QT: INTERNALS AND PRINCIPLES

Get to Grips with Qt

Martin Rotter

January 18, 2013

Abstract

Qt is one of the best-known and the mightiest general-purpose libraries available. Its functionality covers each and every thinkable programming area, including threading, graphical interfaces, relational databases, networks, 2D/3D painting and many more. This text aims at one modest task - providing the solid base for learning and understanding Qt by revealing its internals and principles.

Acknowledgements

Firstly, I would like to thank to my bachelor thesis leader Mgr. Tomáš Kühr. Special praise goes to my closest friends, especially to my family and to people who let me know that love can be wonderful.

Preface

Whom this book is for?

This book is for anyone who is interested in creating dynamic and multi-platform applications using Qt framework. It does not matter if you are hugely experienced software engineer or self-taught enthusiast. Information included in this book will be useful either way.

What is covered by this book?

Covering all compoments of Qt framework in one book is impossible task because of massive complexity of those libraries. It's better to focus on certain aspects only. Each Qt-related books begins with graphical interfaces. Probably, that's not the best approach because graphical interfaces form very complex science. You need to be able to manage easy Qt-related tasks first in order be able to master harder ones.

That's why this book starts with very fundamental topics, therefore pushing graphical-interfaces-related topics back to further chapters. You will learn the Qt framework versioning system, compilation process or Qt classes structure within the framework. Meta-object system will follow. Understanding meta-object system is the key for next progress and is the main precondition for building solid Qt-based applications. Graphical interfaces will be explored deeply too.

After managing basics of Qt libraries usage, we can advance to other needed topics, such as networking, relational dabases and threading.

Finally, you will apply your newly gained knowledge to build applications, which can be easily maintainable, compilable and easy to package and ship to your customers.

This books equippes you primarily with principles. Facts (which are unknown to you and are not included in this book) can be found in (Qt-Project, 2012).

Note that in this paper, we discuss relatively new (as of January, 2013) Qt 5. You will learn more about Qt 5 new features later.

What is not covered by this book?

As said earlier, it's not possible to cover all nooks of Qt libraries in one book. We will omit some hugely admired Qt features, so that we can concentrate on other ones. Qt Meta Language (QML) will be ignored completely, along with whole QtQuick and other stuff for cell-phones or tablet devices. 2D and 3D painting features won't be described too but you will clap on them from time to time as they are needed for advanced Graphical User Interface (GUI) tweaking.

Some other parts of Qt are ignored too. You will be informed about some of them throughout the book.

How this book is structured?

As we said earlier, there are basically two main stories told by this book. First one lets you know something about Qt and its features. Analogy to this story is called Laboratory Qt and it is the first part of the book.

Then, you will learn to use your Qt-related skills in a part called Real-world Qt and that's the second (and more exciting) story.

Are there any prerequisities?

Of course there are. Qt itself is based on the C++ programming language and thus C++ knowledge is main prerequisite. One could argue that Qt has bindings into many better programming languages and I would respond: "It's true." But C++ is core language for Qt and for you, as future Qt developer, using Qt in its native programming language is important.

C++ went through massive update recently and we face its eleventh version. So we will use C++ 11 in this book. You can learn more about C++ 11 in (Du Toit, Stefanus, 2012) or in section 1.3.

Text formatting

This book is riddled with pictures, tables and other fancy elements. There are also source code fragments included as seen in Listing 1.

Listing 1: Sample code fragment

```
1 int main(int argc, char *argv[]) {
2     return EXIT_SUCCESS;
3 }
```

Note that sometimes it is needed to highlight *portion of text* or even make it **really visible**. In some cases, there is a need of providing some extra remark to discussed topic. Typical remark looks similar to one below.

One Step Further
This is very interesting text here...

Source code

Topics of this book are supplemented sample applications to describe the matter. You can find source code in *sources* subdirectory.

Licensing

Contents

Part I
Laboratory Qt

1 Foreword

Qt framework is one of the greatest libraries ever made. You probably use it and you don't even know about it. If you use Skype (for online communication) or K Desktop Environment (KDE), then you use Qt too, because those applications are based on Qt.

Skype uses just graphical interface made in Qt but KDE is totally based on Qt as it uses not just graphical interface from Qt but other components too.

Qt penetrated the world of interactive applications and now it can be found even in devices, where it's not generally expected. First public version of Qt was released in 1995 and huge progress was achieved since that time.

In a flow of time, Qt began to be perceived as very dynamic library which is particularly great for graphical interface design. There was very good reason for such an opinions because KDE was released in 1996, invoking quite a sensation. In short, its desktop environment looked great and overpowered other major environments in this aspect. Qt was pushed forward by those events and became massively popular. The only goal of Qt was to be a good library for anyone who does desktop programming.

As years passed by, Qt was more and more robust, KDE made its progress through version 3 and 4, and things have changed. Presently, desktop does not mean everything for application developer. Today cyber-world needs to be interconnected and people want to be mobile. You can't do that with desktop environment running on personal computer. You need cell-phone. Cell-phone with (possibly) good-looking environment and fancy applications. Unfortunately, Qt 4 was not able to offer this kind of functionality to its users - programmers, so they looked at the competition and chose Android as their platform, leaving Qt behind.

Luckily, Qt 5 appeared, bringing us some new exciting features, giving itself a chance to compete its opponents in category of mobile development toolkits. If we add rock-solid desktop features, we have versatile and stable base to build on.

1.1 What is Qt?

As said previously, Qt is framework, toolkit or, simply, set of libraries. It has very roots in Norway. Original creators are Haavard Nord and Eirik Chambe-Eng. Basically Qt framework consists of:

- set of libraries written in C++

- meta-object compiler
- QtScript interpreter
- tools for internationalization and GUI design
- scripts for various build systems like CMake
- other tools, e. g. integrated development environment, examples or documentation browser

So as you see, Qt is not just collection of header/source files. It's completed with a variety of other stuff. You will learn more about Qt structure in section 2.

1.2 Companies behind Qt

Qt lives for more than two decades and its owners changed accordingly. Haavard Nord and Eirik Chambe-Eng assembled themselves in a team and called it Quasar Technologies. Later company was renamed to Trolltech. This company led Qt development for period of 12 exciting years, preffering desktop development.

But as we know, things have changed and smartphones became massively popular lately. That's why Trolltech was acquired by Nokia. It was obvious that Nokia can bring something new to Qt as it is leading company in smartphones world production. Nokia promised that they would keep Qt open-souce and made it available via public Git[1] repository. But Nokia somehow was not able to utilize potential of Qt and sold it to another company called Digia.

1.2.1 Licensing

Qt uses two separate licenses:

1. **Commercial license**, which provides you (as indie developer) with possibility to produce *closed-source* (proprietary) or *open-source* applications, you can do whatever you want with your copy of Qt. This kind of license is usually sold per particular platform and it is generally rather expensive. It may cost around several thousands US dollars and this price may get even higher if you buy license for more platforms or if you have bigger development team. This license is usually bought by developers who want to sell their software for money and/or stay closed-source, otherwise open-source license is much better choice.

[1] Git is revision control system originally created to support Linux kernel development. Founding author is well-know Linus Torvalds. Git is multi-platform and runs on Windows, Linux or Mac OS X. It's Portable Operating System Interface (POSIX)-compatible.

Commercial license grants you even more rights. You can link Qt statically to your application and/or include other proprietary software in it. Technical support is available for commercial users.

2. **Open-source** license, which provides you (and your users) with much more freedom but forcing you to share source code of your application with the community and allowing anyone to change your application and redistribute it under the same terms. Used license is GNU LGPL license in version 2.1, allowing you to use GNU GPL(Stallman, Richard M. 2007) for your projects.

Licenses have always been quite a problem for Qt framework. Commercial license was fine. But non-commercial was not. Qt used its own license before GNU GPL was chosen as the primary one. Problem was that Q Public License wasn't GPL compatible. This problem became much more obvious when KDE established itself as one the most favored desktop environments, gaining milions of users. They were naturally afraid of KDE becoming the piece of proprietary software, which was more or less possible with Q Public License.

Luckily this problem got solved by releasing Qt under GNU GPL.

1.3 C plus plus as base stone

C++ is known as general-purpose programming language, based on famous C. It was created around 1979 by Bjarne Stroustrup, bringing in many Object-oriented programming (OOP) features such as implementation of classes, polymorphism, entity overloading or inheritance. You can find very tiny example of basic techniques in Listing 2.

Listing 2: Basic OOP techniques in C++

```
/* Base class declaration */
class BaseClass {
    public:
        BaseClass();

        void whoAmI() const;
};

/*
 * Class declaration
 * This class inherits BaseClass.
 */
class InheritingClass : public BaseClass {
    public:
        InheritingClass();

```

```
17          void whoAmI() const;
18  };
19
20  /* Example usage of BaseClass and InheritingClass classes. */
21  int main() {
22      BaseClass class_1;
23      InheritingClass class_2;
24      class_1.whoAmI();
25      class_2.whoAmI();
26
27      BaseClass *class_3 = &class_2;
28      class_3->whoAmI();
29
30      ((InheritingClass*) class_3)->whoAmI();
31
32      return 0;
33  }
```

Listing 3: Output of application from Listing 2

```
1  BaseClass instance constructed.
2  BaseClass instance constructed.
3  InheritingClass instance constructed.
4  I am BaseClass.
5  I am InheritingClass.
6  I am BaseClass.
7  I am InheritingClass.
```

C++ has many characteristics – some are bad while other ones may be great. Let's compare usefulness of its abilities.

SYNTAX bad

C++ is known to have some oddities rooted in its syntax. E. g. we can be confused by rife usages of `const` keyword. One `const` marks methods which can operate only with constant objects and another distinguishes constant variables from non-constant ones. Even the greatest fan of C++ has to admit bizarre usage of this keyword. You can read about this topic in (Prata, Stephen, 2011, p. 90-92, p. 537).

POINTERS vs. REFERENCES bad

This could be one of conventions-related issues. Programmers are not entirely sure whether to use pointers or references for passing values to functions. Generally, terms of references and pointers usage are not strictly set.

MEMORY MANAGEMENT bad, good

This is very discussed topic these years as many programmers transitioned to programming languages, which produce *managed code*. Nowadays programmers heavily depend on managed code and they have troubles with manual object deletion and other related actions.

C++ is considered to be a fairly low-level programming language. Its "*low-levelness*" applies to the way the memory is managed. In this case, no automatic memory management is implemented, yielding responsibility to the programmer. He (or perhaps she) has to take care of memory allocation and deallocation. There is certainly quite big pronenes to errors in this approach. Programmers simply forgets to free allocated memory space and memory leak occurs.

In the other, manual management of allocated objects gives programmer bigger power to control application memory consumption and that's perfect on devices with limited system memory. Manual control of object life can be also much faster than automatic resource management provided by *garbage collectors*.

Neither virtual machine nor complex runtime environment supports execution of C++ application, thus "nobody" supervises actions of your application, except operating system. Your application is left alone with its segment of primary memory and your application is entrusted with everything, including memory management.

One Step Further
Term *managed code* means that all resources (usually called *objects* in the object-oriented programming) generated by code execution are maintained and managed by an external entity. This entity is often called *a virtual machine* and usually includes sophisticated garbage collector, which is responsible for freeing needless resources from memory.

THREADING bad

C++ doesn't contain unified interface for threading.[2] That could make pure C++ poorly usable for developing more complex applications if no 3rd-party threading library is not available.

FAST CODE EXECUTION great

C++ code execution is amazingly fast compated to other modern programming languages. Direct compilation (see more in section 4) into machine

[2] Threading is supported in new C++ 11 standard. You can read about threading inclusion in (Du Toit, Stefanus, 2012, p. 1114-1160).

code is the cause here. Other favorite languages are compiled into bytecode, thus they have to be compiled just-in-time by virtual machine and that is time consuming job, thus making application execution slow.

Let's make a little test and compare C++ with C# . C# code is known to be compiled into Intermediate Language (IL), which is bytecode, and ran by special runtime.

One of the simplest tasks to compare these two languages could be simple integer array sorting. Quicksort algorithm will do that. Consider implementations in C++ (Listing 4) and C# (Listing 5). Furthermore, we can use try to maximally optimize C# code execution speed by allowing "unsafe code" and using pointers instead of references. This approach is shown in Listing 6.

Series of sample sortings was made with each implementation. Subject of sorting was array filled with descendingly-valued integers. Such an array can be denoted as $Array = \{x, x-1, x-2, \ldots, 0\}$. Series contains 20 these arrays. Results of comparison are display in Table 1.

Listing 4: Quicksort implementation in C++

```
void QuickSort::quickSort(int *array, int p, int r) {
    int q;
    if (p < r) {
        q = partition(array, p, r);
        quickSort(array, p, q - 1);
        quickSort(array, q + 1, r);
    }
}

int QuickSort::partition(int *array, int p, int r) {
    int x = array[r];
    int i = p - 1;
    int j;
    for (j = p; j < r; j++) {
        if (array[j] <= x) {
            i += 1;
            swap(&array[i], &array[j]);
        }
    }
    swap(&array[i + 1], &array[r]);
    return i + 1;
}

void QuickSort::swap(int *lhs, int *rhs) {
    int temp = *lhs;
    *lhs = *rhs;
    *rhs = temp;
```

```
}
```

Listing 5: Quicksort implementation in C#

```
static void quickSort(int[] array, int p, int r) {
    int q;
    if (p < r) {
        q = partition(array, p, r);
        quickSort(array, p, q - 1);
        quickSort(array, q + 1, r);
    }
}

static int partition(int[] array, int p, int r) {
    int x = array[r];
    int i = p - 1;
    int j;
    for (j = p; j < r; j++) {
        if (array[j] <= x) {
            i += 1;
            swap(ref array[i], ref array[j]);
        }
    }
    swap(ref array[i + 1], ref array[r]);
    return i + 1;
}

static void swap(ref int lhs, ref int rhs) {
    int temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

Listing 6: Quicksort implementation in "unsafe" C#

```
static unsafe void quickSort(int* array, int p, int r) {
    int q;
    if (p < r) {
        q = partition(array, p, r);
        quickSort(array, p, q - 1);
        quickSort(array, q + 1, r);
    }
}

static unsafe int partition(int* array, int p, int r) {
    int x = array[r];
    int i = p - 1;
```

```
    int j;
    for (j = p; j < r; j++) {
        if (array[j] <= x) {
            i += 1;
            swap(&array[i], &array[j]);
        }
    }
    swap(&array[i + 1], &array[r]);
    return i + 1;
}

static unsafe void swap(int* lhs, int* rhs) {
    int* temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

Table 1: Results of C++ vs. C# comparison

(a) C++

array size	final time
800	0.001
1600	0.002
2400	0.004
3200	0.008
4000	0.011
4800	0.016
5600	0.022
6400	0.027
7200	0.034
8000	0.043
8800	0.05
9600	0.06
10400	0.071
11200	0.082
12000	0.094
12800	0.107
13600	0.122
14400	0.134
15200	0.15

(b) C#

array size	final time
800	0.3
1600	0.5
2400	0.12
3200	0.20
4000	0.31
4800	0.45
5600	0.66
6400	0.80
7200	0.101
8000	0.124
8800	0.152
9600	0.180
10400	0.215
11200	0.245
12000	0.283
12800	0.324
13600	0.364
14400	0.409
15200	0.456

(c) “Unsafe” C#

array size	final time
800	0.2
1600	0.2
2400	0.4
3200	0.8
4000	0.14
4800	0.19
5600	0.25
6400	0.31
7200	0.40
8000	0.49
8800	0.60
9600	0.72
10400	0.87
11200	0.99
12000	0.113
12800	0.129
13600	0.147
14400	0.164
15200	0.184

We see that C++ outperformed classic C# implementation, while being a-

round 3 times faster. even “unsafe” C# implementation got beaten, although the difference was tiny. So we can state that C++ is faster than C# even in fairly simple task. You may think about performance difference if hugely complex computation (perhaps some 3D graphical computation) is needed to be done.

HUGE COMMUNITYgreat

Plenty of world-renowned software is written using C++ , including many 3D games, almost each program from Adobe and Chromium web browser. Many C++ books are available, making it easier to learn.

MEMORY CONSUMPTIONgood

C++ applications, as stated, need no virtual machine for their execution. They load just base C++ library and extra libraries if needed. Such approach makes significant opposite to robust and greedy (as for memory) runtime environments of certain high-level languages. We can mention primarily .NET Framework and Java Runtime Environment (JRE).

CODE PORTABILITYbad

When it comes to code portability (same as multi-platformity), C++ leaves its audience uncertain. Users can be sure about portability of C++ standard library but that’s all. Standard library is not trully packed with stunning features, forcing you to use 3rd-party libraries for advanced functionality. Those libraries don’t have to be multi-platform, however, which can result in pain rooted from hypothetical need to port your application to another platform.

1.3.1 Version 11 and its enhancements

C++ programming language was for the first time standardized in 1998. This version is known as C++ 98.

1.4 Qt components

1.4.1 Supported platforms

1.4.2 Qt 5 additions

1.5 Getting and installing Qt

1.5.1 Installing on Windows

1.5.2 Installing on Linux

1.5.3 Compilling Qt

2 Qt framework structure

2.1 Tools and chains

2.2 Modules and linking

3 Using Qt framework

3.1 Qt Creator

3.2 Including Qt header files

4 Compilation process

4.1 Compilers, linkers, assemblers...

4.2 Executable files and their structure

4.3 Classic C plus plus compilation

4.4 Qt-way C plus plus compilation

Part II
Real-world Qt

List of Figures

List of Tables

List of Listings

List of Abbreviations

GUI	Graphical User Interface
IL	Intermediate Language
JRE	Java Runtime Environment
KDE	K Desktop Environment
OOP	Object-oriented programming
POSIX	Portable Operating System Interface
QML	Qt Meta Language

References

Du Toit, Stefanus

2012 *Working Draft N3337, Standard for Programming Language*, tech. rep., International Organization for Standardization/International Electrotechnical Commission.

Nigel, Christian

2010 *Professional C# and .NET 4*, Wiley-Publishing, ISBN: 978-0-470-50225-9.

Prata, Stephen

2011 *Primer Plus*, 6th ed., Addison-Wesley, ISBN: 0-321-77640-2.

Qt-Project

2012 *Qt 5 Online Reference Documentation*, http://qt-project.org/doc/qt-5.0/ (visited on 01/14/2013).

2013 *Qt Online Wikipedia*, http://qt-project.org/wiki/ (visited on 01/14/2013).

Stallman, Richard M.

2007 *GNU General Public License*, version 3, Free Software Foundation, http://www.gnu.org/copyleft/gpl.html (visited on 08/29/2012).

Index