Archived:Porting Android (Java) applications to Qt

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*}}.

This article is somewhat old, and though mostly valid, inaccurate at places. More recent porting from Android to Qt documentation is available as part of the Porting to Qt Library

Introduction

As an initiative of The Open Handset Alliance and Google, Android has been developed as the first open and free platform for mobile phones. The project has been developed since November 5, 2007, when Google Inc., Intel, T-Mobile, Sprint, HTC, Qualcomm and Motorola decided to achieve an ambitious goal: provide services so consumers would face a far better user experience.

An important point of the Android platform is that handset manufacturers and wireless operators can provide suitable versions of Android for their products and services. This characteristic directly impacts on lower cost and innovative products. Therefore, Android can be considered a new and potential option for mobile platforms for smartphones.

On the other hand, Qt has been considered a powerful component in this newest mobile programming arena. Qt is most notably used in Desktop applications, such as KDE, Opera, Skype and VirtualBox, but recently it has been ported to Nokia mobile platforms, such as S60 and Maemo. Initially developed by Trolltech since 1991, Qt is released with two different licenses (open source and commercial), which should make Qt more suitable for non-GPL open source projects and for commercial users. In June 2008, Nokia acquired Trolltech to enable the acceleration of cross-platform software strategy for mobile devices and desktop applications. On September 29, 2008 Nokia renamed Trolltech to Qt.

This article provides information that guides development to port Android applications to Qt.

Overview of Android Platform

Android applications are developed with the Java programming language (Dalvik virtual machine), and device services, such as touchscreen and storage

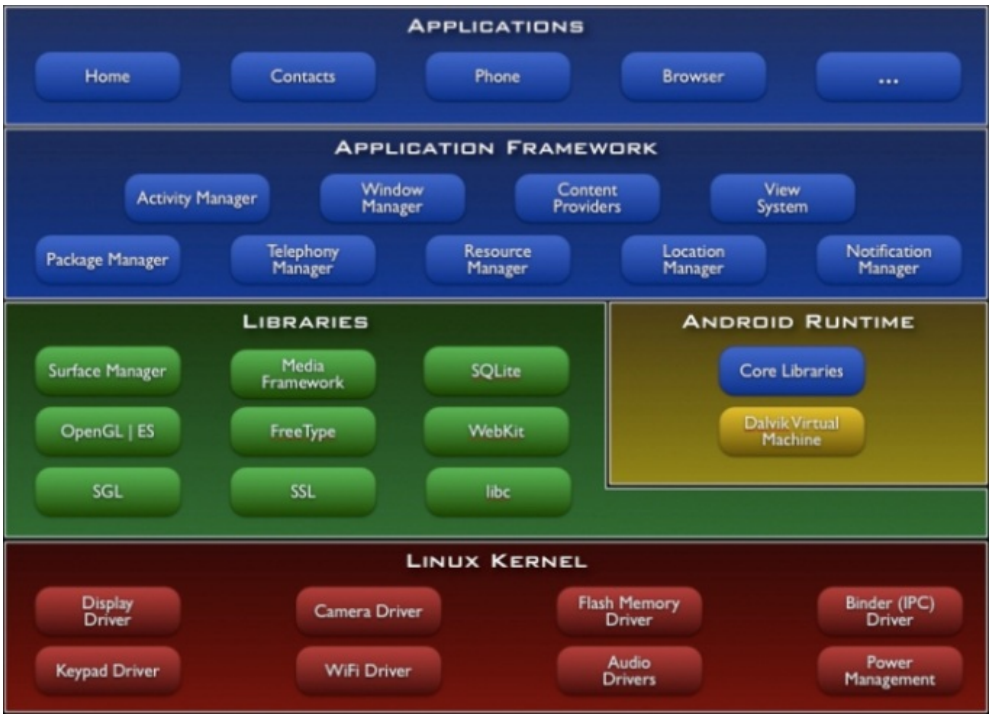
(C) Copyright Nokia 2013. All rights reserved.

features, can be accessed through the Google services API. It is possible to run applications written in C or any other language, but the application has to be compiled to native code and run; this development path is not officially supported by Google.

Since October 2008, Android has been available as an open source project (Apache License). Handset manufacturers and wireless operators are free do add closed and proprietary extensions to their products.

Although Android is based on a Linux kernel, according to Google, it is not a Linux operating system. In addition, it does not have a native windowing system, nor does it support the full set of standard Linux libraries, including the GNU C Library. This characteristic make it difficult to reuse existing Linux applications or libraries. Android does not use standard Java APIs, such as J2SE and J2ME. This prevents compatibility between Java applications written for those platforms and those for the Android platform. Android only reuses the Java language syntax, but does not provide the full-class libraries and APIs bundled with J2SE or J2ME.

The next picture shows the current Android architecture (from the Android Developer's Guide).



The system has access to mobile phone resources through system drivers, such as camera, display, WiFi and keypad drivers. Then, the next layer consists of libraries and the runtime system of Android. Finally, Android provides a set of libraries (Application Framework) so it is possible to extend them and create new applications.

Android makes it possible to reuse components of other applications. For example, if you need to reuse a component that provides a suitable scroller and made it available to others, you can invoke such a component to do the work for you. Therefore, the system has been designed to start an application process when any part of it is needed, and instantiate the Java objects for that part. Therefore, Android does not provide an entry point, such as main function, but essential components: activity, services, broadcast receives and content providers.

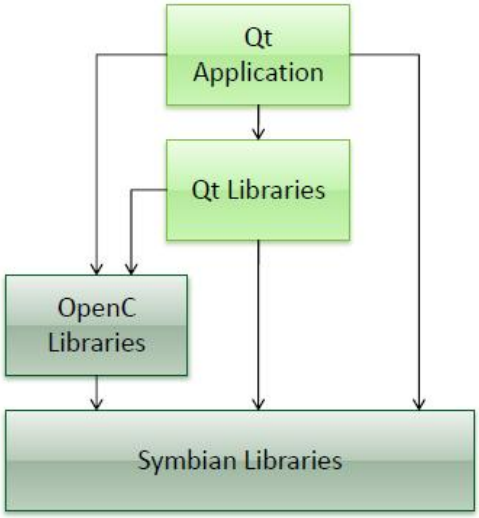
Activities represent screens on an Android application. From an activity, you can display buttons, labels, menus and much more. All activities subclass the *android.app.Activity* class. **Services** do not have visual appearance, but they run in the background. For example, a service may play music while the user performs other tasks. Each service extends the *android.app.Service* base class. **Broadcast receives** are components that receive and react to different broadcast announcements, for example, a message that the battery is low. All receivers extend the *android.content.BroadcastReceive* base class. A **content provider** is responsible for making an application's data available to other applications. Therefore with content providers, it is possible to share data between different applications. All content providers extend the *android.content.ContentProvider* base class.

The Android development environment consists of: the Android SDK, Android source code and, optionally, IDEs that help make programming Android applications a lot quicker. The Android software development kit (SDK) consists of libraries and tools, including an emulator to run applications. The Android SDK is available for Windows, Mac OS X, and Linux. There are different IDEs that provide support for Android development, such as Eclipse (with the Eclipse plug-in for Android).

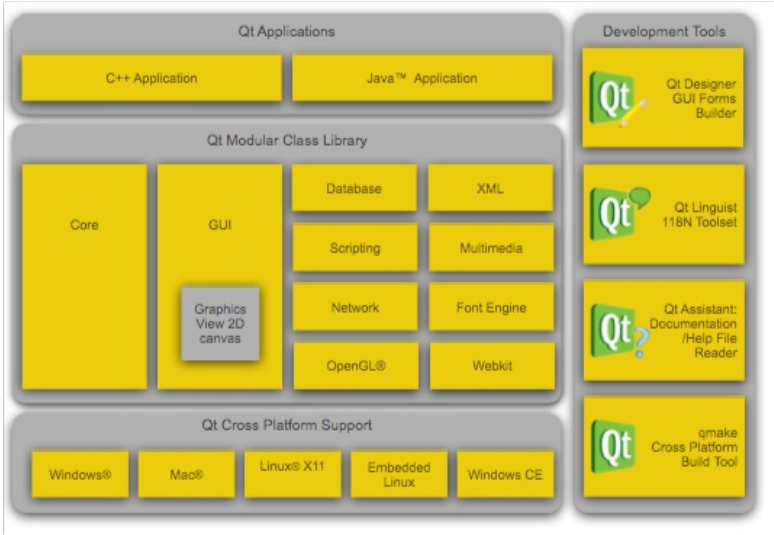
Overview of Qt

Qt applications are developed with C++ enhanced with additional extensions implemented by a pre-processor that generates standard C++ code before compilation. Qt also provides bindings for several other programming languages, like Python, Ruby and Perl.

(C) Copyright Nokia 2013. All rights reserved.



Qt is notably used because of its GUI widgets, but also provides a set of non-GUI related features: SQL database access, XML parsing, thread management, network support and a unified cross-platform API for file handling. Qt has been successfully ported to the Nokia Symbian S60, Maemo, and MeeGo platforms. Qt will "enable you to create advanced applications with innovative user experiences while getting to market quickly". Compared with Android, Qt is an interesting and easy-to-use programming framework that certainly brings a considerable contribution to mobile programming as well as desktop application development.



Qt Mobility has been completely ported to Symbian, and commonly used mobile features are accessible through the Qt Mobility APIs such as camera, geolocation, and contacts APIs. There are rapid on-going progress of making all mobile features accessible, and some features which are not provided by Qt Mobility APIs can still be accessed through native Symbian APIs.

In order to develop Qt Symbian applications, you simply need to install the Nokia Qt SDK, which contains Qt SDK, Qt Mobility SDK, Qt Creator, Qt Simulator for Symbian and N900, and libraries, tools (including the emulator for S60 environment), and all documentation you need to get started.

Qt Creator is an IDE built on Qt itself, and contains an easy-to-use visual designer for designing your application UI. An alternative to Qt Creator is Carbide++ IDE (based on the Eclipse framework). Both Qt Creator and Carbide C++ IDE provide a fully featured environment that helps a lot with development. You can find interesting articles on the Nokia Developer Wiki covering Qt (see Category Qt).

Qt applications are easily built with Qt Creator (included in Nokia Qt SDK) or Carbide IDE and they can be launched on the Qt Simulator, Symbian S60 emulator, or even on your device (you have to install Qt libraries first, however a properly packaged Qt application can do this automatically via Nokia Smart Installer). The entry point of any Qt application is the main method (remember that Android applications start from an Activity instance).

Examples

This is a simple "Hello World" application on Android.

Invalid language.

You need to specify a language like this: `<source lang="html4strict">...</source>`

Supported languages for syntax highlighting:

4cs, 6502acme, 6502kickass, 6502tasm, 68000devpac, abap, actionscript, actionscript3, ada, algol68, apache, applescript, apt_sources, asm, asp, autoconf, autohotkey, autoit, avisynth, awk, bascomavr, bash, basic4gl, bf, bibtex, blitzbasic, bnf, boo, c, c_loadrunner, c_mac, caddcl, cadlisp, cfdg, cfm, chaiscript, cil, clojure, cmake, cobol, coffeescript, cpp, cpp-qt, csharp, css, cuesheet, d, dcs, delphi, diff, div, dos, dot,

(C) Copyright Nokia 2013. All rights reserved.

e, ecmascript, eiffel, email, epc, erlang, euphoria, f1, falcon, fo, fortran, freebasic, fsharp, gambas, gdb, genero, genie, gettext, glsl, gml, gnuplot, go, groovy, gwbasic, haskell, hicest, hq9plus, html4strict, html5, icon, idl, ini, inno, intercal, io, j, java, java5, javascript, jquery, kixtart, klonec, klonecpp, latex, lb, lisp, llvm, locobasic, logtalk, lolcode, lotusformulas, lotusscript, lscript, lsl2, lua, m68k, magiksf, make, mapbasic, matlab, mirc, mmix, modula2, modula3, mpasm, mxml, mysql, newlisp, nsis, oberon2, objc, objeck, ocaml, ocaml-brief, oobas, oracle11, oracle8, oxygene, oz, pascal, pcre, per, perl, perl6, pf, php, php-brief, pic16, pike, pixelbender, pli, plsql, postgresql, povray, powerbuilder, powershell, proftpd, progress, prolog, properties, providex, purebasic, pycon, python, q, qbasic, rails, rebol, reg, robots, rpmspec, rsplus, ruby, sas, scala, scheme, scilab, sdlbasic, smalltalk, smarty, sql, systemverilog, tcl, teraterm, text, thinbasic, tsql, typoscript, unicon, uscript, vala, vb, vbnet, verilog, vhdl, vim, visualfoxpro, visualprolog, whitespace, whois, winbatch, xbasic, xml, xorg_conf, xpp, yaml, z80, zxbasic

```
package helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       TextView tv = new TextView(this);
       tv.setText("Hello World");
       setContentView(tv);
   }
}
```

As explained before, the entry point of any Android application is an Activity (more strictly, the method android.app.Activity#onCreate(Bundle)). There's no label class on Android, but text views can be also used as labels. Then, we create a text view and we set the contents of Activity with the text view that was just created.

Now, let's see the same application, but on the Qt framework.

Invalid language.

You need to specify a language like this: `<source lang="html4strict">...</source>`

Supported languages for syntax highlighting:

4cs, 6502acme, 6502kickass, 6502tasm, 68000devpac, abap, actionscript, actionscript3, ada, algol68, apache, applescript, apt_sources, asm, asp, autoconf, autohotkey, autoit, avisynth, awk, bascomavr, bash, basic4gl, bf, bibtex, blitzbasic, bnf, boo, c, c_loadrunner, c_mac, caddcl, cadlisp, cfdg, cfm, chaiscript, cil, clojure, cmake, cobol, coffeescript, cpp, cpp-qt, csharp, css, cuesheet, d, dcs, delphi, diff, div, dos, dot, e, ecmascript, eiffel, email, epc, erlang, euphoria, f1, falcon, fo, fortran, freebasic, fsharp, gambas, gdb, genero, genie, gettext, glsl, gml, gnuplot, go, groovy, gwbasic, haskell, hicest, hq9plus, html4strict, html5, icon, idl, ini, inno, intercal, io, j, java, java5, javascript, jquery, kixtart, klonec, klonecpp, latex, lb, lisp, llvm, locobasic, logtalk, lolcode, lotusformulas, lotusscript, lscript, lsl2, lua, m68k, magiksf, make, mapbasic, matlab, mirc, mmix, modula2, modula3, mpasm, mxml, mysql, newlisp, nsis, oberon2, objc, objeck, ocaml, ocaml-brief, oobas, oracle11, oracle8, oxygene, oz, pascal, pcre, per, perl, perl6, pf, php, php-brief, pic16, pike, pixelbender, pli, plsql, postgresql, povray, powerbuilder, powershell, proftpd, progress, prolog, properties, providex, purebasic, pycon, python, q, qbasic, rails, rebol, reg, robots, rpmspec, rsplus, ruby, sas, scala, scheme, scilab, sdlbasic, smalltalk, smarty, sql, systemverilog, tcl, teraterm, text, thinbasic, tsql, typoscript, unicon, uscript, vala, vb, vbnet, verilog, vhdl, vim, visualfoxpro, visualprolog, whitespace, whois, winbatch, xbasic, xml, xorg_conf, xpp, yaml, z80, zxbasic

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  QLabel label("Hello, world!");
  label.show();
  return app.exec();
}
```

It simply opens an application and shows a label with the “Hello World” string. You can check that the main Qt class (QApplication) is properly initialized with the given main method arguments. As described before, the entry point for Qt applications is the main function of your Qt main class. Then, it creates a simple label and shows its content. Finally, the application returns the code result from execution of the Qt application. Another important point is that such code can also be built and launched on the Maemo platform (obviously, the graphical result is different, since you're using a different platform with different widgets).

The following example shows how we can insert a menu with Android and Qt applications.

On Android, the menus are defined with the method *android.app.Activity#onCreateOptionsMenu(Menu)*. The callbacks for menu items are defined with the method *android.app.Activity#onOptionsItemSelected(MenuItem)*.

Invalid language.

You need to specify a language like this: `<source lang="html4strict">...</source>`

Supported languages for syntax highlighting:

(C) Copyright Nokia 2013. All rights reserved.

4cs, 6502acme, 6502kickass, 6502tasm, 68000devpac, abap, actionscript, actionscript3, ada, algol68, apache, applescript, apt_sources, asm, asp, autoconf, autohotkey, autoit, avisynth, awk, bascomavr, bash, basic4gl, bf, bibtex, blitzbasic, bnf, boo, c, c_loadrunner, c_mac, caddcl, cadlisp, cfdg, cfm, chaiscript, cil, clojure, cmake, cobol, coffeescript, cpp, cpp-qt, csharp, css, cuesheet, d, dcs, delphi, diff, div, dos, dot, e, ecmascript, eiffel, email, epc, erlang, euphoria, f1, falcon, fo, fortran, freebasic, fsharp, gambas, gdb, genero, genie, gettext, glsl, gml, gnuplot, go, groovy, gwbasic, haskell, hicest, hq9plus, html4strict, html5, icon, idl, ini, inno, intercal, io, j, java, java5, javascript, jquery, kixtart, klonec, klonecpp, latex, lb, lisp, llvm, locobasic, logtalk, lolcode, lotusformulas, lotusscript, lscript, lsl2, lua, m68k, magiksf, make, mapbasic, matlab, mirc, mmix, modula2, modula3, mpasm, mxml, mysql, newlisp, nsis, oberon2, objc, objeck, ocaml, ocaml-brief, oobas, oracle11, oracle8, oxygene, oz, pascal, pcre, per, perl, perl6, pf, php, php-brief, pic16, pike, pixelbender, pli, plsql, postgresql, povray, powerbuilder, powershell, proftpd, progress, prolog, properties, providex, purebasic, pycon, python, q, qbasic, rails, rebol, reg, robots, rpmspec, rsplus, ruby, sas, scala, scheme, scilab, sdlbasic, smalltalk, smarty, sql, systemverilog, tcl, teraterm, text, thinbasic, tsql, typoscript, unicon, uscript, vala, vb, vbnet, verilog, vhdl, vim, visualfoxpro, visualprolog, whitespace, whois, winbatch, xbasic, xml, xorg_conf, xpp, yaml, z80, zxbasic

```
package helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Menu;
import android.widget.MenuItem;

public class HelloAndroid extends Activity {

   private final int MENU_QUIT = 1;

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       TextView tv = new TextView(this);
       tv.setText("Hello World");
       setContentView(tv);
   }

   /* Creates the menu items */
   public boolean onCreateOptionsMenu(Menu menu) {
       menu.add(0, MENU_QUIT, 0, "Quit");
       return true;
   }

   /* Handles item selections */
   public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case MENU_QUIT:
        quit();
        return true;
    }
    return false;
   }

}
```

On Qt, it is a little bit easier :). You just need to define a QAction. Then, insert it onto the application menu bar. Finally, the callback is defined with the method *connect* (used to define any callback on Qt).

Invalid language.

You need to specify a language like this: `<source lang="html4strict">...</source>`

Supported languages for syntax highlighting:

4cs, 6502acme, 6502kickass, 6502tasm, 68000devpac, abap, actionscript, actionscript3, ada, algol68, apache, applescript, apt_sources, asm, asp, autoconf, autohotkey, autoit, avisynth, awk, bascomavr, bash, basic4gl, bf, bibtex, blitzbasic, bnf, boo, c, c_loadrunner, c_mac, caddcl, cadlisp, cfdg, cfm, chaiscript, cil, clojure, cmake, cobol, coffeescript, cpp, cpp-qt, csharp, css, cuesheet, d, dcs, delphi, diff, div, dos, dot, e, ecmascript, eiffel, email, epc, erlang, euphoria, f1, falcon, fo, fortran, freebasic, fsharp, gambas, gdb, genero, genie, gettext, glsl, gml, gnuplot, go, groovy, gwbasic, haskell, hicest, hq9plus, html4strict, html5, icon, idl, ini, inno, intercal, io, j, java, java5, javascript, jquery, kixtart, klonec, klonecpp, latex, lb, lisp, llvm, locobasic, logtalk, lolcode, lotusformulas, lotusscript, lscript, lsl2, lua, m68k, magiksf, make, mapbasic, matlab, mirc, mmix, modula2, modula3, mpasm, mxml, mysql, newlisp, nsis, oberon2, objc, objeck, ocaml, ocaml-brief, oobas, oracle11, oracle8, oxygene, oz, pascal, pcre, per, perl, perl6, pf, php, php-brief, pic16, pike, pixelbender, pli, plsql, postgresql, povray, powerbuilder, powershell, proftpd, progress, prolog, properties, providex, purebasic, pycon, python, q, qbasic, rails, rebol, reg, robots, rpmspec, rsplus, ruby, sas, scala, scheme, scilab, sdlbasic, smalltalk, smarty, sql, systemverilog, tcl, teraterm, text, thinbasic, tsql, typoscript, unicon, uscript, vala, vb, vbnet, verilog, vhdl, vim, visualfoxpro, visualprolog, whitespace, whois, winbatch, xbasic, xml, xorg_conf, xpp, yaml, z80, zxbasic

```
#include <QApplication>
#include <QAction>

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  QAction exitAction = new QAction(tr("&Exit"),this);

  // Add action direct to menubar
  menuBar()->addAction(exitAction);
  connect(exitAction, SIGNAL(triggered()),this, SLOT(close()));
}
```

API Mappings

(C) Copyright Nokia 2013. All rights reserved.

File I/O

- To read from or write to files or other devices.
  - Android classes (from package **java.io**): File, FileReader, FileWriter, BufferedReader, BufferedWriter
  - Qt classes: QIODevice, QFile, QTemporaryFile, QBuffer, QProcess, QTcpSocket, QUdpSocket, QDataStream, QTextStream

Networking

- Socket communication.
  - Android classes (from package **java.net**): Socket, InetSocketAddress, ServerSocket
  - Qt classes: QTcpSocket, QUdpSocket
- HTTP/FTP communication.
  - Android classes (from package **java.net**): HttpURLConnection, URL
  - Qt classes: QNetworkAccessManager, QUrl, QUrlInfo
  - Android classes (from package **org.apache.http.***): HttpClient, HttpGet, HttpPost, HttpResponse
  - Qt classes: QHttpHeader, QHttpRequestHeader, QHttpResponseHeader
  - Qt class: QFtp (no equivalent on Android)

Media

- To read from or write to files or other devices.
  - Android classes (from package **android.media**): MediaPlayer, MediaRecorder
  - Qt classes: AudioOutput, MediaController, MediaNode, MediaObject, MediaSource, ObjectDescription, Path, VideoPlayer

Android vs Qt

Android was designed as a platform for mobile phones. Therefore, it provides access to different system resources, such as touch screen, camera, and telephony. On the other hand, Qt is a cross-platform application and UI framework designed for Desktop **and** Mobile environments. Qt provides mechanism to access resources of the mobile phone through Qt Mobility APIs. The following list shows some important differences of Android and Qt.

- Android provides access to **PIM (Contacts, Calendar)**, Qt Mobility provides the Contacts API and Organizer API and Versit API;
- Android provides access to **Telephony**, Qt Mobility provides Bearer Management API;
- Android provides access to **Messaging**, Qt Mobility provides the Messaging API;
- Android provides access to **Camera**, Qt Mobility provides Multimedia API;
- On Android, it is possible to load an application UI from an XML description file (Maemo also support this feature if you use Glade to build your application's UI). Qt provides two ways to design your application UI: Form Designer in Qt Creator for designing application user interfaces, and QML of Qt Quick for fluid interaction with animations.
- Android is based on Java and Qt is based on C++;
- Android applications are restricted to the Android platform, while Qt is cross-platform. A properly designed Qt application can be executed on Symbian, Maemo, and MeeGo mobile devices, and also on Windows, Linux, and Mac OS desktop platforms.

References

- Qt
  - Qt.nokia.com
  - Qt reference documentation
  - Foundations of Qt® Development (Book)
  - C++ GUI Programming with Qt 4 (Book)
  - The Book of Qt 4: The Art of Building Qt Applications (Book)
- Android
  - Wiki
  - Official Site
  - Developers Guide
  - Professional Android Application Development (Book)

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.