacOS/X and UNIX/Linux."

This module is intended to get you started using Qt from Python, and assumes prior knowledge of Python. If you'll be making use of Qt, you'll need to make a separate study of it and its facilities. Refer to our "Graphics" library listing for books that cover Qt.

20.2 The Component parts of using Python with Qt

Here are the component parts that you'll need to install and run the Qt GUI in Python. We've listed them in the order you'll need to source and install them if you're starting from scratch, but you'll often find that many (or most, or all) of the components are a standard part of your distribution, especially on Linux.

On *nix systems, you'll be making use of the (gcc compiler and on Windows systems you'll download appropriate binaries.

1. Python

The Python scripting language – an excellent and flexible language that provides one of the best object oriented models around. We're fans! Plenty of standard classes ("modules") available, open source, active community, well liked and used.

2. Qt

¹ a standard application if you wish to use it

² in terms of user base

The Qt GUI toolkit, written in C++. Provides support for all the main elements that you'll want in a GUI – widgets, geometry managers and event handlers, but with its own extra wrappers and technologies on top to make the whole thing even easier. Quite a large library, runs on Windows, Mac OS X, Unix, Linux.

The Qt toolkit is written and maintained by Trolltech.¹ They say "Qt is a multiplatform C++ application framework developers can use to write single-source applications that run - natively - on Windows, Linux, Unix, Mac OS X and embedded Linux. Qt has been used to build thousands of successful commercial applications worldwide, and is the basis of the open source KDE desktop environment..." Trolltech employs a dual licensing strategy, offering both commercial and free software licensing options to developers.

3. SIP

SIP is a tool for generating Python bindings for C and C++ libraries, and it's used by the PyQt installation process.

4. PyQt

PyQt is a set of Python bindings for the Qt toolkit.

PyQt and SIP are written and supplied by Riverbank Computing Ltd.² "As well as developing closed source software for its clients, Riverbank also supports a number of open source software projects and makes them available to the open source community under the terms of the GNU GPL".

Installation summary

Here's an installation summary for a complete Python/QT system on a *nix system on which everything needs to be installed. Please treat it as a guide and read the documentation for each package that you install ;-)

```

http://www.python.org/download/
tar xzf ...
cd resulting directory
./configure
make
make test
make install (as root)
http://www.trolltech.com/download/qt/x11.html
tar xzf ...
mv resulting directory /usr/local/qt
cd /usr/local/qt
QTDIR=/usr/local/qt
PATH=$QTDIR/bin:$PATH
MANPATH=$QTDIR/doc/man:$MANPATH
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export QTDIR PATH MANPATH LD_LIBRARY_PATH
./configure
make
make install (as root)
http://www.riverbankcomputing.co.uk/sip/download.php
tar xzf ....
cd resulting directory
python configure.py
make
make install (as root)
http://www.riverbankcomputing.co.uk/pyqt/index.php
tar xzf ....
cd resulting directory

```

¹ founded 1994, based in Oslo, with offices in the USA and Australia

² a British company registered in Wimborne, Dorset

```
export PATH=/usr/local/bin:$PATH
python configure.py
make
make install (as root)
```

20.3 Hello Python GUI world

Here's a first sample program – it's called *pyqthello.py*

```
#!/usr/local/bin/python

import qt

app=qt.QApplication(["WHC Demo 1"])
button=qt.QPushButton("This is a QPushButton!", None)

app.setMainWidget(button)
button.show()
app.exec_loop()
```

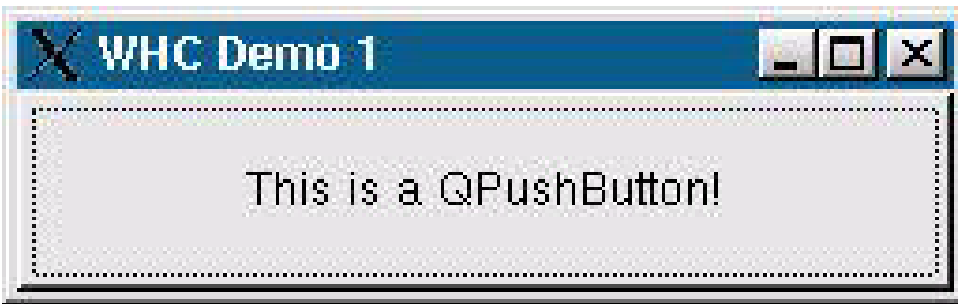


Figure 42 A first program, creating a push button

20.4 Widgets, geometry, events, signals, slots...

Our "hello world" program taught us a lot, yet really wasn't big enough to be useful. Let's take that one step further and add an action to our button and support for multiple windows – program *hq2.py*

```
#!/usr/local/bin/python

import sys
from qt import *

class HelloButton(QPushButton):

    def __init__(self, *args):
        apply(QPushButton.__init__, (self,) + args)
        self.setText("Hello World")

class HelloWindow(QMainWindow):

    def __init__(self, *args):
        apply(QMainWindow.__init__, (self,) + args)
        self.button=HelloButton(self)
        self.setCentralWidget(self.button)
        self.connect(self.button, SIGNAL("clicked()"), self,
SLOT("close()"))

def main(args):
    app=QApplication(args)
```

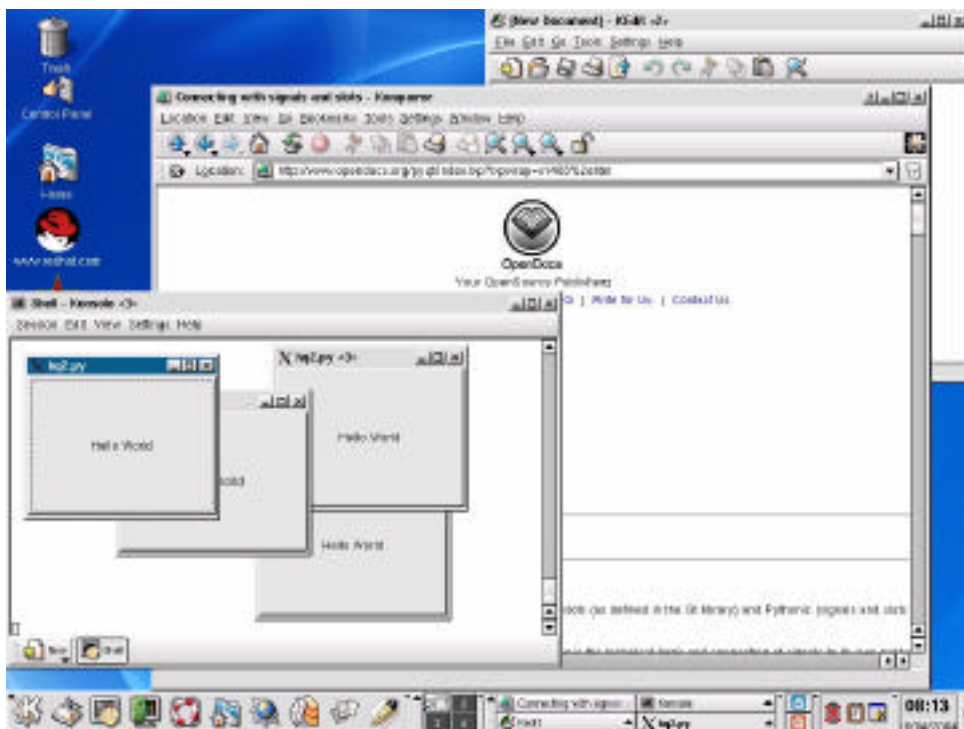
```

winlist = []
for k in range(0,4):
    win=HelloWindow()
    win.show()
    winlist.append(win)
app.connect(app, SIGNAL("lastWindowClosed()"), app, SLOT("quit()"))
app.exec_loop()

if __name__=="__main__":
    main(sys.argv)

```

Figure 43 A PyQt application with multiple windows



Program `hq3.py` introduces user-controlled feedback within a GUI and layout management. We're starting to get to something like a real application now ;-)

```

#!/usr/local/bin/python

import sys
from qt import *

class MainWindow(QMainWindow):

    def __init__(self, *args):
        apply(QMainWindow.__init__, (self, ) + args)

        self.vlayout = QVBoxLayout(self, 10, 5)
        self.hlayout = QHBoxLayout(None, 10, 5)
        self.labelLayout=QHBoxLayout(None, 10, 5)

        self.red = 0
        self.green = 0
        self.blue = 0

        self.dialRed = QDial(0, 255, 1, 0, self)

```

```

# self.dialRed.setBackgroundColor(QColor("red"))
self.dialRed.setNotchesVisible(1)
self.dialGreen = QDial(0, 255, 1, 0, self)
# self.dialGreen.setBackgroundColor(QColor("green"))
self.dialGreen.setNotchesVisible(1)
self.dialBlue = QDial(0, 255, 1, 0, self)
# self.dialBlue.setBackgroundColor(QColor("blue"))
self.dialBlue.setNotchesVisible(1)

self.hlayout.addWidget(self.dialRed)
self.hlayout.addWidget(self.dialGreen)
self.hlayout.addWidget(self.dialBlue)

self.vlayout.addLayout(self.hlayout)

self.labelRed = QLabel("Red: 0", self)
self.labelGreen = QLabel("Green: 0", self)
self.labelBlue = QLabel("Blue: 0", self)

self.labelLayout.addWidget(self.labelRed)
self.labelLayout.addWidget(self.labelGreen)
self.labelLayout.addWidget(self.labelBlue)

self.vlayout.addLayout(self.labelLayout)

QObject.connect(self.dialRed, SIGNAL("valueChanged(int)"),
                self.slotSetRed)
QObject.connect(self.dialGreen, SIGNAL("valueChanged(int)"),
                self.slotSetGreen)
QObject.connect(self.dialBlue, SIGNAL("valueChanged(int)"),
                self.slotSetBlue)

QObject.connect(self.dialRed, SIGNAL("valueChanged(int)"),
                self.slotSetColor)
QObject.connect(self.dialGreen, SIGNAL("valueChanged(int)"),
                self.slotSetColor)
QObject.connect(self.dialBlue, SIGNAL("valueChanged(int)"),
                self.slotSetColor)

def slotSetRed(self, value):
    self.labelRed.setText("Red: " + str(value))
    self.red = value

def slotSetGreen(self, value):
    self.labelGreen.setText("Green: " + str(value))
    self.green = value

def slotSetBlue(self, value):
    self.labelBlue.setText("Blue: " + str(value))
    self.blue = value

def slotSetColor(self, value):
    # self.setBackgroundColor(QColor(self.red, self.green, self.blue))
    # self.labelRed.setBackgroundColor(QColor(self.red, 128, 128))
    # self.labelGreen.setBackgroundColor(QColor(128, self.green, 128))
    # self.labelBlue.setBackgroundColor(QColor(128, 128, self.blue))
    pass

```

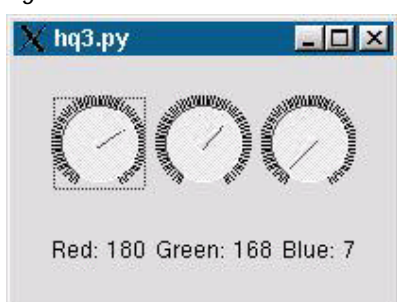
```

def main(args):
    app=QApplication(args)
    win=MainWindow()
    win.show()
    app.connect(app, SIGNAL("lastWindowClosed()")
               , app
               , SLOT("quit()")
               )
    app.exec_loop()

if __name__=="__main__":
    main(sys.argv

```

Figure 44 User-controlled feedback



GUI systems tend to be built around three elements:

- Visible component objects (also known as widgets)
- Geometry and layout managers to control how widgets relate on display
- Events which handle happenings such as button presses

An object oriented language such as Python is ideal for such a system and it's been argued that the support of Qt under Python is better and more natural than Qt under its native C++.

QWidgets

Widgets in Qt are all subclassed off the QWidget – there's QLabel, QPushButton, QSlider and many more. Widgets like QFrame are used to group other widgets, and you'll also use QMainWindow for your top-level window(s).

Objects such as QFont and QColor, are not visible objects (widgets) in their own right, but are associated with widget objects.

QLayouts

Widgets are laid out using subclasses of QLayout such as QGridLayout.

QEvents

Events are handled using Q Events, but there's much more to this in Qt.

In a classic GUI design, the programmer assigns an action to each event and widget combination. This can be as basic as storing the code to be run when an event occurs within a widget definition. Such an approach is not easily expandible, and the first stage of improvement is for the programmer to provide an event handling broker system whereby all events call to a central registry from where appropriate actions are instigated, rather in the manner of a switchboard.

In Qt, the event handling broker is provided for you within the toolkit:

- Possible actions are defined as **slots**
- Events that call the broker are known as **signals**

You can connect as many (or as few) slots as you wish to any particular signal, thus building up the interface from your widgets to your application.

20.5 A simple but practical control

Having introduced you to many of the terms involved with Qt, would you like to see a simple control? Let's see an application (*qupdown.py*) that creates buttons labelled "higher" and "lower" that control a label which reports on the current value of a variable:

```
import sys
from qt import *

class MainWindow(QMainWindow):
    val = 17
    def __init__(self, *args):
        apply(QMainWindow.__init__, (self, ) + args)

        self.vlayout = QHBoxLayout(self, 10, 5)

        self.labelValue = QLabel(str(MainWindow.val), self)
        self.down = QPushButton("Lower", self)
        self.up = QPushButton("Higher", self)

        self.vlayout.addWidget(self.down)
        self.vlayout.addWidget(self.labelValue)
        self.vlayout.addWidget(self.up)

        self.connect(self.down, SIGNAL("clicked()"), self.reduce)
        self.connect(self.up, SIGNAL("clicked()"), self.increase)

    def reduce(self):
        MainWindow.val -= 1
        self.setval()

    def increase(self):
        MainWindow.val += 1
        self.setval()

    def setval(self):
        self.labelValue.setText(str(MainWindow.val))

def main(args):
    app=QApplication(args)
    win=MainWindow()
    win.show()
    app.connect(app, SIGNAL("lastWindowClosed()")
                , app
                , SLOT("quit()")
                )
    app.exec_loop()

if __name__=="__main__":
    main(sys.argv)
```

20.6 Where to learn the use of Python/Qt applications

This module of training notes introduces you to the use of the Qt toolkit within the Python programming language, and is usually presented in association with Well House Consultants' Python training courses. See <http://www.wellho.net> for further

details. You may also like to look at <http://www.python.org>.

Well House Consultants don't offer Qt training; there simply aren't enough hours in the day for us to be experts in everything. If you're familiar with other GUI builder systems such as Tk, it's not terribly hard to learn. If you aren't, then you might like to start off with a book such as Sams Teach Yourself Qt Programming in 24 hours which has good coverage of the fundamental concepts. O'Reilly also publish a book – Programming with Qt – which is good as you get in a little deeper to the subject.

Figure 45 Three recommended Qt books



The standard Qt texts are based around examples in C++ – the home language for Qt – and don't cover the Python interface. GUI Programming with Python using the Qt Toolkit by Boudewijn Rempt does cover the subject, but is not an easy book to obtain. It's open source, and if you're happy to read it on line, visit <http://www.opendocs.org/pyqt/>.

20.7 BlackAdder

Since Python and Qt are a great GUI application development suite, you would expect to find some interesting applications out there. One such that merits a short note is BlackAdder. "BlackAdder is an application development environment that allows professional and hobbyist programmers alike to produce complex applications for the Windows and Linux platforms.

"BlackAdder brings together the Python programming language, the Ruby development language (Ruby is temporarily removed until the Qt bindings are updated to Qt3, currently in progress), the Qt graphical user interface (GUI) toolkit, ODBC database connectivity and an Integrated Development Environment (IDE) that includes an editor, a GUI designer, a debugger and an interactive Python interpreter. (A Ruby interpreter, debugger and ODBC support for Ruby are not yet available.) BlackAdder gives the programmer, in a single package, all they need to develop sophisticated applications."

Demonstration versions of BlackAdder are available for download, but if you want to use it beyond a 30-day evaluation, it's a commercial product you'll need to buy.



Exercise

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.