Getting started with PyQt for Maemo

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

The article is believed to be still valid for the original topic scope.

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee
(28 Apr 2011)
The instructions have not been tested for some time and are unlikely to be correct in terms of installation and SDK. It would also be useful to have similar instructions for PySide, the alternative (and better licensed) Python on Qt Binding.

This article shows how to install Qt4 and PyQt4 on Nokia N800 and illustrates PyQt usage with a very simple example. Qt is a cross-platform application framework that makes it easier to develop applications and user interfaces. PyQt is a set of bindings that allows programmers to access Qt library using Python.

22 Mar 2009

Prerequisites

The prerequisites for installing Qt4 and PyQt4 on N800 are:

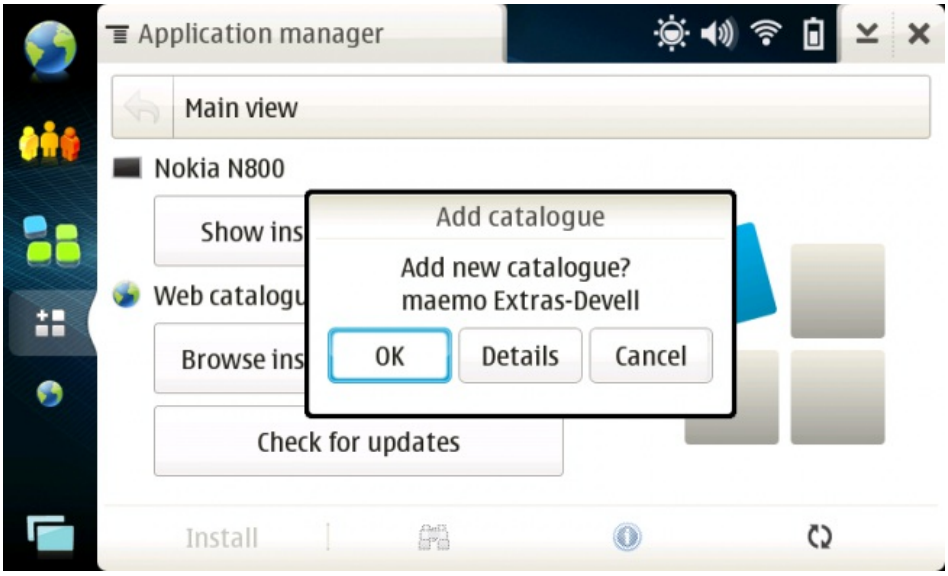
1. Have Maemo OS2008 installed in your device
2. Have Python environment installed in your device

Install Qt4 on N800

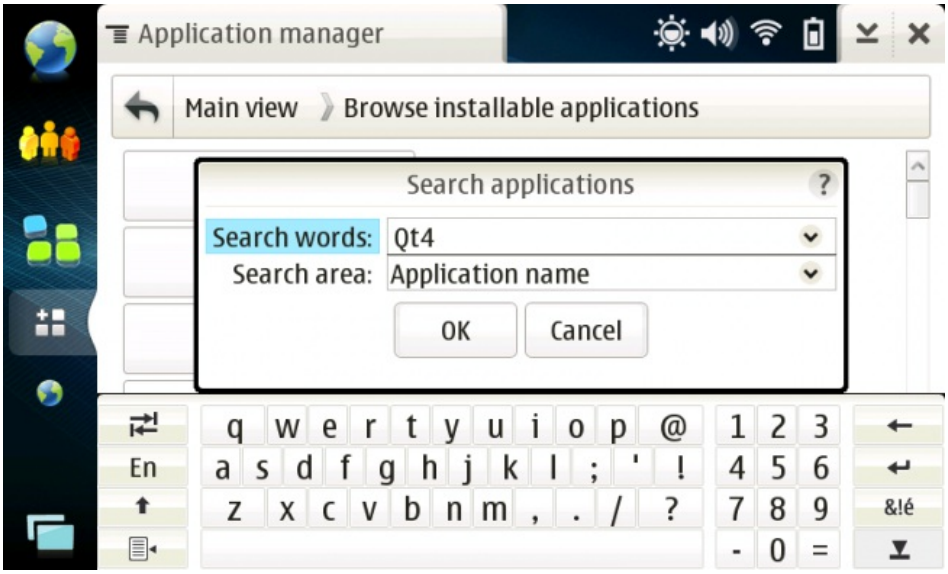
The first thing we need to do is to enable the extras-devel repository on your N800. Go to http://qt4.garage.maemo.org using the N800's web browser. At the middle of the page there is a button labeled *Click to install DEVEL*. Click on it.

It will open the *File Download* box. Click on the *Open* button. After the download finishes, the *Add Catalogue* box will appear. Click on *OK*.

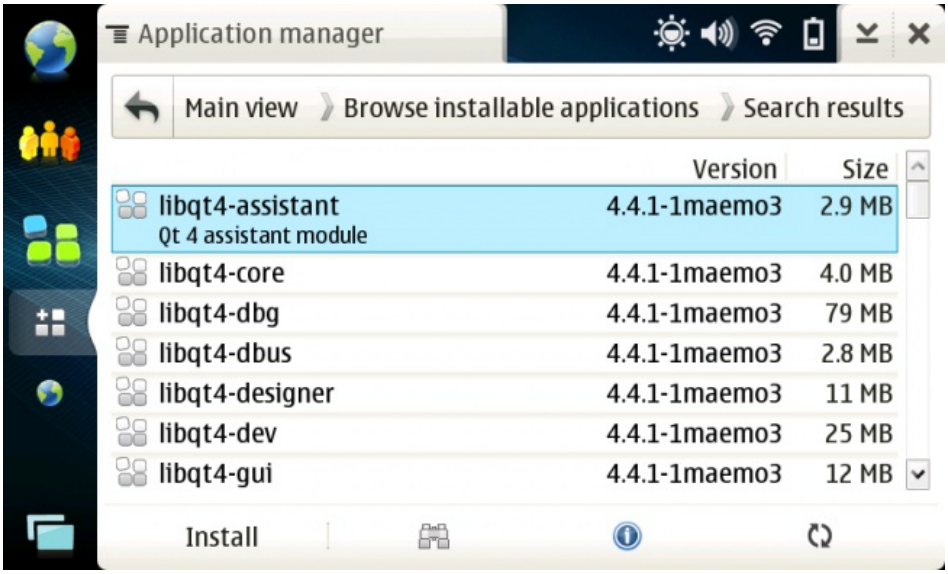
(C) Copyright Nokia 2013. All rights reserved.



We need to install the Qt4 packages that we will need to build our applications. Go to *Application Manager* at N800, click on *Browse installable applications* and on the find button. Let's search for Qt4.

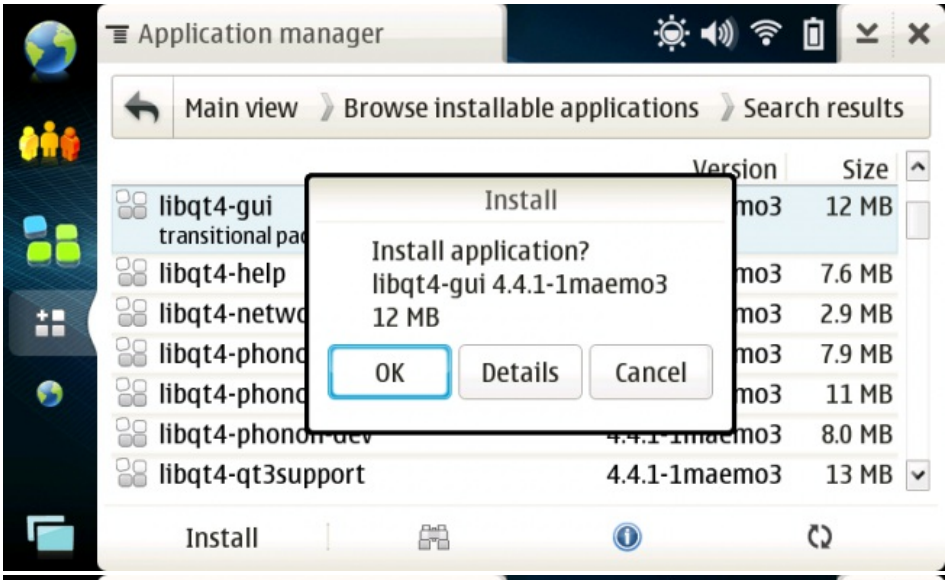


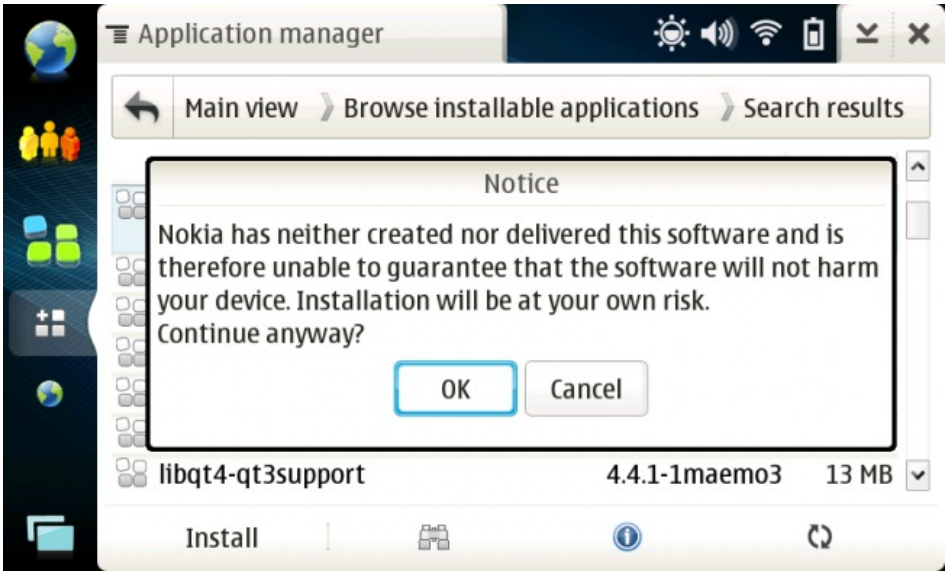
Now we have a list of all packages of Qt4 that we can install. For this introductory example, we will need to install `libqt4-core` and `libqt4-gui`.

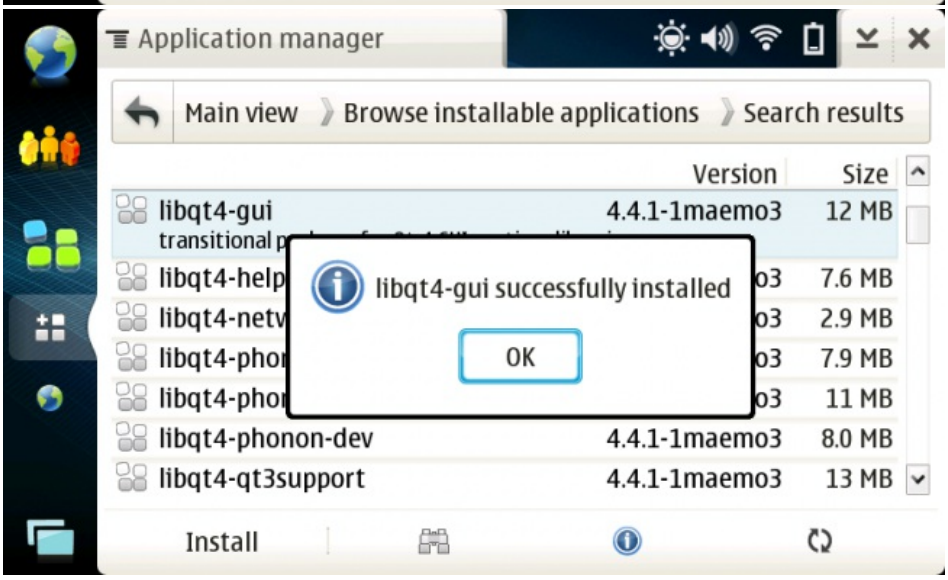


Let's install `libqt4-gui`. Click on `libqt4-gui` and click on *Install*. Then, click on *OK* button.

(C) Copyright Nokia 2013. All rights reserved.





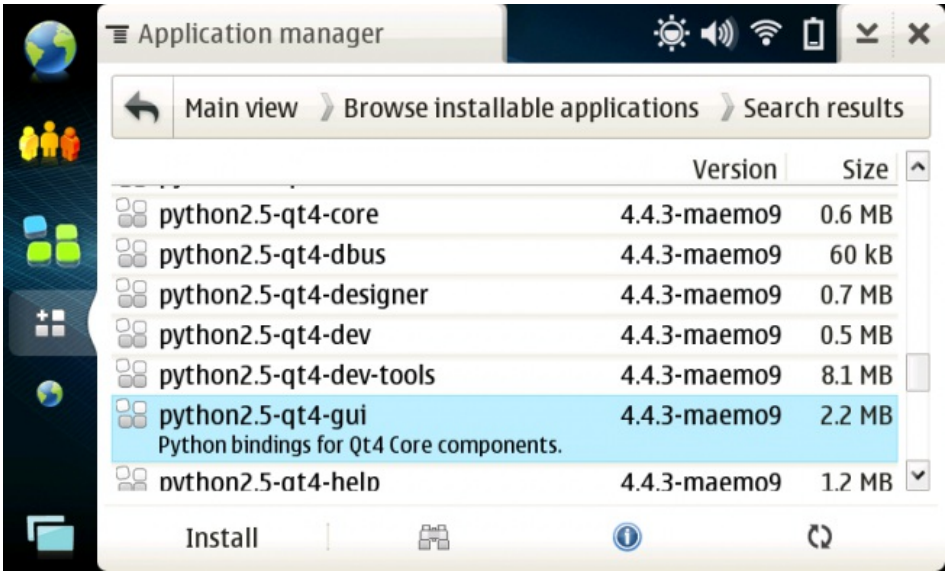


Now search again for Qt4 and install `libqt4-core`. Repeat the procedure as described on the installation of `libqt4-gui`.

Install PyQt4 on N800

The next step is to install the pyqt bindings for qt4-core and qt4-gui libraries. If we go again to *Browse installable applications* and search for Qt4, at the end of the page we will find the pyqt packages. Install `python2.5-qt4-core` and `python2.5-qt4-gui`.

(C) Copyright Nokia 2013. All rights reserved.



Creating a text editor application

Let us create a very simple text editor in Python using Qt to develop the UI. First of all, create a file called `texteditor.py` anywhere at your computer. It will be our Python application.

The first thing we need to do is import what we will need.

```
import sys
from PyQt4 import QtGui
from PyQt4 import QtCore
```

After that, create a `QMainWindow` that will represent our text editor. The `QMainWindow` is a widget.

```
class TextEditor(QtGui.QMainWindow):

    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setWindowTitle('My Text Editor')
```

Now, create the method `main`. It will start our application.

```
if __name__ == '__main__':

    #Creating Qt application
    app = QtGui.QApplication(sys.argv)

    myTextEditor = TextEditor()
    myTextEditor.show()

    #Initing application
    sys.exit(app.exec_())
```

Until now, the file `texteditor.py` looks like this:

```
import sys
from PyQt4 import QtGui
from PyQt4 import QtCore

class TextEditor(QtGui.QMainWindow):

    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setWindowTitle('My Text Editor')

if __name__ == '__main__':

    #Creating Qt application
    app = QtGui.QApplication(sys.argv)

    myTextEditor = TextEditor()
    myTextEditor.show()

    #Initing application
    sys.exit(app.exec_())
```

Now, test it on the N800 to know if it works. Copy this file to the device using `scp` and remote login on the N800 using `ssh`. Execute `texteditor.py` with `python texteditor.py`. If everything is OK, you will see this:

(C) Copyright Nokia 2013. All rights reserved.

Create the text area with `QTextEdit` and set the central widget of `TextEditor`.

```
class TextEditor(QtGui.QMainWindow):

    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setWindowTitle('My Text Editor')

        self.__textEdit = QtGui.QTextEdit()
        self.setCentralWidget(self.__textEdit)
```

The next step is to create the menu and put the *New*, *Open*, *Save*, *Save As* and *Exit* options. First, create a `QAction` for each action. Pay attention that when someone chooses an option, the method at the last parameter of the connect will react to the signal.

```
        exitAction = QtGui.QAction('Exit', self)
        self.connect(exitAction, QtCore.SIGNAL('triggered()'), self.close)

        newFileAction = QtGui.QAction('New', self)
        self.connect(newFileAction, QtCore.SIGNAL('triggered()'), self.newFile)

        openAction = QtGui.QAction('Open', self)
        self.connect(openAction, QtCore.SIGNAL('triggered()'), self.open)

        saveAction = QtGui.QAction('Save', self)
        self.connect(saveAction, QtCore.SIGNAL('triggered()'), self.save)

        saveAsAction = QtGui.QAction('Save As', self)
        self.connect(saveAsAction, QtCore.SIGNAL('triggered()'), self.saveAs)
```

Now, add the actions to the menu bar. To finish the constructor, put a `fileName` variable to store the name of the file that is open. The init looks like this:

```
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setWindowTitle('My Text Editor')

        self.__textEdit = QtGui.QTextEdit()
        self.setCentralWidget(self.__textEdit)

        exitAction = QtGui.QAction('Exit', self)
        self.connect(exitAction, QtCore.SIGNAL('triggered()'), self.close)

        newFileAction = QtGui.QAction('New', self)
        self.connect(newFileAction, QtCore.SIGNAL('triggered()'), self.newFile)
```


(C) Copyright Nokia 2013. All rights reserved.

```
        openAction = QtGui.QAction('Open', self)
        self.connect(openAction, QtCore.SIGNAL('triggered()'), self.open)

        saveAction = QtGui.QAction('Save', self)
        self.connect(saveAction, QtCore.SIGNAL('triggered()'), self.save)

        saveAsAction = QtGui.QAction('Save As', self)
        self.connect(saveAsAction, QtCore.SIGNAL('triggered()'), self.saveAs)

        menubar = self.menuBar()
        file = menubar.addAction(newFileAction)
        file = menubar.addAction(openAction)
        file = menubar.addAction(saveAction)
        file = menubar.addAction(saveAsAction)
        file = menubar.addAction(exitAction)

        self.__fileName = ""
```

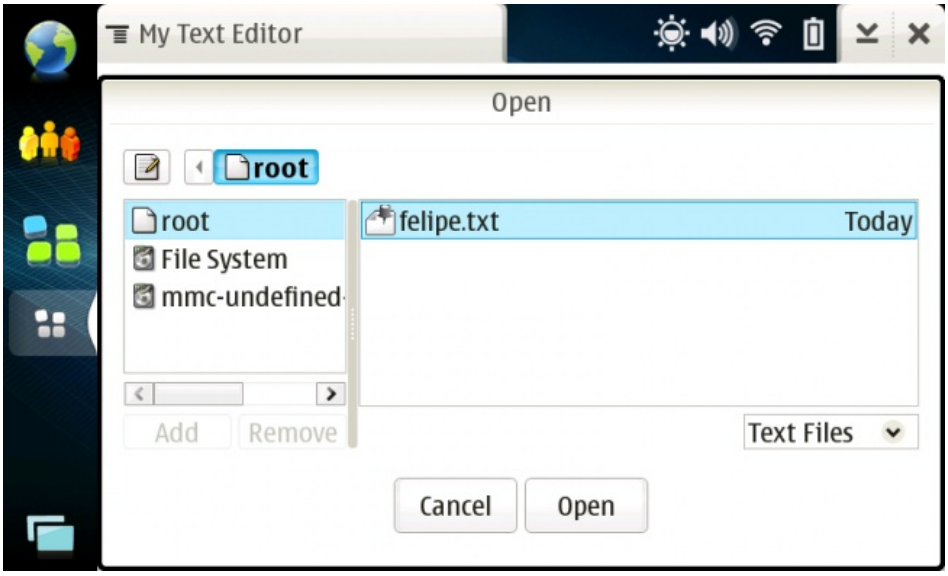
Create the methods `open`, `new`, `save` and `saveAs` at the `TextEditor` class. The close method came from `QMainWindow`.

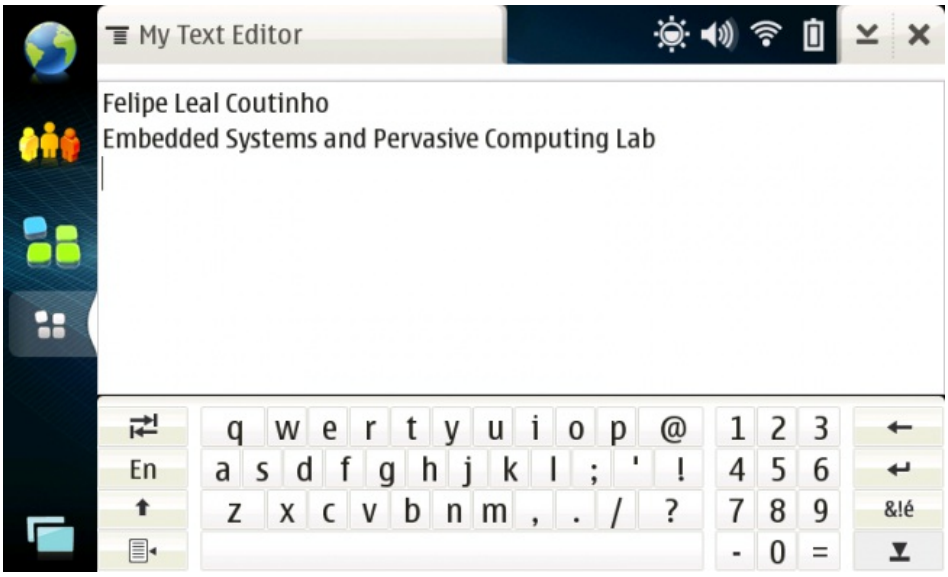
```
    def newFile(self):
        self.__textEdit.clear()
        self.__fileName = ""

    def open(self):
        self.__textEdit.clear()
        self.__fileName = QtGui.QFileDialog.getOpenFileName(self,
                        "Open", ".", "Text Files (*.txt)")

        if self.__userSelectedAFile():
            try:
                fsock = open(self.__fileName, "r")
                self.__textEdit.setPlainText(fsock.read())
                fsock.close()
            except Exception, e:
                QtGui.QMessageBox.critical(self, "Open error", e.message)


    def save(self):
        if not self.__userSelectedAFile():
            self.__openSaveAsDialog()

        if self.__userSelectedAFile():
            try:
                fsock = open(self.__fileName, "w")
                fsock.write(self.__textEdit.document().toPlainText())
                fsock.close()
            except Exception, e:
                QtGui.QMessageBox.critical(self, "Save error", e.message)

    def saveAs(self):
        self.__openSaveAsDialog()

        if self.__userSelectedAFile():
            self.save()

    def __openSaveAsDialog(self):
        self.__fileName = QtGui.QFileDialog.getSaveFileName(self,
                        "Save As", ".", "Text Files (*.txt)")
        if self.__userSelectedAFile() and not self.__fileName.endsWith(".txt", QtCore.Qt.CaseInsensitive):
            self.__fileName = self.__fileName + ".txt"

    def __userSelectedAFile(self):
        return self.__fileName != ""
```

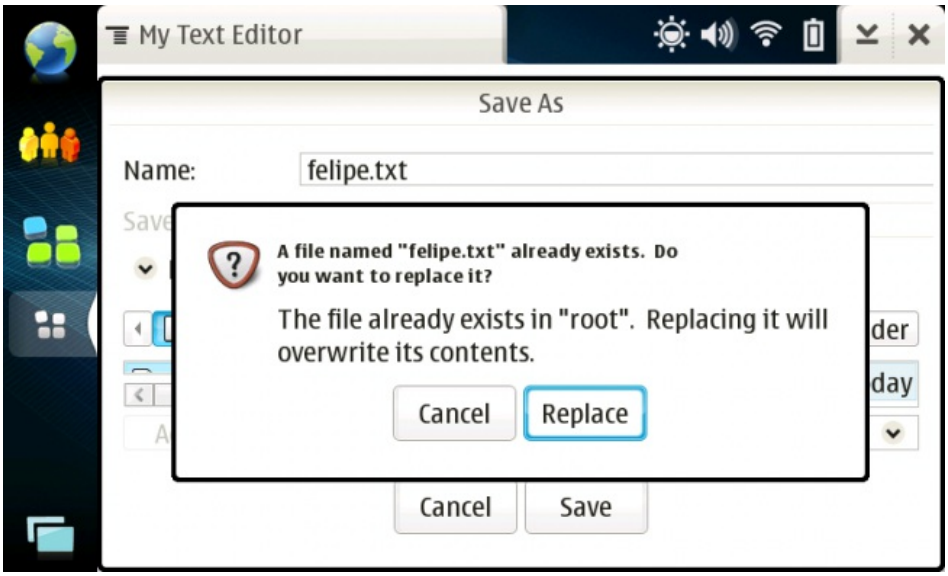
Take a look at the methods `QFileDialog.getOpenFileName` and `QFileDialog.getSaveFileName`. They are very useful because they return the name of the file that the user selected and we can add filters of files.

So, our text editor is complete. Run the program on the N800 and take a look at all the functionalities that it has.

(C) Copyright Nokia 2013. All rights reserved.







If you want to see a bigger project in development that uses PyQt for Maemo, visit https://garage.maemo.org/projects/pyfinancial/.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.