Mobile application development with QML & PySide

PySide
Python for Qt

Martin Kolman, Faculty of Informatics, Masaryk University

slides: http://www.modrana.org/om2012
example program: https://github.com/M4rtinK/expyside

What is PySide

- a project that provides Python bindings for Qt
  - basically a LGPL alternative to the older PyQt project
  - PySide recently became part of the Qt Project
- officially available for Fremantle (N900) and Harmattan (N9)
  - there is an unofficial port for Android
  - and of course it also works on desktop :)

Advantages

- Python is easy to use :)
- no need to (cross-)compile
- code can be easily tweaked on the go
- in combination with Rsync makes for very rapid *change-test* cycles
- big standard library and boatloads of third-party modules

Disadvantages

- bindings don't cover all available libraries
- no Qt Creator support
- no Qt 5 support yet

Setting up the environment

- on a PC
  - install PySide :)
  - install Qt Components from the Forum Nokia PPA
- on a mobile device
  - N900, N950 and N9 are supported by PySide *out of the box*
  - just install the *python-pyside* metapackage and you are ready to go :)
  - on the N900 you might need the *qt-components-10* package

Basic application harness
Python code

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

# A simple PySide example

import sys
import os
from PySide.QtGui import *
from PySide.QtDeclarative import *

WINDOW_TITLE = "PySide Example"

# enable running this program from absolute path
os.chdir(os.path.dirname(os.path.abspath(__file__)))

if __name__ == '__main__':
    app = QApplication(sys.argv) # create the application
    view = QDeclarativeView() # create the declarative view
    view.setSource("main.qml")
    view.setWindowTitle(WINDOW_TITLE)
    view.resize(854,480)
    view.show()
    app.exec_()
```

Basic application harness

QML code

```
import QtQuick 1.1

Rectangle {
    anchors.fill : parent
    Text {
        text: "Hello World"
        anchors.centerIn: parent
    }
}
```

Exporting Python properties to QML

- to export python functions to QML:

1. create a class that instantiates QObject
2. add functions you want to export to this class
3. annotate them
4. instantiate the class an set it as a context property of the declarative view

- the property name ins exported to the *global* QML namespace, so watch out for collisions

Exporting Python properties to QML

Python property code

```python
class PropertyExample(QObject):
    def __init__(self):
        QObject.__init__(self)
        self.rootObject = None
        #NOTE: the root object is needed only by Python properties
        # that call QML code directly

    @QtCore.Slot(result=str)
    def getDate(self):
        """
        return current date & time
        """
        return str(datetime.datetime.now())

    @QtCore.Slot(str)
    def notify(self, text):
        """
        trigger a notification using the
        Qt Quick Components InfoBanner
        """

        #NOTE: QML uses <br> instead of \n for linebreaks
        self.rootObject.notify(text)
```

Exporting Python properties to QML

property export code

```
# add the example property
property = PropertyExample()
rc.setContextProperty("example", property)
```

Exporting Python properties to QML

QML code

```
Text {
    text: example.getDate()
    anchors.horizontalCenter: parent.horizontalCenter
}

Button {
    anchors.horizontalCenter: parent.horizontalCenter
    width : 100
    id : startButton
    text : "notification"
    onClicked : {
        example.notify("entry filed content:<br>" + entryField.text)
    }
}
```

Manipulating QML from Python

- instantiated QML Elements can be directly manipulated from Python
- the easiest way is probably through the root object
  - the root object is created from the file that was set as the declarative view source at startup, in our example this is the *main.qml* file
- but be careful – this ties Python very closely to the (usually ever-changing) QML code

Manipulating QML from Python
Python code

```
def notify(self, text):
    """
    trigger a notification using the
    Qt Quick Components InfoBanner
    """
    rootObject = view.rootObject()
    rootObject.notify(text)
```

Manipulating QML from Python

QML code

```
InfoBanner {
    id: notification
    timerShowTime : 5000
    height : rootWindow.height/5.0
}


function notify(text) {
    notification.text = text;
    notification.show()
}
```

Notifications

- notifications can be easily implemented using the QML **InfoBanner** element
- the **InfoBanner** element is instantiated in the *main.qml* file
- there is also a **notify(text)** function
- this function can be called both from QML and from Python code

  EX: handling more notifications at once

Loading images

- QML supports loading images from files or network
- but what if we want to load an image from raw data in memory or do custom image processing ?
- QDeclarativeImageProvider
  - provides an interface for loading images to QML
  - returns QImage or QPixmap
  - does not update the *Image.progress* property
- reloading an might be a bit problematic due to how image caching works

Loading images
image provider example

```
class ImagesFromPython(QDeclarativeImageProvider):
    def __init__(self):
        # this image provider supports QImage,
        # as specified by the ImageType
        QdeclarativeImageProvider.__init__(self,
QdeclarativeImageProvider.ImageType.Image)

    def requestImage(self, pathId, size, requestedSize):
        # we draw the text provided from QML on the image
        text = pathId
        # for an example image, PySide logo in SVG is used
        image = QImage("pyside.svg")
        image.scaled(requestedSize.width(),requestedSize.height())
        painter = QtGui.QPainter(image)
        painter.setPen("white")
        painter.drawText(20, 20, text)
        return image
```

Loading images

registering the image provider

```
provider = ImagesFromPython()
view.engine().addImageProvider("from_python", provider)

# NOTE: view.engine().addImageProvider("from_python",
# ImagesFromPython())
# doesn't work for some reason
```

Loading images
using the image provider from QML

```
Image {
    anchors.horizontalCenter: parent.horizontalCenter
    width : 200
    height : 200
    smooth : true
    // NOTE: the image provider name in the Image.source
    // URL is automatically lower-cased !!
    source : "image://from_python/" + entryField.text
}
```

Persistent configuration

- can be easily achieved on the Python side
- just export a property with properly annotated get/set methods
- on the Python side, it can be as simple as dictionary that is loaded from file with Marshal on startup and saved back on shutdown
- or other “backends” like configparser, configObj, csv, sqlite, etc. can be used

Simple rapid prototyping

- Python has a big advantage - you don't have to compile the source code
- the same source can be used to run an application both on your desktop computer or your mobile device
- this can be used for a very rapid on-device testing
- **develop anywhere** !
  - the only thing you need is IP connectivity between your desktop/laptop and your mobile device
  - basically any wireless AP will do
  - also works with the built-in mobile hotspot ! :)

Simple rapid prototyping

- requirements
  - *rsync* on your mobile device
  - *scp* might be used as a less-effective alternative
  - SSH-PKY authentication (so that you don't have to enter the password on every sync)
  - the IP address of your computer and your mobile device

The rsync script
app_rsync.sh

```
#!/bin/bash

IP=$1

# NOTE: this deletes any on-device changes to the
# application source files on every sync
# also, the .git folder is not synced (if present)

rsync -avzsh --delete --progress -e 'ssh' my_username@$
{IP}:/home/my_username/coding/app /home/user/coding
--exclude '.git'
```

The startup script
app_start.sh

```
#!/bin/bash

cd software/coding/app
python main.py
```

The sync & run script
run_app.sh

```
#!/bin/sh

# optional automatic IP address detection
#source_ip=`sh get_source.sh`

# place dependent IP addresses
source_ip=192.168.1.2
#source_ip=192.168.0.3
#source_ip=192.168.1.4
#source_ip=192.168.1.5

sh app_rsync.sh $source_ip
#sh temp_rsync.sh $source_ip
sh app_start.sh
```

Installation & usage

- installation
  - place the scripts to a convenient folder on your mobile device
- usage
  - log-in to your mobile device
  - set your PC IP in the main script (optional)
  - run the the scripts as appropriate

Why 3 scripts ?

- better readability
- flexibility – the individual scripts can be used separately:
  - sync & start the application
  - just sync
  - just start the application

Packaging

- is not really needed during development
  - unless you are developing for Harmattan and need Aegis tokens
- programs using PySide can be accepted to the Nokia store (formerly *Ovi Store*)
  - Python applications already in the store:
    - Mieru
    - GPodder
    - RePho (?)
    - and others

How to create packages for Harmattan & Fremantle

- with PySide assistant
  - **http://wiki.meego.com/Python/pyside-assistant**
- with Khertan's sdist_maemo module
  - **http://www.khertan.net/softwares/Sdist_Maemo/**
- with my packaging script that uses modified sdist_maemo and OBS to create Nokia Store-compatible packages
  - **http://www.modrana.org/misc/mieru_build_example.zip**
- using merlin1991's bdist_hdeb module
  - **http://forum.meego.com/showthread.php?t=5523**

PySide applications

- Mieru, RePho, modRana
  - https://github.com/M4rtinK
- Gpodder
  - https://github.com/gpodder
- gotoVienna
  - https://github.com/kelvan/gotoVienna
- AGTL
  - https://github.com/webhamster/advancedcaching

Thank you !

Questions ? :)

Want to contact me ? :)
Martin Kolman
email**: martin.kolman@gmail.com**
jabber: **m4rtink@jabbim.cz**
github**: https://github.com/M4rtinK**