QML

QML (**Qt Modeling Language**[4]) is a user interface markup language. It is a declarative language (similar to CSS and JSON) for designing user interface–centric applications. Inline JavaScript code handles imperative aspects. It is associated with Qt Quick, the UI creation kit originally developed by Nokia within the Qt framework. Qt Quick is used for mobile applications where touch input, fluid animations and user experience are crucial. QML is also used with Qt3D[5] to describe a 3D scene and a "frame graph" rendering methodology. A QML document describes a hierarchical object tree. QML modules[6] shipped with Qt include primitive graphical building blocks (e.g., Rectangle, Image), modeling components (e.g., FolderListModel, XmlListModel), behavioral components (e.g., TapHandler, DragHandler, State, Transition, Animation), and more complex controls (e.g., Button, Slider, Drawer, Menu). These elements can be combined to build components ranging in complexity from simple buttons and sliders, to complete internet-enabled programs.

QML elements can be augmented by standard JavaScript both inline and via included .js files. Elements can also be seamlessly integrated and extended by C++ components using the Qt framework.

QML is the language; its JavaScript runtime is the custom V4 engine,[7] since Qt 5.2[8]; and Qt Quick is the 2D scene graph and the UI framework based on it. These are all part of the Qt Declarative module, while the technology is no longer called Qt Declarative.

QML and JavaScript code can be compiled into native C++ binaries with the Qt Quick Compiler.[9] Alternatively there is a QML cache file format[10] which stores a compiled version of QML dynamically for faster startup the next time it is run.

Contents

QML

Paradigm	Multi-paradigm: declarative, reactive, scripting
Developer	Qt Project
First appeared	2009
Stable release	5.15.0[1] / May 26, 2020
Typing discipline	dynamic, strong
Website	qt-project.org/doc/qt-5/qmlapplications.html (http://qt-project.org/doc/qt-5/qmlapplications.html)
Influenced by	
XAML,[2] JSON, JavaScript, Qt	
Influenced	
Qt, Ring[3]	

QML

Filename extension	`.qml`
Developed by	Qt Project
Type of format	Scripting language
Website	qt-project.org/doc/qt-5/qmlapplications.html (http://qt-project.org/doc/qt-5/qmlapplications.html)

Adoption

- KDE Plasma 4[11] and KDE Plasma 5 through Plasma-framework
- Liri OS (https://liri.io/)
- Simple Desktop Display Manager
- reMarkable tablet device (https://remarkable.com/)[12][13]
- Unity2D[14]
- Sailfish OS[15][16][17]
- BlackBerry 10[18]
- MeeGo[19][20]
- Maemo[21]
- Tizen[22]
- Mer[23][24][25]
- Ubuntu Phone[26]
- Lumina (desktop environment)[27]
- Many open-source applications (https://github.com/search?l=QML&q=qml&type=Repositories)

Syntax, semantics

Basic syntax

Example:

```
import QtQuick 2.9  // import from Qt 5.9

Rectangle {
    id: canvas
    width: 250
    height: 200
    color: "blue"

    Image {
        id: logo
        source: "pics/logo.png"
        anchors.centerIn: parent
        x: canvas.height / 5
    }
}
```

Objects are specified by their type, followed by a pair of braces. Object types always begin with a capital letter. In the example above, there are two objects, a Rectangle; and its child, an Image. Between the braces, one can specify information about the object, such as its properties. Properties are specified as property: value. In the example above, we can see the Image has a property named source, which has been assigned the value `pics/logo.png`. The property and its value are separated by a colon.

The id property

Each object can be given a special unique property called an id. Assigning an id enables the object to be referred to by other objects and scripts. The first Rectangle element below has an id, `myRect`. The second Rectangle element defines its own width by referring to `myRect.width`, which means it will have the same width value as the first Rectangle element.

```
Item {
    Rectangle {
        id: myRect
        width: 120
        height: 100
    }
    Rectangle {
        width: myRect.width
        height: 200
    }
}
```

Note that an id must begin with a lower-case letter or an underscore, and cannot contain characters other than letters, digits and underscores.

Property bindings

A property binding specifies the value of a property in a declarative way. The property value is automatically updated if the other properties or data values change, following the reactive programming paradigm.

Property bindings are created implicitly in QML whenever a property is assigned a JavaScript expression. The following QML uses two property bindings to connect the size of the rectangle to that of otherItem.

```
Rectangle {
    width: otherItem.width
    height: otherItem.height
}
```

QML extends a standards-compliant JavaScript engine, so any valid JavaScript expression can be used as a property binding. Bindings can access object properties, make function calls, and even use built-in JavaScript objects like Date and Math.

Example:

```
Rectangle {
    function calculateMyHeight() {
        return Math.max(otherItem.height, thirdItem.height);
    }
    anchors.centerIn: parent
    width: Math.min(otherItem.width, 10)
    height: calculateMyHeight()
    color: width > 10 ? "blue" : "red"
}
```

States

States are a mechanism to combine changes to properties in a semantic unit. A button for example has a pressed and a non-pressed state, an address book application could have a read-only and an edit state for contacts. Every element has an "implicit" base state. Every other state is described by listing the properties and values of those elements which differ from the base state.

Example: In the default state, myRect is positioned at 0,0. In the "moved" state, it is positioned at 50,50. Clicking within the mouse area changes the state from the default state to the "moved" state, thus moving the rectangle.

```
import QtQuick 2.0

Item {
    id: myItem
    width: 200; height: 200

    Rectangle {
        id: myRect
        width: 100; height: 100
        color: "red"
    }
    states: [
        State {
            name: "moved"
            PropertyChanges {
                target: myRect
                x: 50
                y: 50
            }
        }
    ]
    MouseArea {
        anchors.fill: parent
        onClicked: myItem.state = 'moved'
    }
}
```

State changes can be animated using Transitions.

For example, adding this code to the above Item element animates the transition to the "moved" state:

```
transitions: [
    Transition {
        from: "*"
        to: "moved"
        NumberAnimation { properties: "x,y"; duration: 500 }
    }
 ]
```

Animation

Animations in QML are done by animating properties of objects. Properties of type real, int, color, rect, point, size, and vector3d can all be animated.

QML supports three main forms of animation: basic property animation, transitions, and property behaviors.

The simplest form of animation is a PropertyAnimation, which can animate all of the property types listed above. A property animation can be specified as a value source using the Animation on property syntax. This is especially useful for repeating animations.

The following example creates a bouncing effect:

```
Rectangle {
    id: rect
    width: 120; height: 200

    Image {
        id: img
```

```
        source: "pics/qt.png"
        x: 60 - img.width/2
        y: 0

        SequentialAnimation on y {
            loops: Animation.Infinite
            NumberAnimation { to: 200 - img.height; easing.type: Easing.OutBounce; duration:
2000 }
            PauseAnimation { duration: 1000 }
            NumberAnimation { to: 0; easing.type: Easing.OutQuad; duration: 1000 }
        }
    }
}
```

Qt/C++ integration

Usage of QML does not require Qt/C++ knowledge to use, but it can be easily extended via Qt.[28][29] Any C++ class derived from QObject can be easily registered as a type which can then be instantiated in QML.

Familiar concepts

QML provides direct access to the following concepts from Qt:

- QObject signals – can trigger callbacks in JavaScript
- QObject slots – available as functions to call in JavaScript
- QObject properties – available as variables in JavaScript, and for bindings
- QWindow – Window creates a QML scene in a window
- Q*Model – used directly in data binding (e.g. QAbstractItemModel)[30][31][32]

Signal handlers

Signal handlers are JavaScript callbacks which allow imperative actions to be taken in response to an event. For instance, the MouseArea element has signal handlers to handle mouse press, release and click:

```
MouseArea {
    onPressed: console.log("mouse button pressed")
}
```

All signal handler names begin with "on".

Development tools

Because QML and JavaScript are very similar, almost all code editors supporting JavaScript will work. However full support for syntax highlighting, code completion, integrated help, and a WYSIWYG editor are available in the free cross-platform IDE Qt Creator since version 2.1 and many other IDEs.

The qml executable can be used to run a QML file as a script. If the QML file begins with a shebang it can be made directly executable. However packaging an application for deployment (especially on mobile platforms) generally involves writing a simple C++ launcher and packaging the necessary QML files as resources.

References

1. "Qt 5.15 Released" (https://www.qt.io/blog/qt-5.15-released).
2. "Which interface for a modern application?" (https://www.scriptol.com/ajax/ajax-xul-xaml.php). *scriptol*.
3. Ring Team (5 December 2017). "The Ring programming language and other languages" (http://ring-lang.sourceforge.net/doc1.6/introduction.html#ring-and-other-languages). *ring-lang.net*. ring-lang.
4. "Qt Declarative API Changes | Qt Blog" (https://web.archive.org/web/20140325005611/http://blog.qt.digia.com/blog/2009/08/21/qt-declarative-api-changes/#comment-4727). March 25, 2014. Archived from the original (http://blog.qt.digia.com/blog/2009/08/21/qt-declarative-api-changes/#comment-4727) on March 25, 2014.
5. "Qt 3D Overview | Qt 3D 5.13.1" (https://doc.qt.io/qt-5/qt3d-overview.html). *doc.qt.io*.
6. "All QML Types | Qt 5.13" (https://doc.qt.io/qt-5/qmltypes.html). *doc.qt.io*. Retrieved September 7, 2019.
7. Knoll, Lars (2013-04-15). "Evolution of the QML engine, part 1" (http://blog.qt.io/blog/2013/04/15/evolution-of-the-qml-engine-part-1/). Retrieved 2018-05-11.
8. "What's New in Qt 5.2" (http://doc.qt.io/qt-5/whatsnew52.html). Retrieved 2018-05-11.
9. "Qt Quick Compiler" (http://doc.qt.io/QtQuickCompiler/). Retrieved September 7, 2019.
10. "Deploying QML Applications | Qt 5.13" (https://doc.qt.io/qt-5/qtquick-deployment.html#qml-caching). *doc.qt.io*. Retrieved September 7, 2019.
11. "Development/Tutorials/Plasma4/QML/GettingStarted" (https://techbase.kde.org/Development/Tutorials/Plasma4/QML/GettingStarted). *KDE TechBase*. KDE.
12. Dragly, Svenn-Arne. "Developing for the reMarkable tablet" (https://dragly.org/2017/12/01/developing-for-the-remarkable/index.html). *dragly*.
13. "QML Demo for the reMarkable Paper Tablet" (https://github.com/reHackable/reHackable-HelloWorld). *GitHub*.
14. "Ubuntu's Unity Written In Qt/QML For "Unity Next" " (https://www.phoronix.com/scan.php?page=news_item&px=MTMxNzM). Michael Larabel.
15. "Combining C++ with QML in Sailfish OS applications" (https://sailfishos.org/develop/tutorials/combining-c-with-qml/).
16. "Tutorial - QML Live Coding With Qt QmlLive" (https://sailfishos.org/wiki/Tutorial_-_QML_Live_Coding_With_Qt_QmlLive).
17. "QML to C++ and C++ to QML" (https://together.jolla.com/question/86050/qml-to-c-and-c-to-qml/). Jolla.
18. "QML fundamentals" (http://developer.blackberry.com/native/documentation/dev/qml_fundamentals/index.html). Blackberry.
19. "Intro to QML for Meego" (https://slidedocument.com/intro-to-qml-meego). Nokia.
20. "MeeGo and Qt / QML demos assault MWC" (https://www.iotgadgets.com/2011/02/meego-qt-qml-demo-assaults-mwc-nokia-intel/). IoT Gadgets.
21. "QML on N900" (http://wiki.maemo.org/QML). *maemo.org*. Maemo Community.
22. "Qt Launches on Tizen with Standard Look and Feel" (http://qtfortizen.blogspot.no/2013/05/1.0alpha1.html).
23. "Mer" (http://www.merproject.org/).
24. "Mer wiki" (https://wiki.merproject.org/wiki/Main_Page).
25. "Lipstick QML UI on MeeGo CE / Mer" (https://www.iotgadgets.com/2011/10/lipstick-qml-ui-meego-ce-mer/). IoT Gadgets.
26. "QML - the best tool to unlock your creativity" (https://docs.ubuntu.com/phone/en/apps/qml/). Ubuntu.
27. "Looking at Lumina Desktop 2.0" (https://www.trueos.org/blog/looking-lumina-desktop-2-0/). TrueOS.

28. Alpert, Alan. "The Qt/QML User Story" (https://imaginitis.net/the-qtqml-user-story-redux/). *Incorrigible Imaginings*.
29. Alpert, Alan. "The many ways to unite QML and C++" (https://www.qtdeveloperdays.com/north-america/many-ways-unite-qml-and-c.html). *Qt Developer Days*. BlackBerry.
30. Dahlbom, J. "QAbstractItemModels in QML views" (https://jdahlbom.wordpress.com/2010/04/22/qabstractitemmodels-in-qml-views/). *The missing pieces*.
31. "Sorting and filtering a TableView" (https://blog.qt.io/blog/2014/04/16/qt-weekly-6-sorting-and-filtering-a-tableview/). The Qt Company.
32. Brad, van der Laan. "How to use Qt's QSortFilterProxyModel" (http://imaginativethinking.ca/use-qt-qsortfilterproxymodel/). *ImaginativeThinking*.

External links

- QML Reference Documentation (http://doc.qt.io/qt-5/qmlreference.html)
- First steps with QML (http://doc.qt.io/qt-5/qmlfirststeps.html)
- QML Examples and Tutorials (http://doc.qt.io/qt-5/qtquick-codesamples.html)
- Qt Blog (http://blog.qt.io/)
- QML Tutorial (http://doc.qt.io/qt-5/qml-tutorial.html)
- Qt Developer Guides (http://wiki.qt.io/Developer_Guides)
- Exporting QML from Photoshop and GIMP (http://blog.qt.io/blog/2010/10/19/exporting-qml-from-photoshop-and-gimp/)
- Application complete invoicing system in QML - Khitomer (https://www.youtube.com/watch?v=lZypEYdAdOw)
- QML Creator (https://play.google.com/store/apps/details?id=com.wearyinside.qmlcreator)
- QML Book (http://qmlbook.org/)

How-tos

- Integrating QML and C++ (http://doc.qt.io/qt-5/qtqml-cppintegration-topic.html)
- Important C++ Classes Provided By The Qt QML Module (http://doc.qt.io/qt-5/qtqml-cppclasses-topic.html)