The Qt Company

QT AND QML ESSENTIALS 010-003

Exam Curriculum

The Qt Company provides Qt and QML developers with three kinds of certification exams:

- *Qt and QML Essentials*
- *Widget UI and Application Engine with Qt*
- *Qt Quick UI*

Exam candidates may receive Certified Qt and QML Developer, Certified Qt C++ Specialist, and Certified Qt Quick Specialist certificates by passing the exams. A certificate is a valuable document, endorsing one's Qt and QML knowledge to an employer or a customer.

To achieve the Qt and QML Developer status, an exam candidate is required to pass the Qt and QML Essentials exam. The Qt C++ or Qt Quick Specialist status is granted to candidates, who additionally pass either or both Widget UI and Application Engine and Qt Quick UI exams. The former specialist exam tests candidates' knowledge of Qt C++ APIs, including, e.g., widgets, threads, model/view framework, and QObject. The latter exam tests Qt Quick and QML knowledge.

The exams can be taken in any order, but the candidate cannot receive either of the specialist certificates before the Qt and QML Essentials exam has been passed as well. So the Qt and QML Developer certificate is required for both specialist certificates as well.

Certificate exams can be taken in any authorized PearsonVUE test center. The details of the test center locations and instructions how to make an appointment and attend the exam can be found at http://www.pearsonvue.com/qtcompany/. The exam price varies from test center to test center. The exact price can be inquired directly from test centers.

Qt and QML Essentials exam will test candidates' basic Qt and QML knowledge. The exam curriculum is defined in detail in this document. The exam contains 50 multi choice questions and the candidate has 60 minutes to select the correct statement(s). No coding is required in the exam, although the questions and question options may contain short code snippets.

1 FUNDAMENTALS OF QT AND QML PROGRAMMING

Exam candidates need to know how to create, build, and debug basic Qt and QML applications using QtCreator IDE. They need to know Qt modules, Qt licensing options, and QObject features, such as meta-object system, properties, and event handling.

1.1 QT MODULES

- *The need for cross platform libraries*
- *Overview of Qt modules and their key functionality*
- *The difference between native APIs and cross platform APIs*
- *The difference between toolkits and frameworks*

1.2 QT LICENSING

- Know Qt licensing options: GPL, LGPL, commercial
- Understand how to choose the right license
- Understand the difference between application development and device creation licenses
- Basic differences between GPLv2, GPLv3, LGPLv2.1, and LGPLv3

1.3 BASIC DEVELOPMENT STEPS

- Simple widget-based and QML-based application creation using wizards in QtCreator
- Documentation, community, and other sources for help
- Qt project file (.pro) creation and management
- qmake key variables, basic syntax, and variable values
- Basic development and build tools (Qt Designer, Qt Help, Qt Linguist, qmake, Makefiles, moc, uic)

1.4 QTCREATOR

- Setup of kits, Qt versions, compilers, and debuggers
- Creation of projects using wizards
- Basic development steps, including edit, code style, build, debug, and deploy
- *Project Build and Run Settings* in *Projects* mode

1.5 DEBUGGING AIDS

- Be able to log debug messages in C++ and QML
- Understand the usage of asserts

1.6 QOBJECT

Memory Management

- Parent-child relationship and object ownership
- Stack and heap allocations
- Qt smart pointers for managing object life-time and object sharing

Meta-Object System

- Meta-object concept
- *Q_OBJECT* usage
- Meta-object creation
- Meta-object contents
- Properties

Signals and Slots

- Observer patterns and signals and slots
- Signal/slots advantages
- Signal/slots connections using string-based and pointer-based signals/slots
- Signal/slot overloading

- *Connection types and use cases, when to use each type*

Events and Event Handling

- *Event concept*
- *Differences between events and signal/slots*
- *Event handler implementation*
- *Event filters and their implementation*
- *Event propagation*
- *Event types (spontaneous, synthetic, synchronous, asynchronous)*
- *Custom events and custom event handling*

2 ESSENTIAL QT CORE CLASSES

Exam candidates must be able to use QObject and value-based classes properly and understand the differences between them. The functionality and APIs of some value-based essentials classes, like QString, QByteArray, and item containers should be known in detail. Also the understanding of the Qt meta-type system and QVariant is required in the exam.

2.1 VALUE-BASED CLASSES

- *The concept of implicit sharing*
- *Memory allocation and usage of implicit shared classes*

QString, QByteArray, QUrl, QRegularExpression

- *String allocation and copying*
- *Basic string manipulation functions*
- *Text coding*
- *Differences between ASCII literals, strings, and byte arrays*
- *Know when it is beneficial to use QByteArray instead of QString*
- *Understand QUrl and the components of an URL*
- *Using regular expressions in string manipulation (candidates do not need to manage regular expression syntax itself)*

Qt Meta-Type System and QVariant

- *Understand the central role of QVariant for many other container classes*
- *Know which types are supported by default and be able to store new data types*
- *Know QVariant essentials functions for type checking and conversions*

2.2 ITEM CONTAINERS

- *Container concept*
- *Differences between Java and STL styled APIs*
- *Container types*
- *To be able to identify the best suited container type for your needs*
- *Requirements for a type to be storable in a container*
- *Container iterators*
- *Difference between standard and mutable iterators*

- *The foreach keyword on containers*
- *Know essential STL algorithms*

2.3 OTHER ESSENTIAL QT CORE CLASSES

QSignalMapper

- *Signal mapper usage*

QTimer

- *QTimer concept*
- *QTimer usage*
- *Timer operation modes*

Generic IO, Files and Printing

- *QFile usage*
- *QIODevice read and write functionality*
- *File I/O using Qt*
- *Text encoding and files*
- *Differences between QDataStream and QTextStream*
- *Know the related classes for file and directory access*
- *Text document printing in PDF format using QPdfWriter*

3 COMMON APPLICATION BUILDING BLOCKS

Candidates should be able to manage application resource, create locale-aware applications, and store application settings to a persistent storage.

3.1 RESOURCES

- *Understand the advantage of resource files over ordinary files*
- *Loading of resources*
- *Built-in and external resources*
- *Accessing resource elements*
- *Learn how to add resource file to the project*
- *Know the resource XML file structure*
- *Be able to access resources from the application*

3.2 SETTINGS

- *Know how to store application settings in different formats with help of QSettings*
- *Be aware of and be able to control the storage location of settings on different platforms*

3.3 LOCALIZATION

- *Know the basics about Qt support of translation of strings into other languages*

- *Know that resource files can be localized*
- *QLocale class and its API*

4 DEVELOPING A WIDGET APPLICATION

Candidates must be able to create widget-based applications using QtCreator wizard. The basic structure of main windows and the principles of layout management should be understood as well.

4.1 WINDOWS AND WIDGETS

- *Differences between windows (QWindow) and widgets (QWidget)*
- *Window and widget programming APIs at high level: window management, geometry, event handling*

4.2 ESSENTIAL WIDGETS

- *QMainWindow usage*
- *Know how to implement a menu*
- *Be able to use a toolbar and status bar*
- *Understand the concept of actions*
- *Use of actions*
- *Know how to use QVariant to store data in actions*
- *Scrolled areas*
- *Central widget*
- *Docked widgets*
- *Input widgets, display widgets, container widgets*

4.3 LAYOUT MANAGEMENT

- *Geometry and geometry manipulation of a widget*
- *Top-level widgets*
- *Layout managers, their usage, and limitations*
- *The use of Qt Designer tool*

5 DEVELOPING A QML APPLICATION

In addition to widget-based application development, exam candidates must be able to create simple QML applications with basic QML elements. They must know the options for handling element layouts, how to use signal handlers, how to make property bindings, and how to create custom items with custom properties and signals.

5.1 QT QUICK AND QML

- *Essential Qt Quick classes: QQuickView, QQuickItem, QQuickPaintedItem (no need to be able to subclass item classes)*
- *The role of QQuickItem*
- *The relationship between Item and QQuickItem*
- *Item features*
- *Essential QML module classes: QQmlEngine, QQmlComponent, QQmlContext*
- *QML Designer*

- *Basic QML debugging*
- *QML profiling*
- *QML design process*
- *QML modules and import*
- *Module versioning*

5.2 CREATING QML-BASED UIS

- *Element hierarchy, parents and children*
- *Visual parenting*
- *Stacking order*
- *Types, properties, attached properties*
- *Methods*
- *Essential element types for graphics, user interactions, and layout*
- *Properties*
- *Property bindings*
- *Custom items*
- *Custom properties and property aliases*
- *Custom signals*

5.3 LAYOUTS

- *Nested layouts*
- *Anchors: anchor lines and item-based anchors, margins*
- *Positioners: row, column, grid, flow, spacing*
- *Element painting and clipping*
- *Element coordinates*
- *Element z-value*

5.4 USER INTERACTIONS

- *Mouse interactions: clicking, double-clicking, mouse buttons, hovering, dragging*
- *Key handing*
- *Focus request*
- *FocusScope*
- *Touch handling, touch points*
- *Gestures: tap, long tap, flickable, pinch gesture*