Game Programming Using Qt

This book begins with an introduction to creating an application and preparing a working environment for both desktop and mobile platforms. It then dives deeper into the basics of creating graphical interfaces and Qt's core concepts of data processing and display before you try creating a game. As you progress, you'll learn how to enrich your games by implementing network connectivity and employing scripting. It then delves into Qt Quick, OpenGL, and various other tools to add game logic, design animation, add game physics, and build astonishing UI for games. In the final chapters, you'll learn to exploit mobile device features such as accelerators and sensors to build engaging user experiences. If you are planning to learn about Qt and its associated toolsets to build apps and games, this book is a must-have.

What this book will do for you...

- Install Qt on your system
- Understand the basic concepts of every Qt game and application
- Develop 2D object-oriented graphics using Qt Graphics View
- Build multiplayer games or add a chat function to your games with Qt's Network module
- Script your game with Qt Script
- Program resolution-independent and fluid UI using QML and Qt Quick
- Control your game flow using the sensors on a mobile device
- See how to test and debug your game easily with Qt Creator and Qt Test

The Beginner's Guide approach...

- Clear step-by-step instructions for the most useful tasks
- Learn by doing – start working right away
- Leave out the boring bits
- Inspiring, realistic examples that give you ideas for your own work
- Tasks and challenges to encourage experimentation

What this book will do for you...

- Install Qt on your system
- Understand the basic concepts of every Qt game and application
- Develop 2D object-oriented graphics using Qt Graphics View
- Build multiplayer games or add a chat function to your games with Qt's Network module
- Script your game with Qt Script
- Program resolution-independent and fluid UI using QML and Qt Quick
- Control your game flow using the sensors on a mobile device
- See how to test and debug your game easily with Qt Creator and Qt Test

The Beginner's Guide approach...

- Clear step-by-step instructions for the most useful tasks
- Learn by doing – start working right away
- Leave out the boring bits
- Inspiring, realistic examples that give you ideas for your own work
- Tasks and challenges to encourage experimentation

In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 2 'Installation'
- A synopsis of the book's content
- More information on Game Programming Using Qt
About the Authors

**Witold Wysota** is a software architect and developer living in Poland. He started his adventure with Qt in 2004 when he joined QtForum.org and started answering questions about programming with Qt. Shortly afterwards, he became part of the moderator group of the forum. In 2006, together with the moderator team, he established QtCentre.org, which quickly became the largest community-driven support site devoted to Qt. For a number of years, he conducted commercial, academic, and public trainings and workshops and has been giving lectures on Qt, Qt Quick, and related technologies. He is a certified Qt developer and was a member of Qt Education Advisory Board with Nokia, where he helped prepare materials in order to use Qt in educational activities.

Witold was also a technical reviewer for *Foundations of Qt Development*, Johan Thelin, *Apress Publishing*, a book about Qt 4, and an author of a couple of articles on programming with Qt.

In real life, he is a passionate adept of Seven Star Praying Mantis, a traditional style of Chinese martial arts.
Lorenz Haas is a passionate programmer who started his Qt career with Qt 3. Thrilled by Qt’s great community, especially the one at QtCentre.org, he immersed himself in this framework, became one of the first certified Qt developers and specialists, and turned his love for Qt into his profession.

Lorenz is now working at a medium-sized IT company based in Erlangen, Germany, as a lead software architect. He mainly develops machine controls and their user interfaces as well as general solutions for the industry sector. Additionally, he runs his own small consultancy and programming business called Histomatics (http://www.histomatics.de).

A few years ago, he started contributing to Qt Creator. He added a couple of refactoring options that you probably rely on a regular basis if you use Qt Creator. He is also the author of the Beautifier plugin.
Preface

As a leading cross-platform toolkit for all significant desktop, mobile, and embedded platforms, Qt is becoming more popular by the day. This book will help you learn the nitty-gritty of Qt and will equip you with the necessary toolsets to build apps and games. This book is designed as a beginner’s guide to take programmers that are new to Qt from the basics, such as objects, core classes, widgets, and so on, and new features in version 5.4, to a level where they can create a custom application with best practices when it comes to programming with Qt.

With a brief introduction on how to create an application and prepare a working environment for both desktop and mobile platforms, we will dive deeper into the basics of creating graphical interfaces and Qt’s core concepts of data processing and display before you try to create a game. As you progress through the chapters, you’ll learn to enrich your games by implementing network connectivity and employing scripting. Delve into Qt Quick, OpenGL, and various other tools to add game logic, design animation, add game physics, and build astonishing UIs for games. Toward the end of this book, you’ll learn to exploit mobile device features, such as accelerators and sensors, to build engaging user experiences.

What this book covers

*Chapter 1, Introduction to Qt,* will familiarize you with the standard behavior that is required when creating cross-platform applications as well as show you a bit of history of Qt and how it evolved over time with an emphasis on the most recent architectural changes in Qt.

*Chapter 2, Installation,* will guide you through the process of installing a Qt binary release for desktop platforms, setting up the bundled IDE, and looking at various configuration options related to cross-platform programming.

*Chapter 3, Qt GUI Programming,* will show you how to create classic user interfaces with the Qt Widgets module. It will also familiarize you with the process of compiling applications using Qt.
Preface

Chapter 4, *Qt Core Essentials*, will familiarize you with the concepts related to data processing and display in Qt—file handling in different formats, Unicode text handling and displaying user-visible strings in different languages, and regular expression matching.

Chapter 5, *Graphics with Qt*, describes the whole mechanism related to creating and using graphics in Qt in 2D and 3D. It also presents multimedia capabilities for audio and video (capturing, processing, and output).

Chapter 6, *Graphics View*, will familiarize you with 2D-object-oriented graphics in Qt. You will learn how to use built-in items to compose the final results as well as create your own items supplementing what is already available and possibly animate them.

Chapter 7, *Networking*, will demonstrate the IP networking technologies that are available in Qt. It will teach you how to connect to TCP servers, implement a reliable server using TCP, and implement an unreliable server using UDP.

Chapter 8, *Scripting*, shows you the benefits of scripting in applications. It will teach you how to employ a scripting engine for a game by using JavaScript. It will also suggest some alternatives to JavaScript for scripting that can be easily integrated with Qt.

Chapter 9, *Qt Quick Basics*, will teach you to program resolution-independent fluid user interfaces using a QML declarative engine and Qt Quick 2 scene graph environment. In addition, you will learn how to implement new graphical items in your scenes.

Chapter 10, *Qt Quick*, will show you how to bring dynamics to various aspects of a UI. You will see how to create fancy graphics and animations in Qt Quick by using the particle engine, GLSL shaders and built-in animation, and state machine capabilities, and you will learn how to use these techniques in games.

Chapter 11, *Miscellaneous and Advanced Concepts*, covers the important aspects of Qt programming that didn’t make it into the other chapters but may be important for game programming. This chapter is available online at the link https://www.packtpub.com/sites/default/files/downloads/Advanced_Concepts.pdf.
In this chapter, you will learn how to install Qt on your development machine, including Qt Creator, an IDE tailored to use with Qt. You will see how to configure the IDE for your needs and learn the basic skills to use that environment. In addition to this, the chapter will describe the process of building Qt from the source code, which can be useful for customizing your Qt installation as well as getting a working Qt installation for embedded platforms. By the end of this chapter, you will be able to prepare your working environment for both desktop and embedded platforms using tools included in the Qt release.

Installing the Qt SDK

Before you can start using Qt on your machine, it needs to be downloaded and installed. Qt can be installed using dedicated installers that come in two flavors—the online installer, which downloads all the needed components on the fly, and a much larger offline installer, which already contains all the required components. Using an online installer is easier for regular desktop installs, so we will prefer this approach.
**Time for action – installing Qt using an online installer**

First, go to [http://qt.io](http://qt.io) and click on Download. This should bring you to a page containing a list of options for different licensing schemes. To use the open source version, choose the Open Source edition licensed under GPL and LGPL. Then, you can click on the Download Now button to retrieve the online installer for the platform that you are currently running on or you can click on any of the header sections to reach a more comprehensive list of options. The links to online installers are at the beginning of the list, as shown in the following screenshot. Click and download the one suited to your host machine:

![Recommended](Recommended.png)

When the download completes, run the installer, as shown:
Click on Next and after a while of waiting as the downloader checks remote repositories, you'll be asked for the installation path. Be sure to choose a path where you have write access (it's best to put Qt into your personal directory unless you ran the installer as the system administrator user). Clicking on Next again will present you with choices of components that you wish to install, as shown in the following screenshot. You will be given different choices depending on your platform.
Choose whichever platforms you need, for example, to build native and Android applications on Linux, choose both gcc-based installation and one for the desired Android platform. When on Windows, you have to make additional choices. When using Microsoft compilers, you can choose whether to use native OpenGL drivers (the versions with the OpenGL suffix) or to emulate OpenGL ES using DirectX calls. If you don’t have a Microsoft compiler or you simply don’t want to use it, choose the version of Qt for the MinGW compiler. If you don’t have a MinGW installation, don’t worry—the installer will also install it for you.

After choosing the needed components and clicking on Next again, you will have to accept the licensing terms for Qt by marking an appropriate choice, as shown in the following screenshot. After clicking on Install, the installer will begin downloading and installing the required packages. Once this is done, your Qt installation will be ready. At the end of the process, you will be given an option to launch Qt Creator.

What just happened?
The process we went through results in the whole Qt infrastructure appearing on your disk. You can examine the directory you pointed to the installer to see that it created a number of subdirectories in this directory—one for each version of Qt chosen with the installer and another one called Tools that contains Qt Creator. You can see that if you ever decide to install another version of Qt, it will not conflict with your existing installation. Furthermore, for each version, you can have a number of platform subdirectories that contain the actual Qt installations for particular platforms.
Setting up Qt Creator

After Qt Creator starts, you should be presented with the following screen:

The program should already be configured properly for you to use the version of Qt and compiler that were just installed, but let's verify that anyway. From the Tools menu, choose Options. Once a dialog box pops up, choose Build & Run from the side list. This is the place where we can configure the way Qt Creator builds our project. A complete build configuration is called a kit. It consists of a Qt installation and a compiler that will be executed to perform the build. You can see tabs for all the three entities in the Build & Run section of the Options dialog box.

Let's start with the Compilers tab. If your compiler was not autodetected properly and is not in the list, click on the Add button, choose your compiler type from the list, and fill the name and path to the compiler. If the settings were entered correctly, Creator will autofill all the other details. Then, you can click on Apply to save the changes.
Next, you can switch to the Qt Versions tab. Again, if your Qt installation was not detected automatically, you can click on Add. This will open a file dialog box where you will need to find your Qt installation’s directory wherein all the binary executables are stored (usually in the bin directory) and select a binary called qmake. Qt Creator will warn you if you choose a wrong file. Otherwise, your Qt installation and version should be detected properly. If you want, you can adjust the version name in the appropriate box.

The last tab to look at is the Kits tab. It allows you to pair a compiler with the Qt version to be used for compilation. In addition to this, for embedded and mobile platforms, you can specify a device to deploy to and a sysroot directory containing all the files needed to build the software for the specified embedded platform.

**Time for action – loading an example project**

Qt comes with a lot of examples. Let’s try building one to check whether the installation and configuration were done correctly. In Qt Creator, click on the Welcome button on the top-left corner of the window to go the initial screen of the IDE. On the right-hand side of the page that appears (refer to the previous screenshot) there are a couple of tabs among which one of them happens to be called Examples. Clicking on that tab will open a list of examples with a search box. Make sure that the version of Qt you just installed is chosen in the list next to the search box. In the box, enter affine to filter the list of examples and click on Affine Transformations to open the project. If you are asked whether you want to copy the project to a new folder, agree. Qt Creator will then present you with the following window:
What just happened?
Qt Creator loaded the project and set a view that will help us to learn from example projects. The view is divided into four parts. Let’s enumerate them starting from the left side. First there is Qt Creator’s working mode selector that contains an action bar, which allows us to toggle between different modes of the IDE. Then, there is the project view that contains a list of files for the project. Next comes the source code editor, displaying the main part of the project’s source code. Finally, far to the right, you can see the online help window displaying the documentation for the opened example.

Time for action – running the Affine Transformations project
Let’s try building and running the project to check whether the building environment is configured properly. First, click on the icon in the action bar directly over the green triangle icon to open the build configuration popup, as shown in the following screenshot:
The exact content that you get may vary depending on your installation, but in general, on the left-hand side you will see the list of kits configured for the project and on the right-hand side you will see the list of build configurations defined for that kit. Choose a kit for your desktop installation and any of the configurations defined for that kit. You can adjust configurations by switching Qt Creator to the project management mode by clicking on the Projects button in the working mode selector bar. There, you can add and remove kits from the project and manage build configurations for each of the kits, as shown in the following screenshot:

You can adjust, build, and clean steps, and toggle shadow building (that is, building your project outside the source code directory tree).
To build the project, click on the hammer icon at the bottom of the action bar. You can also click on the green triangle icon to build and run the project. If everything works, after some time, the application should be launched, as shown in the next screenshot:

![Screenshot of the application](image)

**What just happened?**

How exactly was the project built? If you open the **Projects** mode and look at **Build Settings** for a kit assigned to the project (as seen in one of the previous screenshots), you will notice that a number of build steps were defined. The first step for Qt projects is usually the qmake step, which runs a special tool that generates a Makefile for the project that is fed in the second step to a classic make tool. You can expand each of the steps by clicking on the respective **Details** button to see configuration options for each of the steps.

While make is considered as a standard tool for building software projects, qmake is a custom tool provided with Qt. If you go back to the **Edit** mode and see which files are listed in the project contents, you will notice a file with a pro extension. This is the main project file that contains a list of source and headers files in the project, definitions of Qt modules active for the project, and optionally, external libraries that the project needs to link against. If you want to learn the details of how such project files are managed, you can switch to the **Help** mode, choose **Index** from the drop-down list on the top of the window, and type qmake Manual to find the manual for the tool. Otherwise, just let Qt Creator manage your project for you. For self-contained Qt projects, you don’t need to be a qmake expert.
Building Qt from sources

In most cases for desktop and mobile platforms, the binary release of Qt you download from the webpage is sufficient for all your needs. However, for embedded systems, especially for those ARM-based systems, there is no binary release available or it is too heavy resource-wise for such a lightweight system. In such cases, a custom Qt build needs to be performed. There are two ways to do such a build. One is to download the sources as a compressed archive just like the binary package. The other is to download the code directly from a Git repository. Since the first way is pretty much self-explanatory, we'll focus on the second approach.

Time for action – setting up Qt sources using Git

First, you need to install Git on your system if you don’t already have it. How to do that depends on your operating system. For Windows, simply download an installer from https://git-for-windows.github.io. For Mac, the installer is available at http://code.google.com/p/git-osx-installer. For Linux, the simplest way is to use your system’s package manager. For instance, on Debian-based distributions, just issue the `sudo apt-get install git` command on a terminal and wait until the installation gets completed.

Afterwards, you need to clone Qt’s Git repository. Since Git is a command-line tool, we’ll be using the command line from now on. To clone Qt’s repository to a directory where you want to keep the sources, issue the following command:

```
git clone git://code.qt.io/qt/qt5.git
```

If all goes well, Git will download a lot of source code from the network and create a `qt5` directory, containing all the files that were downloaded. Then, change the current working directory to the one containing the freshly downloaded code:

```
cd qt5
```

Then you need to run a Perl script that will set up all the additional repositories for you. If you don’t have Perl installed, you should do that now (you can get Perl for Windows from http://www.activestate.com/activeperl/downloads). Then, issue the following command:

```
perl init-repository
```

The script will start downloading all the modules required for Qt and should complete successfully after a period dependent on your network link speed.
What just happened?

At this point in the qt5 directory, you have a number of subdirectories for different Qt modules (some of them were mentioned in Chapter 1, Introduction to Qt) each with a local Git repository containing the source code for the respective Qt modules and tools. Each of the modules can be updated separately if required.

Time for action – configuring and building Qt

Having the sources in place, we can start building the framework. To do that, in addition to a supported compiler, you will need Perl and Python (Version 2.7 or later) installed. For Windows, you will also need Ruby. If you are missing any of the tools, it’s a good time to install them. Afterwards, open the command line and change the current working directory to the one containing the Qt source code. Then, issue the following command:

configure -opensource -nomake tests

This will launch a tool that detects whether all the requirements are met and will report any inconsistencies. It will also report the exact configuration of the build. You can customize the build (for example, if you need to enable or disable some features or cross-compile Qt for an embedded platform) by passing additional options to configure. You can see the available options by running configure with the -help switch.

If configure reports problems, you will have to fix them and restart the tool. Otherwise, start the build process by invoking make (or an equivalent like mingw32-make if using MinGW or nmake if using MSVC).

Instead of nmake, you can use a tool called jom that is bundled with Qt. It will reduce the compilation time on multicore machines, which is what the default nmake tool can’t do. For make and mingw32-make, you can pass the -j N parameter, where N stands for the number of cores in your machine.

What just happened?

After some time (usually less than an hour), if all goes well, the build should be complete and you will be ready to add the compiled framework to the list of kits available in Qt Creator.

In Unix systems after the build gets completed, you can invoke a make install command with super-user privileges (obtained for example, with sudo) to copy the framework to a more appropriate place.
Summary

By now, you should be able to install Qt on your development machine. You can now use Qt Creator to browse the existing examples and learn from them or to read the Qt reference manual to gain additional knowledge. You can also just start a new C++ project and start writing code for it, build, and execute it. Once you become an experienced Qt developer, you will also be able to make your own custom build of Qt. In the next chapter, we will finally start using the framework and you will learn how to create graphical user interfaces by implementing our very first simple game.
Where to buy this book

You can buy Game Programming Using Qt from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers. Click here for ordering and shipping details.