A Brief Introduction to Qt

Bruce Bolden

March 25, 2009

What is Qt[1]?

A platform independent (cross-platform) graphics library for the development of applications with/without UIs (user interfaces).

Features

- Intuitive C++ class library
- Portability across desktop and embedded operating systems
- *Look and Feel* of the native OS
- Integrated development tools with cross-platform IDE
- High runtime performance and small footprint on embedded systems

Note: Qt is pronounced *cute* by Europeans.

[1]Qt is pronounced *cute* by Europeans.

Some frequently Used Acronyms

API	Application Program Interface
GUI	Graphical User Interface
IDE	Integrated Development Environment
KDE	K Desktop Environment
LGPL	GNU Lesser General Public License
RTTI	Run Time Type Identification
SDK	Software Development Toolkit

KDE details: `http://www.kde.org/`

Why Qt?

- Not Java
- Platform independent (cross-platform)
- Large C++-based library
- The choice for KDE development
- Easy transition to OpenGL

Qt History

- Trolltech was founded in 1994
- Nokia acquired Trolltech ASA, in June 2008
- Active development!
  - 4.5 released March 3, 2009
  - 4.2 released August 24, 2006

This is Qt 4.5: `http://www.youtube.com/watch?v=8xRfNsY53GY`

Qt History—Details

Nokia acquired Trolltech ASA, in June 2008, to enable the acceleration of their cross-platform software strategy for mobile devices and desktop applications, and to develop its Internet services business. On September 29, 2008 Nokia renamed Trolltech to Qt Software.[2]

Trolltech was founded in 1994. The core team of designers at Trolltech started developing Qt in 1992, and the first commercial version of Qt was released in 1995. Since then, Trolltech has experienced rapid growth, and Qt is currently used in thousands of successful commercial software development projects world wide. At Trolltech, we continuously work to improve and expand Qt to ensure that it always represents the state of the art in usability, look and feel, performance, and stability.[3]

[2] `http://www.qtsoftware.com/about`
[3] `http://doc.trolltech.com/4.0/trolltech.html`

Qt Information

Main Qt Site

`http://www.qtsoftware.com`

Qt In Use

`http://www.qtsoftware.com/qt-in-use`

Qt Documentation

`http://doc.trolltech.com/`
`http://doc.trolltech.com/4.5/index.html`
`http://doc.trolltech.com/4.2/index.html`

Designing Qt-Style C++ APIs

`http://doc.trolltech.com/qq/qq13-apis.html`

Qt Development Tools

- Qt Creator: Cross-Platform Qt IDE
- GUI Designer
- qmake

Qt Books

- C++ GUI Programming with Qt4 2e
  http://www.amazon.com/C-GUI-Programming-with-Qt4/
  dp/B0013TX6YE/
- The Book of Qt 4 - The Art of Building Qt Applications
  http://www.amazon.com/Book-Qt-Art-Building-Applications/
  dp/1593271476
- An Introduction to Design Patterns in C++ with Qt 4
  http://www.amazon.com/Introduction-Design-Patterns-Perens-Source
  dp/0131879057
- Foundations of Qt Development
  http://www.amazon.com/Foundations-Development-Experts-Voice-Sour
  dp/1590598318
- Rapid GUI Programming with Python and Qt (2007)
  http://www.amazon.com/Programming-Python-Prentice-Software-Devel
  dp/0132354187
- Rapid GUI Development with QtRuby (2005)
  http://www.pragprog.com/titles/ctrubyqt/rapid-gui-development-wi
- KDE Programming Bible (KDE 2, 2000)
  http://www.amazon.com/KDE-Programming-Bible-Arthur-Griffith/
  dp/0764546821

Platform Support for Qt

- Windows
- Mac OS X
- Linux/X11 (KDE)
- Windows CE
- Embedded Linux
- Target over 80 million devices with Qt for S60 (coming mid-2009)
  `http://wiki.forum.nokia.com/index.php/Category:Qt_for_S60`
- Create innovative applications for the Maemo[4] platform with Qt for Linux/X11
  `http://wiki.forum.nokia.com/index.php/Category:Maemo`

[4]Maemo is a computer architecture platform built on desktop open source components. It is aimed at enabling applications and innovative technology for mobile handheld devices. `http://www.qtsoftware.com/products/platform`

Qt Videos

- KDE windows with Qt 4.5 (2:21)
  http://www.youtube.com/watch?v=IPVd2fnUrQU
- Qt Animation Framework (0:17)
  http://www.youtube.com/watch?v=00a7eSJvWEw
- John Conways “Game of Life” in Qt 4 (0:29)
  http://www.youtube.com/watch?v=nZwwX-Eo_aM
- Qt Widgets enter the third dimension: WolfenQt (4:10)
  http://www.youtube.com/watch?v=MXS3xKV-UM0
- Qt Creator - 01 An Introduction (1:35)
  http://www.youtube.com/watch?v=U7yje3D1UM4
- OpenGL test with Qt4 (5:26)
  http://www.youtube.com/watch?v=5A0_IW08Qq8
- OpenGL test with Qt4 v2 (4:34)
  http://www.youtube.com/watch?v=mZ6vvZcvUGI

More Qt Videos

- Qt 4.5: The Cocoa Port (1:49)
  `http://www.youtube.com/watch?v=tWIq1YFRyqE`
- Qt for Mac on Cocoa in Plain English (8:42)
  `http://www.youtube.com/watch?v=B-SKcIFA7z0`
- GTK vs. Qt (7:17)
  `http://www.youtube.com/watch?v=MXS3xKV-UM0`
- CGAL: The Open Source Computational Geometry Algorithms Library (54:59)
  `http://www.youtube.com/watch?v=3DLfkWWw_Tg`

Language Support for Qt

- Python
- Ada
- Pascal
- Perl
- PHP
- Ruby
- Java (Qt Jambi)

Note: Qt Jambi — a port of Qt to the Java programming language — has been discontinued in order to focus resources on the Qt cross platform application and UI framework. Qt Jambi will be maintained for one year after the March 2009 release of Qt Jambi 4.5.0_01, and will be made available upon release under the LGPL license.[5]

[5]http://www.qtsoftware.com/products/programming-language-support

Building a Qt project

The following steps can be used to build a Qt application from the command line:

- Generate a Qt project file: `qmake -project`

  Looks at all the files *and* subdirectories to generate a project file that is used to generate a makefile for the project. The name of the project file is the directory name.

  Note: Hiding files (backups) in subdirectories doesn't seem to work, since `qmake` finds the files and adds them to the make file. Changing the name or compressing the files may hide them.

- Generate a makefile: `qmake proj.pro`
- Build application (executable): `make`
- Run/execute: `open Proj.app`

  Runs the final executable. Proper bundles are created in version 4.x.

Building MinDraw

Looking at the initial contents of the MinDraw directory before creating the project file.

```
drawarea.cpp    drawarea.h      main.cpp
mainwindow.cpp  mainwindow.h
```

Project File Creation

Create the project file using the command `qmake -project`

After creating the project file, `MinDraw.pro` is generated:

```
MinDraw.pro     drawarea.h      mainwindow.cpp
drawarea.cpp    main.cpp        mainwindow.h
```

Note that the project name is the directory with the file suffix `pro`.

Make File Creation

Create the makefile using the command `qmake MinDraw.pro`.

After creating the makefile, we see:

```
Makefile        drawarea.cpp    main.cpp        mainwindow.h
MinDraw.pro     drawarea.h      mainwindow.cpp
```

Application Creation

After running the makefile/or using QtCreator to build the application:

```
Makefile        drawarea.o      mainwindow.o
MinDraw.app     main.cpp        moc_drawarea.cpp
MinDraw.pro     main.o          moc_drawarea.o
drawarea.cpp    mainwindow.cpp  moc_mainwindow.cpp
drawarea.h      mainwindow.h    moc_mainwindow.o
```

Note: The file names starting with `moc_`. These are generated by Qt's meta object tool `moc`, the Meta-Object Compiler.

Some Qt Internals

- `QObject`
- Meta-Object System
- Signals and Slots
- `QPainter`
- `namespace Qt`

`QObject`

The `QObject` class is the base class of all Qt objects.

```
#include <QObject>
```

Note that the `Q_OBJECT` macro is mandatory for any object that implements signals, slots or properties. You also need to run `moc` (the *Meta Object Compiler*) on the source file. The use of the `Q_OBJECT` macro is strongly recommend in all subclasses of `QObject` regardless of whether or not they actually use signals, slots and properties, since failure to do so may lead certain functions to exhibit strange behavior.

`Q_OBJECT` **location**

The `Q_OBJECT` macro must appear in the private section of a class definition that declares its own signals and slots or that uses other services provided by Qt's meta-object system.

Meta-Object System

Qt's Meta-Object System provides the signals and slots mechanism for inter-object communication, run-time type information, and the dynamic property system.

The Meta-Object System is based on three things:

1. The `QObject` class provides a base class for objects that can take advantage of the meta-object system.

2. The `Q_OBJECT` macro inside the private section of the class declaration is used to enable meta-object features, such as dynamic properties, signals, and slots.

3. The Meta-Object Compiler (`moc`) supplies each `QObject` subclass with the necessary code to implement meta-object features. The `moc` tool reads a C++ source file. If it finds one or more class declarations that contain the `Q_OBJECT` macro, it produces another C++ source file which contains the meta-object code for each of those classes. This generated source file is either `#include`'d into the class's source file or, more usually, compiled and linked with the class's implementation.

Additional Meta-object Features

In addition to providing the *signals and slots* mechanism for communication between objects (the main reason for introducing the system), the meta-object code provides the following additional features:

- `QObject::metaObject()` returns the associated meta-object for the class.
- `QMetaObject::className()` returns the class name as a string at run-time, without requiring native run-time type information (RTTI) support through the C++ compiler.
- `QObject::inherits()` function returns whether an object is an instance of a class that inherits a specified class within the `QObject` inheritance tree.
- `QObject::tr()` and `QObject::trUtf8()` translate strings for internationalization.
- `QObject::setProperty()` and `QObject::property()` dynamically set and get properties by name.

Dynamic Casts

It is also possible to perform dynamic casts using `qobject_cast()` on `QObject` classes. The `qobject_cast()` function behaves similarly to the standard C++ `dynamic_cast()`, with the advantages that it doesn't require RTTI support and it works across dynamic library boundaries. It attempts to cast its argument to the pointer type specified in angle-brackets, returning a non-zero pointer if the object is of the correct type (determined at run-time), or 0 if the object's type is incompatible.

Signals and Slots

Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks.

Note: Signals in Qt are not `signals` in Unix-like operating systems.

The `QPainter` Class

`#include <QPainter>`

The `QPainter` class performs low-level painting on widgets and other paint devices.

The painter provides highly optimized functions to do most of the drawing GUI programs require. `QPainter` can draw everything from simple lines to complex shapes like pies and chords. It can also draw aligned text and pixmaps. Normally, it draws in a "natural" coordinate system, but it can also do view and world transformation.

The typical use of a painter (object):

1. Construct a painter.
2. Set a pen, a brush etc.
3. Draw.
4. Destroy the painter.

A common use of `QPainter` is inside a widget's paint event. Here's one simple example:

```
void DrawArea::paintEvent(QPaintEvent * /* event */)
{
    QPainter painter(this);

    painter.setPen( Qt::red );
    painter.drawLine ( 50,  50, 350, 350 );
    painter.drawLine ( 50, 350, 350,  50 );

    painter.setPen( Qt::green );
    painter.drawText( 200, 200, "The Center" );
}
```

Prototypes for drawing a line, text, and changing the pen color:

```
void QPainter::drawLine( int x1, int y1, int x2, int y2 );

void QPainter::drawText( int x, int y, const QString & text );

void QPainter::setPen();
```

`namespace Qt`

Is defined in `src/corelib/global/qnamespace.h`. The file is about 1600 (1625 to be exact) lines (was about 1300 lines long (1354 to be exact) in Qt 4.3). Numerous properties are defined in it.

Colors

```
    enum GlobalColor {
        color0,
        color1,
        black,
        white,
        darkGray,
        gray,
        lightGray,
        red,
        green,
        blue,
        cyan,
        magenta,
        yellow,
        darkRed,
        darkGreen,
        darkBlue,
        darkCyan,
        darkMagenta,
        darkYellow,
        transparent
    };
```

Keyboard Modifiers

```
enum KeyboardModifier {
    NoModifier           = 0x00000000,
    ShiftModifier        = 0x02000000,
    ControlModifier      = 0x04000000,
    AltModifier          = 0x08000000,
    MetaModifier         = 0x10000000,
    KeypadModifier       = 0x20000000,
    GroupSwitchModifier  = 0x40000000,
    // Do not extend the mask to include 0x01000000
    KeyboardModifierMask = 0xfe000000
};
Q_DECLARE_FLAGS(KeyboardModifiers, KeyboardModifier)
```

Videos

Questions?