Python Programming/PyQt4

WARNING: The examples on this page are a mixture of PyQt3 and PyQt4 - use with caution!

This tutorial aims to provide a hands-on guide to learn the basics of building a small Qt4 application in Python.

To follow this tutorial, you should have basic Python knowledge. However, knowledge of Qt4 is not necessary. I'm using Linux in these examples and am assuming you already have a working installation of Python and PyQt4. To test this, open a Python shell (by typing 'Python' in a console to start the interactive interpreter) and type:

```
>>> import PyQt4
```

If no error message appears, you should be ready to go.

The examples in this tutorial as easy as possible, showing useful ways to write and structure your program. It is important that you read the source code of the example files, most of the explanations are in the code. The best way to get comfortable with PyQt is play around with the examples and try to change things.

Hello, world!

Let's start easy: popping up a window and displaying something. The following small program will pop up a window showing "Hello, world!".

```
#!/usr/bin/env python

import sys
from PyQt4 import Qt

# We instantiate a QApplication passing the arguments of the script to
it:
a = Qt.QApplication(sys.argv)

# Add a basic widget to this application:
# The first argument is the text we want this QWidget to show, the
second
# one is the parent widget. Since Our "hello" is the only thing we use
(the
# so-called "MainWidget", it does not have a parent.
hello = Qt.QLabel("Hello, World")

# ... and that it should be shown.
hello.show()

# Now we can start it.
a.exec_()
```

About 7 lines of code, and that's about as easy as it can get.

A Button

Let's add some interaction! We'll replace the label saying "Hello, World!" with a button and assign an action to it. This assignment is done by connecting a **signal**, an event which is sent out when the button is pushed, to a **slot**, which is an action, normally a function that is run in the case of that event.

```
#!/usr/bin/env python

import sys
from PyQt4 import Qt

a = Qt.QApplication(sys.argv)

# Our function to call when the button is clicked
def sayHello():
    print "Hello, World!"

# Instantiate the button
hellobutton = Qt.QPushButton("Say 'Hello world!'",None)

# And connect the action "sayHello" to the event "button has been
clicked"
a.connect(hellobutton, Qt.SIGNAL("clicked()"), sayHello)

# The rest is known already...
#a.setMainWidget(hellobutton)
hellobutton.show()
a.exec_()
```

You can imagine that coding this way is not scalable nor the way you'll want to continue working. So let's make that stuff pythonic, adding structure and actually using object-orientation in it. We create our own application class, derived from a QApplication and put the customization of the application into its methods: One method to build up the widgets and a slot which contains the code that's executed when a signal is received.

```
#!/usr/bin/env python

import sys
from PyQt4 import Qt

class HelloApplication(Qt.QApplication):

    def __init__(self, args):
        """ In the constructor we're doing everything to get our
application
            started, which is basically constructing a basic
QApplication by
            its __init__ method, then adding our widgets and finally
starting
            the exec_loop."""
```

```
        Qt.QApplication.__init__(self, args)
        self.addWidgets()
        self.exec_()

    def addWidgets(self):
        """ In this method, we're adding widgets and connecting signals
 from
            these widgets to methods of our class, the so-called
"slots"
        """
        self.hellobutton = Qt.QPushButton("Say 'Hello world!'",None)
        self.connect(self.hellobutton, Qt.SIGNAL("clicked()"),
self.slotSayHello)
        self.hellobutton.show()

    def slotSayHello(self):
        """ This is an example slot, a method that gets called when a
signal is
            emitted """
        print "Hello, World!"

# Only actually do something if this script is run standalone, so we
can test our
# application, but we're also able to import this program without
actually running
# any code.
if __name__ == "__main__":
    app = HelloApplication(sys.argv)
```

GUI Coding

... so we want to use Qt3 Designer for creating our GUI. In the picture, you can see a simple GUI, with in green letters the names of the widgets. What we are going to do is We compile the .ui file from Qt designer into a python class We subclass that class and use it as our mainWidget This way, we're able to change the user interface afterwards from Qt designer, without having it messing around in the code we added.

```
pyuic testapp_ui.ui -o testapp_ui.py
```

makes a Python file from it which we can work with.

The way our program works can be described like this: We fill in the lineedit Clicking the add button will be connected to a method that reads the text from the lineedit, makes a listviewitem out of it and adds that to our listview. Clicking the deletebutton will delete the currently selected item from the listview. Here's the heavily commented code (only works in PyQt3):

```
#!/usr/bin/env python

from testapp_ui import TestAppUI
from qt import *
import sys
```

```
class HelloApplication(QApplication):

    def __init__(self, args):
        """ In the constructor we're doing everything to get our
application
            started, which is basically constructing a basic
QApplication by
            its __init__ method, then adding our widgets and finally
starting
            the exec_loop."""
        QApplication.__init__(self,args)

        # We pass None since it's the top-level widget, we could in
fact leave
        # that one out, but this way it's easier to add more dialogs or
 widgets.
        self.maindialog = TestApp(None)

        self.setMainWidget(self.maindialog)
        self.maindialog.show()
        self.exec_loop()

class TestApp(TestAppUI):

    def __init__(self,parent):
        # Run the parent constructor and connect the slots to methods.
        TestAppUI.__init__(self,parent)
        self._connectSlots()

        # The listview is initially empty, so the deletebutton will
have no effect,
        # we grey it out.
        self.deletebutton.setEnabled(False)

    def _connectSlots(self):
        # Connect our two methods to SIGNALS the GUI emits.

self.connect(self.addbutton,SIGNAL("clicked()"),self._slotAddClicked)

self.connect(self.deletebutton,SIGNAL("clicked()"),self._slotDeleteClicked)

    def _slotAddClicked(self):
        # Read the text from the lineedit,
        text = self.lineedit.text()
        # if the lineedit is not empty,
        if len(text):
```

```
            # insert a new listviewitem ...
            lvi = QListViewItem(self.listview)
            # with the text from the lineedit and ...
            lvi.setText(0,text)
            # clear the lineedit.
            self.lineedit.clear()

            # The deletebutton might be disabled, since we're sure that
 there's now
            # at least one item in it, we enable it.
            self.deletebutton.setEnabled(True)

    def _slotDeleteClicked(self):
        # Remove the currently selected item from the listview.
        self.listview.takeItem(self.listview.currentItem())

        # Check if the list is empty - if yes, disable the
deletebutton.
        if self.listview.childCount() == 0:
            self.deletebutton.setEnabled(False)

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

also this code is useful and it works on PyQt4 and it have many useful options

```
#!/usr/bin/env python
# Copyright (c) 2008-10 Qtrac Ltd. All rights reserved.
# This program or module is free software: you can redistribute it
and/or
# modify it under the terms of the GNU General Public License as
published
# by the Free Software Foundation, either version 2 of the License, or
# version 3 of the License, or (at your option) any later version. It
is
# provided for educational purposes and is distributed in the hope that
# it will be useful, but WITHOUT ANY WARRANTY; without even the implied
# warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
# the GNU General Public License for more details.
#
#
#   Versions
#
# 1.0.1 Fixed bug reported by Brian Downing where paths that contained
#       spaces were not handled correctly.
# 1.0.2 Fixed bug reported by Ben Thompson that if the UIC program
#       fails, no problem was reported; I try to report one now.
# 1.1.0 Added Remember path option; if checked the program starts with
```

```
#       the last used path, otherwise with the current directory,
unless
#       overridden on the command line
# 1.1.1 Changed default path on Windows to match PyQt 4.4
# 1.2.1 Changed import style + bug fixes
# 1.2.2 Added stderr to error message output as per Michael Jackson's
#       suggestion
# 1.2.3 Tried to make the paths work on Mac OS X
# 1.2.4 Added more options

from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
from future_builtins import *

import os
import platform
import stat
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

__version__ = "1.2.5"

Windows = sys.platform.lower().startswith(("win", "microsoft"))

class OptionsForm(QDialog):

    def __init__(self, parent=None):
        super(OptionsForm, self).__init__(parent)

        settings = QSettings()
        if sys.platform.startswith("darwin"):
            pyuic4Label = QLabel("pyuic4 (pyuic.py)")
        else:
            pyuic4Label = QLabel("pyuic4")
        self.pyuic4Label = QLabel(settings.value("pyuic4",
                QVariant(PYUIC4)).toString())
        self.pyuic4Label.setFrameStyle(QFrame.StyledPanel|
                                       QFrame.Sunken)
        pyuic4Button = QPushButton("py&uic4...")
        pyrcc4Label = QLabel("pyrcc4")
        self.pyrcc4Label = QLabel(settings.value("pyrcc4",
                QVariant(PYRCC4)).toString())
        self.pyrcc4Label.setFrameStyle(QFrame.StyledPanel|
                                       QFrame.Sunken)
        pyrcc4Button = QPushButton("p&yrcc4...")
```

```
        pylupdate4Label = QLabel("pylupdate4")
        self.pylupdate4Label = QLabel(settings.value("pylupdate4",
                QVariant(PYLUPDATE4)).toString())
        self.pylupdate4Label.setFrameStyle(QFrame.StyledPanel|
                                           QFrame.Sunken)
        pylupdate4Button = QPushButton("&pylupdate4...")
        lreleaseLabel = QLabel("lrelease")
        self.lreleaseLabel = QLabel(settings.value("lrelease",
                QVariant("lrelease")).toString())
        self.lreleaseLabel.setFrameStyle(QFrame.StyledPanel|
                                         QFrame.Sunken)
        lreleaseButton = QPushButton("&lrelease...")
        toolPathGroupBox = QGroupBox("Tool Paths")

        pathsLayout = QGridLayout()
        pathsLayout.addWidget(pyuic4Label, 0, 0)
        pathsLayout.addWidget(self.pyuic4Label, 0, 1)
        pathsLayout.addWidget(pyuic4Button, 0, 2)
        pathsLayout.addWidget(pyrcc4Label, 1, 0)
        pathsLayout.addWidget(self.pyrcc4Label, 1, 1)
        pathsLayout.addWidget(pyrcc4Button, 1, 2)
        pathsLayout.addWidget(pylupdate4Label, 2, 0)
        pathsLayout.addWidget(self.pylupdate4Label, 2, 1)
        pathsLayout.addWidget(pylupdate4Button, 2, 2)
        pathsLayout.addWidget(lreleaseLabel, 3, 0)
        pathsLayout.addWidget(self.lreleaseLabel, 3, 1)
        pathsLayout.addWidget(lreleaseButton, 3, 2)
        toolPathGroupBox.setLayout(pathsLayout)

        resourceModuleNamesGroupBox = QGroupBox(
                "Resource Module Names")
        qrcFiles = bool(int(settings.value("qrc_resources",
"1").toString()))
        self.qrcRadioButton = QRadioButton("&qrc_file.py")
        self.qrcRadioButton.setChecked(qrcFiles)
        self.rcRadioButton = QRadioButton("file_&rc.py")
        self.rcRadioButton.setChecked(not qrcFiles)

        radioLayout = QHBoxLayout()
        radioLayout.addWidget(self.qrcRadioButton)
        radioLayout.addWidget(self.rcRadioButton)
        resourceModuleNamesGroupBox.setLayout(radioLayout)

        self.pyuic4xCheckBox = QCheckBox("Run pyuic4 with -&x "
                " to make forms stand-alone runable")
        x = bool(int(settings.value("pyuic4x", "0").toString()))
        self.pyuic4xCheckBox.setChecked(x)
```

```
        buttonBox = QDialogButtonBox(QDialogButtonBox.Ok|
                                     QDialogButtonBox.Cancel)

        layout = QVBoxLayout()
        layout.addWidget(toolPathGroupBox)
        layout.addWidget(resourceModuleNamesGroupBox)
        layout.addWidget(self.pyuic4xCheckBox)
        layout.addWidget(buttonBox)
        self.setLayout(layout)

        self.connect(pyuic4Button, SIGNAL("clicked()"),
                lambda: self.setPath("pyuic4"))
        self.connect(pyrcc4Button, SIGNAL("clicked()"),
                lambda: self.setPath("pyrcc4"))
        self.connect(pylupdate4Button, SIGNAL("clicked()"),
                lambda: self.setPath("pylupdate4"))
        self.connect(lreleaseButton, SIGNAL("clicked()"),
                lambda: self.setPath("lrelease"))
        self.connect(buttonBox, SIGNAL("accepted()"), self.accept)
        self.connect(buttonBox, SIGNAL("rejected()"), self.reject)

        self.setWindowTitle("Make PyQt - Options")


    def accept(self):
        settings = QSettings()
        settings.setValue("pyuic4", QVariant(self.pyuic4Label.text()))
        settings.setValue("pyrcc4", QVariant(self.pyrcc4Label.text()))
        settings.setValue("pylupdate4",
                QVariant(self.pylupdate4Label.text()))
        settings.setValue("lrelease",
QVariant(self.lreleaseLabel.text()))
        settings.setValue("qrc_resources",
                "1" if self.qrcRadioButton.isChecked() else "0")
        settings.setValue("pyuic4x",
                "1" if self.pyuic4xCheckBox.isChecked() else "0")
        QDialog.accept(self)


    def setPath(self, tool):
        if tool == "pyuic4":
            label = self.pyuic4Label
        elif tool == "pyrcc4":
            label = self.pyrcc4Label
        elif tool == "pylupdate4":
            label = self.pylupdate4Label
```

```
        elif tool == "lrelease":
            label = self.lreleaseLabel
        path = QFileDialog.getOpenFileName(self,
                "Make PyQt - Set Tool Path", label.text())
        if path:
            label.setText(QDir.toNativeSeparators(path))


class Form(QMainWindow):

    def __init__(self):
        super(Form, self).__init__(None)

        pathLabel = QLabel("Path:")
        settings = QSettings()
        rememberPath = settings.value("rememberpath",
                QVariant(True if Windows else False)).toBool()
        if rememberPath:
            path = (unicode(settings.value("path").toString()) or
                    os.getcwd())
        else:
            path = (sys.argv[1] if len(sys.argv) > 1 and
                    QFile.exists(sys.argv[1]) else os.getcwd())
        self.pathLabel = QLabel(path)
        self.pathLabel.setFrameStyle(QFrame.StyledPanel|
                                     QFrame.Sunken)
        self.pathLabel.setToolTip("The relative path; all actions will
"
                "take place here,<br>and in this path's subdirectories "
                "if the Recurse checkbox is checked")
        self.pathButton = QPushButton("&Path...")
        self.pathButton.setToolTip(self.pathLabel.toolTip().replace(
                "The", "Sets the"))
        self.recurseCheckBox = QCheckBox("&Recurse")
        self.recurseCheckBox.setToolTip("Clean or build all the files "
                "in the path directory,<br>and all its subdirectories, "
                "as deep as they go.")
        self.transCheckBox = QCheckBox("&Translate")
        self.transCheckBox.setToolTip("Runs <b>pylupdate4</b> on all "
                "<tt>.py</tt> and <tt>.pyw</tt> files in conjunction "
                "with each <tt>.ts</tt> file.<br>Then runs "
                "<b>lrelease</b> on all <tt>.ts</tt> files to produce "
                "corresponding <tt>.qm</tt> files.<br>The "
                "<tt>.ts</tt> files must have been created initially by "
                "running <b>pylupdate4</b><br>directly on a <tt>.py</tt> "
                "or <tt>.pyw</tt> file using the <tt>-ts</tt> option.")
        self.debugCheckBox = QCheckBox("&Dry Run")
```

```
        self.debugCheckBox.setToolTip("Shows the actions that would "
                "take place but does not do them.")
        self.logBrowser = QTextBrowser()
        self.logBrowser.setLineWrapMode(QTextEdit.NoWrap)
        self.buttonBox = QDialogButtonBox()
        menu = QMenu(self)
        optionsAction = menu.addAction("&Options...")
        self.rememberPathAction = menu.addAction("&Remember path")
        self.rememberPathAction.setCheckable(True)
        self.rememberPathAction.setChecked(rememberPath)
        aboutAction = menu.addAction("&About")
        moreButton = self.buttonBox.addButton("&More",
                QDialogButtonBox.ActionRole)
        moreButton.setMenu(menu)
        moreButton.setToolTip("Use <b>More-&gt;Tool paths</b> to set the "
                "paths to the tools if they are not found by default")
        self.buildButton = self.buttonBox.addButton("&Build",
                QDialogButtonBox.ActionRole)
        self.buildButton.setToolTip("Runs <b>pyuic4</b> on all "
                "<tt>.ui</tt> "
                "files and <b>pyrcc4</b> on all <tt>.qrc</tt> files "
                "that are out-of-date.<br>Also runs <b>pylupdate4</b> "
                "and <b>lrelease</b> if the Translate checkbox is "
                "checked.")
        self.cleanButton = self.buttonBox.addButton("&Clean",
                QDialogButtonBox.ActionRole)
        self.cleanButton.setToolTip("Deletes all <tt>.py</tt> files that "
                "were generated from <tt>.ui</tt> and <tt>.qrc</tt> "
                "files,<br>i.e., all files matching <tt>qrc_*.py</tt>, "
                "<tt>*_rc.py</tt> and <tt>ui_*.py.")
        quitButton = self.buttonBox.addButton("&Quit",
                QDialogButtonBox.RejectRole)

        topLayout = QHBoxLayout()
        topLayout.addWidget(pathLabel)
        topLayout.addWidget(self.pathLabel, 1)
        topLayout.addWidget(self.pathButton)
        bottomLayout = QHBoxLayout()
        bottomLayout.addWidget(self.recurseCheckBox)
        bottomLayout.addWidget(self.transCheckBox)
        bottomLayout.addWidget(self.debugCheckBox)
        bottomLayout.addStretch()
        bottomLayout.addWidget(self.buttonBox)
        layout = QVBoxLayout()
        layout.addLayout(topLayout)
        layout.addWidget(self.logBrowser)
        layout.addLayout(bottomLayout)
```

```
        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)

        self.connect(aboutAction, SIGNAL("triggered()"), self.about)
        self.connect(optionsAction, SIGNAL("triggered()"),
self.setOptions)
        self.connect(self.pathButton, SIGNAL("clicked()"),
self.setPath)
        self.connect(self.buildButton, SIGNAL("clicked()"), self.build)
        self.connect(self.cleanButton, SIGNAL("clicked()"), self.clean)
        self.connect(quitButton, SIGNAL("clicked()"), self.close)

        self.setWindowTitle("Make PyQt")


    def closeEvent(self, event):
        settings = QSettings()
        settings.setValue("rememberpath",
                QVariant(self.rememberPathAction.isChecked()))
        settings.setValue("path", QVariant(self.pathLabel.text()))
        event.accept()


    def about(self):
        QMessageBox.about(self, "About Make PyQt",
                """<b>Make PyQt</b> v {0}
                <p>Copyright &copy; 2007-10 Qtrac Ltd.
                All rights reserved.
                <p>This application can be used to build PyQt
                applications.
                It runs pyuic4, pyrcc4, pylupdate4, and lrelease as
                required, although pylupdate4 must be run directly to
                create the initial .ts files.
                <p>Python {1} - Qt {2} - PyQt {3} on {4}""".format(
                __version__, platform.python_version(),
                QT_VERSION_STR, PYQT_VERSION_STR,
                platform.system()))


    def setPath(self):
        path = QFileDialog.getExistingDirectory(self,
                "Make PyQt - Set Path", self.pathLabel.text())
        if path:
            self.pathLabel.setText(QDir.toNativeSeparators(path))
```

```
    def setOptions(self):
        dlg = OptionsForm(self)
        dlg.exec_()


    def build(self):
        self.updateUi(False)
        self.logBrowser.clear()
        recurse = self.recurseCheckBox.isChecked()
        path = unicode(self.pathLabel.text())
        self._apply(recurse, self._build, path)
        if self.transCheckBox.isChecked():
            self._apply(recurse, self._translate, path)
        self.updateUi(True)


    def clean(self):
        self.updateUi(False)
        self.logBrowser.clear()
        recurse = self.recurseCheckBox.isChecked()
        path = unicode(self.pathLabel.text())
        self._apply(recurse, self._clean, path)
        self.updateUi(True)


    def updateUi(self, enable):
        for widget in (self.buildButton, self.cleanButton,
                self.pathButton, self.recurseCheckBox,
                self.transCheckBox, self.debugCheckBox):
            widget.setEnabled(enable)
        if not enable:
            QApplication.setOverrideCursor(QCursor(Qt.WaitCursor))
        else:
            QApplication.restoreOverrideCursor()
            self.buildButton.setFocus()


    def _apply(self, recurse, function, path):
        if not recurse:
            function(path)
        else:
            for root, dirs, files in os.walk(path):
                for dir in sorted(dirs):
                    function(os.path.join(root, dir))


    def _make_error_message(self, command, process):
```

```
        err = ""
        ba = process.readAllStandardError()
        if not ba.isEmpty():
            err = ": " + str(QString(ba))
        return "<font color=red>FAILED: %s%s</font>" % (command, err)


    def _build(self, path):
        settings = QSettings()
        pyuic4 = unicode(settings.value("pyuic4",
                                        QVariant(PYUIC4)).toString())
        pyrcc4 = unicode(settings.value("pyrcc4",
                                        QVariant(PYRCC4)).toString())
        prefix = unicode(self.pathLabel.text())
        pyuic4x = bool(int(settings.value("pyuic4x", "0").toString()))
        if not prefix.endswith(os.sep):
            prefix += os.sep
        failed = 0
        process = QProcess()
        for name in os.listdir(path):
            source = os.path.join(path, name)
            target = None
            if source.endswith(".ui"):
                target = os.path.join(path,
                                      "ui_" + name.replace(".ui", ".py"))
                command = pyuic4
            elif source.endswith(".qrc"):
                if bool(int(settings.value("qrc_resources",
"1").toString())):
                    target = os.path.join(path,
                                          "qrc_" + name.replace(".qrc",
".py"))
                else:
                    target = os.path.join(path, name.replace(".qrc",
"_rc.py"))
                command = pyrcc4
            if target is not None:
                if not os.access(target, os.F_OK) or (
                   os.stat(source)[stat.ST_MTIME] >
                   os.stat(target)[stat.ST_MTIME]):
                    args = ["-o", target, source]
                    if command == PYUIC4 and pyuic4x:
                        args.insert(0, "-x")
                    if (sys.platform.startswith("darwin") and
                        command == PYUIC4):
                        command = sys.executable
                        args = [PYUIC4] + args
```

```
                    msg = ("converted <font color=darkblue>" + source +
                           "</font> to <font color=blue>" + target +
                           "</font>")
                    if self.debugCheckBox.isChecked():
                        msg = "<font color=green># " + msg + "</font>"
                    else:
                        process.start(command, args)
                        if (not process.waitForFinished(2 * 60 * 1000)
or
                            not QFile.exists(target)):
                            msg = self._make_error_message(command,
                                                           process)
                            failed += 1
                    self.logBrowser.append(msg.replace(prefix, ""))
                else:
                    self.logBrowser.append("<font color=green>"
                            "# {0} is up-to-date</font>".format(
                            source.replace(prefix, "")))
                QApplication.processEvents()
        if failed:
            QMessageBox.information(self, "Make PyQt - Failures",
                    "Try manually setting the paths to the tools "
                    "using <b>More-&gt;Options</b>")


    def _clean(self, path):
        prefix = unicode(self.pathLabel.text())
        if not prefix.endswith(os.sep):
            prefix += os.sep
        deletelist = []
        for name in os.listdir(path):
            target = os.path.join(path, name)
            source = None
            if (target.endswith(".py") or target.endswith(".pyc") or
                target.endswith(".pyo")):
                if name.startswith("ui_") and not name[-1] in "oc":
                    source = os.path.join(path, name[3:-3] + ".ui")
                elif name.startswith("qrc_"):
                    if target[-1] in "oc":
                        source = os.path.join(path, name[4:-4] +
".qrc")
                    else:
                        source = os.path.join(path, name[4:-3] +
".qrc")
                elif name.endswith(("_rc.py", "_rc.pyo", "_rc.pyc")):
                    if target[-1] in "oc":
                        source = os.path.join(path, name[:-7] + ".qrc")
```

```
                    else:
                        source = os.path.join(path, name[:-6] + ".qrc")
                elif target[-1] in "oc":
                    source = target[:-1]
                if source is not None:
                    if os.access(source, os.F_OK):
                        if self.debugCheckBox.isChecked():
                            self.logBrowser.append("<font color=green>"
                                    "# delete {0}</font>".format(
                                    target.replace(prefix, "")))
                        else:
                            deletelist.append(target)
                    else:
                        self.logBrowser.append("<font color=darkred>"
                                "will not remove "
                                "'{0}' since `{1}' not found</font>"
                                .format(target.replace(prefix, ""),
                                source.replace(prefix, "")))
        if not self.debugCheckBox.isChecked():
            for target in deletelist:
                self.logBrowser.append("deleted "
                        "<font color=red>{0}</font>".format(
                        target.replace(prefix, "")))
                os.remove(target)
                QApplication.processEvents()


    def _translate(self, path):
        prefix = unicode(self.pathLabel.text())
        if not prefix.endswith(os.sep):
            prefix += os.sep
        files = []
        tsfiles = []
        for name in os.listdir(path):
            if name.endswith((".py", ".pyw")):
                files.append(os.path.join(path, name))
            elif name.endswith(".ts"):
                tsfiles.append(os.path.join(path, name))
        if not tsfiles:
            return
        settings = QSettings()
        pylupdate4 = unicode(settings.value("pylupdate4",
                             QVariant(PYLUPDATE4)).toString())
        lrelease = unicode(settings.value("lrelease",
                           QVariant(LRELEASE)).toString())
        process = QProcess()
        failed = 0
```

```
        for ts in tsfiles:
            qm = ts[:-3] + ".qm"
            command1 = pylupdate4
            args1 = files + ["-ts", ts]
            command2 = lrelease
            args2 = ["-silent", ts, "-qm", qm]
            msg = "updated <font color=blue>{0}</font>".format(
                    ts.replace(prefix, ""))
            if self.debugCheckBox.isChecked():
                msg = "<font color=green># {0}</font>".format(msg)
            else:
                process.start(command1, args1)
                if not process.waitForFinished(2 * 60 * 1000):
                    msg = self._make_error_message(command1, process)
                    failed += 1
            self.logBrowser.append(msg)
            msg = "generated <font color=blue>{0}</font>".format(
                    qm.replace(prefix, ""))
            if self.debugCheckBox.isChecked():
                msg = "<font color=green># {0}</font>".format(msg)
            else:
                process.start(command2, args2)
                if not process.waitForFinished(2 * 60 * 1000):
                    msg = self._make_error_message(command2, process)
                    failed += 1
            self.logBrowser.append(msg)
            QApplication.processEvents()
        if failed:
            QMessageBox.information(self, "Make PyQt - Failures",
                    "Try manually setting the paths to the tools "
                    "using <b>More-&gt;Options</b>")


app = QApplication(sys.argv)
PATH = unicode(app.applicationDirPath())
if Windows:
    PATH = os.path.join(os.path.dirname(sys.executable),
                        "Lib/site-packages/PyQt4")
    if os.access(os.path.join(PATH, "bin"), os.R_OK):
        PATH = os.path.join(PATH, "bin")
if sys.platform.startswith("darwin"):
    i = PATH.find("Resources")
    if i > -1:
        PATH = PATH[:i] + "bin"
PYUIC4 = os.path.join(PATH, "pyuic4")
if sys.platform.startswith("darwin"):
    PYUIC4 = os.path.dirname(sys.executable)
```

```
    i = PYUIC4.find("Resources")
    if i > -1:
        PYUIC4 = PYUIC4[:i] +
"Lib/python2.6/site-packages/PyQt4/uic/pyuic.py"
PYRCC4 = os.path.join(PATH, "pyrcc4")
PYLUPDATE4 = os.path.join(PATH, "pylupdate4")
LRELEASE = "lrelease"
if Windows:
    PYUIC4 = PYUIC4.replace("/", "\\") + ".bat"
    PYRCC4 = PYRCC4.replace("/", "\\") + ".exe"
    PYLUPDATE4 = PYLUPDATE4.replace("/", "\\") + ".exe"
app.setOrganizationName("Qtrac Ltd.")
app.setOrganizationDomain("qtrac.eu")
app.setApplicationName("Make PyQt")
if len(sys.argv) > 1 and sys.argv[1] == "-c":
    settings = QSettings()
    settings.setValue("pyuic4", QVariant(PYUIC4))
    settings.setValue("pyrcc4", QVariant(PYRCC4))
    settings.setValue("pylupdate4", QVariant(PYLUPDATE4))
    settings.setValue("lrelease", QVariant(LRELEASE))
form = Form()
form.show()
app.exec_()
```

Useful to Know

Creating the GUI in Qt Designer not only makes creating the GUI easier, but it's also a great learning tool. You can test what a widget looks like, see what's available in Qt, and have a look at properties you might want to use.

The C++ API documentation is also a very useful (read: necessary) tool when working with PyQt. The API translation is straightforward, and after a little experience, you'll find the developers API docs one of the tools you really need. When working from KDE, konqueror's default shortcut is qt:[widgetname], so [alt]+[F2], "qt:qbutton directly takes you to the right API documentation page. Trolltech's doc section has much more documentation which you might want to have a look at.

The first 3 examples in this tutorial have been created using PyQt4, the last one uses syntax that only works with PyQt3.

Note: The previous version of this page (aplicable to pyqt3) is/was available at http://vizzzion.org/?id=pyqt.

Article Sources and Contributors

Python Programming/PyQt4 *Source*: https://en.wikibooks.org/w/index.php?oldid=2348547 *Contributors*: Baijum81, Chuckhoffmann, Danh, Fishpi, Herbythyme, JackPotte, Mr.Z-man, Sasa.tomic, 19 anonymous edits

License