The road to Qt 5

Version 0.6, Lars Knoll, 8. May 2011

Overview

Qt 4.0 was released in 2005. After 6 years it is now time to think about a renewal and how we need to adjust Qt to the huge changes in the market that have happened over the last few years.

UI technology has undergone massive changes over the last three years, especially with the introduction of fully touch based user interfaces. Direct manipulation of objects on the screen is now the preferred way of interaction and require a user experience that is always smooth and uses advanced animations. UX design becomes much more important as many of these user interfaces are now designed to the use case and less to a uniform look and feel of a certain platform.

Application development and many of the innovations do nowadays happen mostly on devices (especially phones and tablets) and a lot less on traditional desktop computers. But many of the UX paradigms introduced on mobile devices now make their way back to the desktop. This implies that traditional, mainly static widgets are loosing their predominant role also on the desktop platforms. The introduction of Qt Quick (QML) has done a lot to have a good offering for creating up to date user experiences.

However, Qt 4.7 does contain a some legacy that will in the long term slow down the progress of the Qt platform. While many parts are still very valuable to our developer base some are also blocking our way forward in the current Qt 4.x frame.

Thus a renewal with Qt 5 is needed. Qt 5 will give us the freedom to do certain changes and break compatibility with the past where required for the future. We will however need to take care to bring our existing developer base along with us and not break any fundamentals as experienced when moving from Qt 3 to Qt 4. To be explicit here, this implies that while we will break binary compatibility, we will do our utmost to keep source compatibility wherever possible. As such most applications should not require much more than a recompile to move from Qt4 to Qt5.

The major change with Qt 5 will be in parts of our graphics stack and the programming model advocated to applications.

Qt 5 will initially focus on a smaller set of operating systems/platforms. The core platforms will be Linux on Wayland and X11, Mac and Windows. Some other operating systems that are currently supported by Qt 4 (esp. commercial Unix systems) will not receive any direct attention from Nokia, but can be supported by 3rd parties. This implies that while we keep the core of Qt available on all supported platforms, Qt will begin to offer differentiated functionality on some OS'es.

Another major change with Qt 5 will be in the development model. Qt 4 was mainly developed in house in Trolltech and Nokia and the results where published to the developer community.

Qt 5 needs to be developed in the open and as an open source project from the very start.

Vision

Qt 5.0 will be the foundation for a new way of developing applications. While offering all of the power of native Qt/C++, the focus will over time shift to a QML centric model, where C++ is mainly used to implement modular backend functionality for QML.

This implies that we don't want Qt 5 to be disruptive for existing code developed for Qt 4. It will rather be a restructuring that will allow us to change the way we will think about application development in the future.

Breaking binary compatibility will allow us to restructure our code base according to these needs and put QML into the center of Qt.

Programming model

While we will keep traditional Qt/C++ applications running on the desktop, Qt 5 will bring some fundamental changes in how applications are being written.

In Qt 5 the entry point for applications can be QML instead of C++. We expect that all UI will be written in QML. JavaScript will become a first class citizen and we expect that a lot of application logic will be written in JS instead of C++.

The expectation is that many application developers will actually start out with QML and JavaScript and only implement functionality in C++ when required. For that use case, the full power of the C++ APIs offered by Qt can be used to implement time critical and complex application functionality.

For this use case, C++ based functionality will mainly be exposed to applications as QML modules. The existing concepts in QObject make this a straightforward task, and we need to make the creation of QML modules as simple and straightforward as possible.

This will later on allow us to expose lots of application functionality as QML modules allowing for nice mash-ups of different applications by simply importing QML modules from different applications. This will become an extremely flexible and powerful replacement for the plugin and shared library functionality offered in C++ based framework.

While we keep the QWidget based programming model and API set available for compatibility, we also see QML as the future for user interfaces on the desktop. The solution here will most likely be QML based component sets that integrate with the native styling APIs on the desktop platforms.

Development model

A very important aspect of Qt 5 is the development model we will use. Qt 5 will be developed in the open and as an open source project from day 1. The modularized set of repositories that will contain the source code for Qt 5 will be hosted outside of Nokia and all communication about Qt 5 development will happen on open mailing lists and IRC channels.

All feature discussions have to take place in the open. Different groups can contribute their favorite features to the project. Development of a framework will naturally require strong steering and strict rules on quality of the code as well as a match with the goals of Qt.

We will setup infrastructure to help achieving this goal. A lot of work has been done here as part of the 'open governance' project and we will need to now deploy what we have so far and simply get started.

Changes to the Qt architecture

Since we don't want to change too many of Qt's fundamentals, and the fact that we want to make it easy for existing applications to move to Qt 5 require us to be careful with the amount of changes we do to our existing code base. Therefore we believe we will be able to do the change in a timely manner. We would like to have a stable (beta quality) Qt 5 code base available by the end of 2011.

Many of the changes we are going to do are about restructuring our code base into a new modular structure where every shared library lives in its own repository. We will remove a few very rarely used APIs where a compatibility requirement makes some required changes impossible. As the last big change we will re-architecture most of our graphics stack to be fully optimized for QML, allowing for smooth touch based user interfaces.

As Qt 5 will have a broader focus than Qt 4 which was mainly targeted at the desktop operating systems, we will separate out many of the components only required for the desktop into it's own library.

Embedded devices are traditionally less powerful than desktops and to create an offering that can compete in the market we will not make compromises on performance.

Window system integration

One major change in Qt 5 will be how we integrate into different windowing systems. The lighthouse abstraction layer has proven to be a better way to abstract windowing systems then our old approach using _foo.cpp files. The fact that we are nowadays not using native child windows anymore also makes this approach a lot more feasible. So with Qt 5, lighthouse will become our default way to integrate with different windowing systems.

Our primary platforms for development and testing of Qt will be Linux with the Wayland windowing system, X11, Mac OS X and Microsoft Windows. Other OS's will get integrated if we find a team that will support the work.

Graphics stack

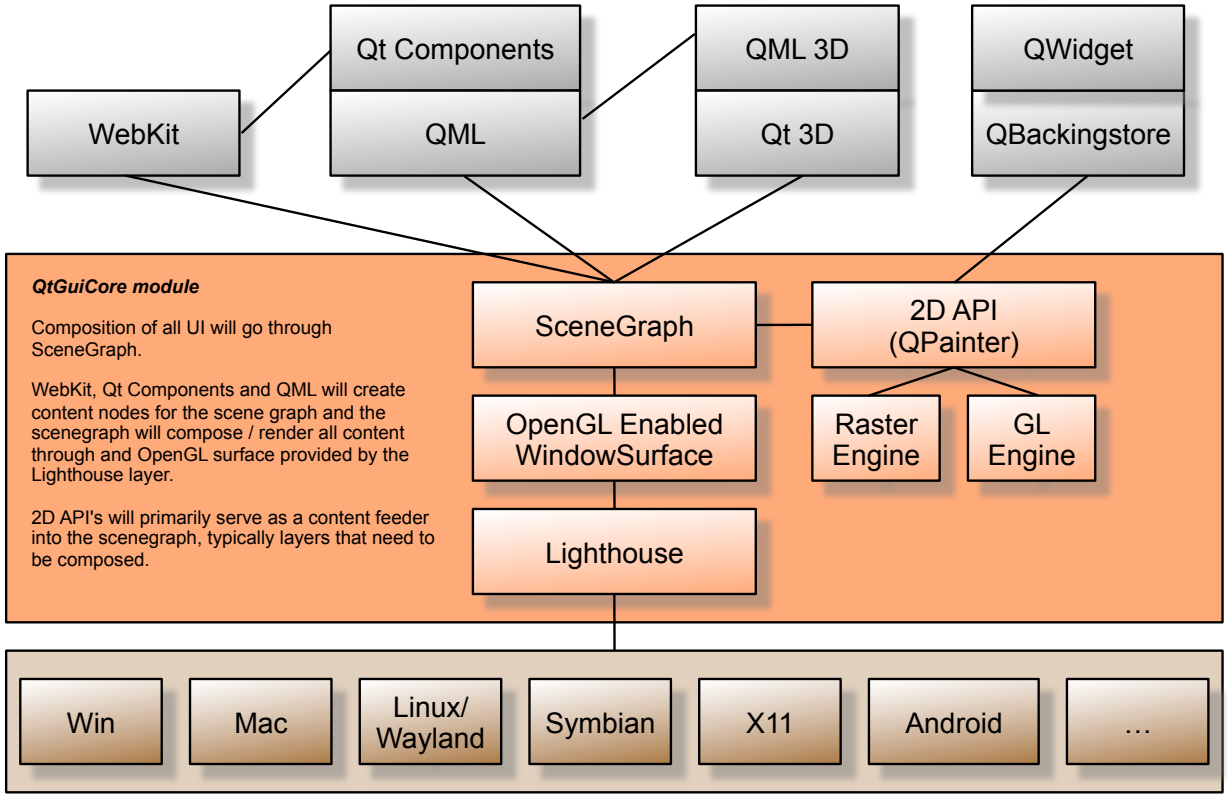
Qt 5 will be optimized for displaying user interfaces using QML. Classic QWidget based applications will get integrated on top of the QML optimized stack.

For all our work, we assume OpenGL ES 2.0 as a minimum requirement. On very low end systems, a SW implementation such as Mesa could fill the gap, while we can use ANGLE (a translation layer from OpenGL to DirectX) on Windows if required.

Qt 5 will allocate top-level windows/surfaces through the lighthouse window system integration. A GL based scenegraph that is especially optimized for QML is being used to place objects into the scenegraph. All drawing and animations in the scene graph will be driven by the vsync signal of the graphics hardware.

QML will use the scenegraph directly as the basis for rendering and animating graphical objects on the screen. The scenegraph will give us a very direct mapping from QML to OpenGL objects and a best in class graphics performance. It will also allow things such as GL based shader effects to be directly implemented with a few lines of QML code.

The existing QPainter based drawing API will continue to be supported. Its main use is to draw documents (e.g. WebKit, text documents, etc.) into off-screen surfaces. Drawing for offscreen surfaces will either happen through OpenGL (for FBOs) or the software rasterizer (QImage). QPainter will also still allow to create a few selected other output formats such as PDF for printing.



QML

QML in Qt 4.7 uses QGraphicsView as its rendering backend. All drawing happens through imperative drawing commands making suboptimal use of the GPU and making certain things as shader effects on subtrees of items very hard to implement.

For that reason we will be moving over to the scene graph that has been developed with QML in mind over the last year. This new scene graph based version of QML will be backwards compatible on the QML side, but QML items implemented in C++ (by inheriting QDeclarativeItem) will need to be slightly changed. We will aim at providing an item in the scene graph that is as (source) compatible as possible to QDeclarativeItem to ease transition.

The existing QGraphicsView based QML will continue to be supported for a transitionary period, but will not be developed any further. Furthermore it will depend on QWidget and thus be not recommended (or available) for usage on devices.

QtScript

QtScript is the JavaScript API that Qt offers. It is currently implemented using a forked JavaScriptCore (JSC) library and keeping up with development by the community is very difficult, esp. as Apple (who owns JSC) is not giving any promises on API stability.

Google has started an alternative engine a couple of years ago called V8. The engine has a clean, easy to use and well defined API. In addition it is right now quite a bit faster than JSC.

These arguments have made it clear that we need to move QtScript over to use V8. The added advantage is that we will be able to have only one JS engine in Qt going forward (we currently have two copies of JSC in Qt). This one engine can then be used by both QML and WebKit.

With Qt 5 it is our goal to simplify some of the APIs in QtScript. We will remove support for certain more esoteric APIs, esp those that rely heavily on JSC internals.

WebKit

QtWebKit will also undergo some changes in the API. While we will continue to offer most of the current API set, it has become clear that a “WebKit2” based model where the HTML rendering happens in a separate thread or process is becoming a must for most things.

The implementation of that WebKit2 model is currently being driven along as a research project and has already made very good progress. A process separation will in the longer term allow us to completely sandbox the HTML rendering process. This is the only viable way to reliably close security issues for the browser.

Mobility APIs

The current set of mobility APIs will form the basis for a set of modules that will become part of Qt 5.

Library structure

The library structure will need to get slightly modified compared to what we have in Qt 4 to adjust to changes in the market and the new QML centric approach.

As a first change we should consider merging most of QtNetwork into QtCore. Very few applications nowadays get along without network connectivity and such a merge will allow us to use networking in more places in Qt itself.

The second bigger reorganization is to split all QWidget based functionality form QtGui into a QtWidgets library. The reason here is that the QWidget derived classes are very much

being used by our developers on the desktop, but have little relevance for the QML based user interfaces we'd like to promote going forward. In some respect the QtWidgets library will become the layer that will help our existing users transition over to Qt 5.

The remaining QtGui will contain the low level graphics and text handling functionality as well as support to create top level windows and surfaces. The scene graph will be an integral part of this library and the support for QWidgets will build upon it.

We need to however ensure that the new QtGui library still contains everything that is required to create fully functional QML applications for broad use cases. This includes desktop operating systems as well as other devices.

An open question is where to put the QML engine and the scene graph based QML items. They could either live in the new QtGui or a module of it's own. QtGui however sounds like an attractive option as QML is supposed to form the basis of our new offering.

The old graphics view based QML items should use the same QML engine as the new items, but will continue to live in a QtDeclarative module that links against QtWidgets (for graphics view).

All QWidget functionality in QtWebKit will need to be moved into a small QtWebKitWidgets library to keep QtWebkit independent of QWidget. The same is true for other modules (e.g. QtMultimedia) that implement QWidget based classes.

Parts of our QtOpenGL module will be merged into a new module containing the graphics kernel for QML based applications. The remaining pieces will most likely be merged into QtWidgets.

Deprecated modules

We are intending to deprecate and remove certain modules from our core offering. The Qt3Support module (and related functionality in other places in Qt) will be completely removed.