Porting WRT widgets to QML

This article introduces the basic information and steps needed to port a Symbian Web Runtime widget to Qt Quick.

Introduction

WRT widgets are built using classical Web technologies (HTML, CSS, JavaScript) and using the functionalities provided by the WRT runtime environment and by the Platform Services APIs. On the other side, QML is a declarative language allowing to build User Interfaces and even complete applications with a JavaScript based syntax.

Why should you port a WRT widget to QML?

QML offers multiple benefits over classical WRT widgets, including:

- **improved performances**: QML applications run faster than WRT widgets, meaning smoother animations and improved user experience
- **cool UI transitions**: QML offers inbuilt support for creating great-looking UI effects and transitions, without the need to write any JavaScript code
- **Qt extensibility**: if QML doesn't offer all the functionality needed in an application, it can be easily extended by writing the needed functionalities in Qt, and accessing those by the QML layer
- **Cross platform**: being part of Qt, QML allows to create UIs and applications that can be deployed on multiple operating systems, including Symbian, MeeGo, Windows, Linux, OSX

Where to start?

A first introduction to the differences between the Web and QML environments is available in the article: Introduction to QML for Web developers. The topics covered by that article include:

- Standard and custom UI elements
- Positioning of UI elements
- UI manipulation
- UI styling
- JavaScript

Knowledge of the topics covered in that article is needed to fully understand the differences and the techniques to successfully port an application from the WRT world to QML.

This article will focus on the additional steps required to port WRT-specific functionalities to the QML environment.

Focus management

The WRT environment supports three different kinds of focus management, settable by a WRT widget via the widget setNavigationType method:

- cursor based: based on the mouse cursor, with the cursor moved by the user using the navigation pad
- tabbed based: no cursor is shown, the user moves the focus using the navigation pad
- JavaScript based: all focus management must be handled and implemented by the widget in JavaScript

On the other side, QML offers a unique focus management mechanism, based on the FocusScope element.

Saving local data

WRT offers two methods to locally save and load textual data: the widget setPreferenceForKey and preferenceForKey methods.

QML embeds a more powerful and complete Offline Storage API, that allows to use SQLite databases to access local offline storage. The API allows to write and read data from local databases with plain JavaScript: a SQL Local Storage Example is available to see the API in action.

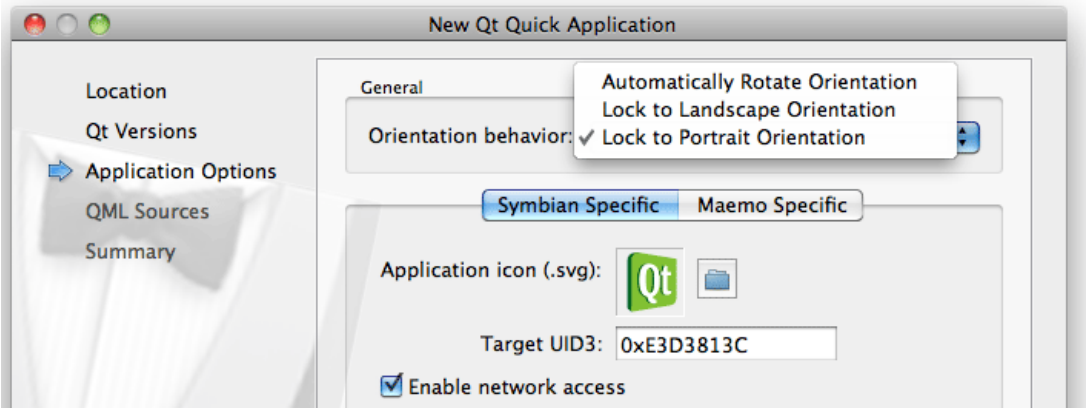
Managing the device orientation

WRT allows to programmatically force the display orientation with the setDisplayLandscape and setDisplayPortrait methods of the widget object.

Qt Quick does not currently allow to lock the device orientation directly via QML, so a line of Qt code is needed. Specifically, it is necessary to call the setOrientation method of the QmlApplicationViewer instance:

```
QmlApplicationViewer viewer;
viewer.setOrientation(QmlApplicationViewer::ScreenOrientationLockPortrait);
viewer.setMainQmlFile(QLatin1String("myQmlFile.qml"));
```

The Qt Creator offers an easy way to perform this operation: when creating a new Qt Quick Application, just select the desired orientation in the project wizard, as shown by the picture below.



The Softkeys

WRT has an inbuilt support for adding and managing the menu items associated to the device softkeys.

QML does not provide any JavaScript APIs to manage softkey options, so Qt code is necessary in this case. A complete example is available on the Qt Reference pages: Qt Soft Keys Example. For more information, check out the Softkeys section on the QWidget Class Reference page.

Audio and Video

In WRT widgets, audio and video playback is possible mainly through the usage of embedded Flash Lite movies, as in the case of the On demand Web TV – have your favorite channels with you.

QML does not offer an inbuilt support for audio and video, but the Multimedia QML plugin from Qt Mobility 1.1 brings this functionality to the QML enviroment. Thanks to this plugin, audio and video can be used in QML applications with plain declarative syntax.

Platform Services

WRT allows to access native OS functionalities through a set of JavaScript APIs, named Platform Services.

While QML has no inbuilt APIs to access low-level functionalities, there are solutions that include:

- Qt Mobility QML plugins: through a set of plugins, the Qt Mobility project adds new functionalities to the QML language.
- Extending QML with C++ code: more information is available here

The Qt Mobility QML Plugins

This sections shows how functionalities based on Platform Services can be ported to QML applications with the help of Qt Mobility QML plugins.

Platform Services API	QML API	Notes
AppManager API	-	-
Calendar API	Organizer plugin from Qt Mobility 1.2	
Contacts API	Contacts plugin from Qt Mobility 1.2	
LandMarks API	Location plugin from Qt Mobility 1.2	
Location API	Location plugin from Qt Mobility 1.2	
Logging API	-	-
Media Management API	Gallery plugin from Qt Mobility 1.1	
Messaging API	Messaging plugin from Qt Mobility 1.1	
Sensors API	Sensors plugin from Qt Mobility 1.2	
System Information API	System Information API from Qt Mobility 1.2	Check out the available [QML Classes