

Qt Quick for Qt Developers

Introduction to Qt Quick



Based on Qt 5.4 (QtQuick 2.4)

Contents



- 30,000 feet Qt overview
- Meet Qt Quick
- Concepts

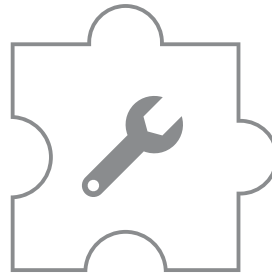
- Overview of the Qt library
 - Qt framework presentation
 - Qt Quick inside the Qt framework
- Understanding of QML syntax and concepts
 - Elements and identities
 - Properties and property binding
- Basic user interface composition skills
 - Familiarity with common elements
 - Understanding of anchors and their uses
 - Ability to reproduce a design

The Leading C++ Cross-Platform Framework



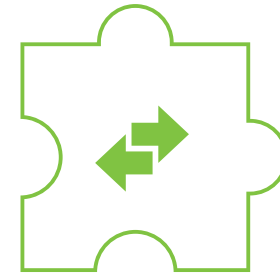
Cross-Platform Class Library

One Technology for
All Platforms



Integrated Development Tools

Shorter Time-to-Market



Cross-Platform IDE, Qt Creator

Productive development
environment

Used by over 800,000 developers in 70+ industries

Proved & tested technology – since 1994

Qt UI Offering – Choose the Best of All Worlds

Qt Quick

C++ on the back, declarative UI design (QML) in the front for beautiful, modern touch-based User Experiences.



Qt Widgets

Customizable C++ UI controls for traditional desktop look-and-feel. Also good for more static embedded UIs for more limited devices / operating systems.



Web / Hybrid

Use HTML5 for dynamic web documents, Qt Quick for native interaction.



The Widget World

File Edit View Go Bookmarks Help

Back Home Sync Find

Contents Index Bookmarks Search

Index

Look for:

2D Painting Example

40000 Chips

<qdrawutil.h> - Drawing Utility Functions

<QtAlgorithms> - Generic Algorithms

<QtConcurrentFilter> - Concurrent Filter and Filter...

<QtConcurrentMap> - Concurrent Map and Map-Re...

<QtConcurrentRun> - Asynchronous Run

<QtCore/qmath.h> - Math Functions

<QtEndian> - Endian Conversion Functions

<QtGlobal> - Global Qt Declarations

<QtPlugin> - Macros for Defining Plugins

_deviceType

_touchPoints

_touchPointStates

_widget

A Quick Start to Qt Designer

A Short Path to XQuery

A standard ActiveX and the "simple" ActiveQt widget

abort

aborted

abortEvaluation

abortHostLookup

Open Pages

Qt 4.8: 40000 Chips

- demos/chip/chip.h
- demos/chip/mainwindow.cpp
- demos/chip/mainwindow.h
- demos/chip/view.cpp
- demos/chip/view.h
- demos/chip/main.cpp
- demos/chip/chip.pro
- demos/chip/images.qrc

The 40000 Chips demo shows how to visualize a huge scene with 40000 chip items using Graphics View. It also shows Graphics View's powerful navigation and interaction features, allowing you to zoom and rotate each of four views independently, and you can select and move items around the scene.

Chip Demo

Top left view Antialiasing OpenGL

Top right view Antialiasing OpenGL

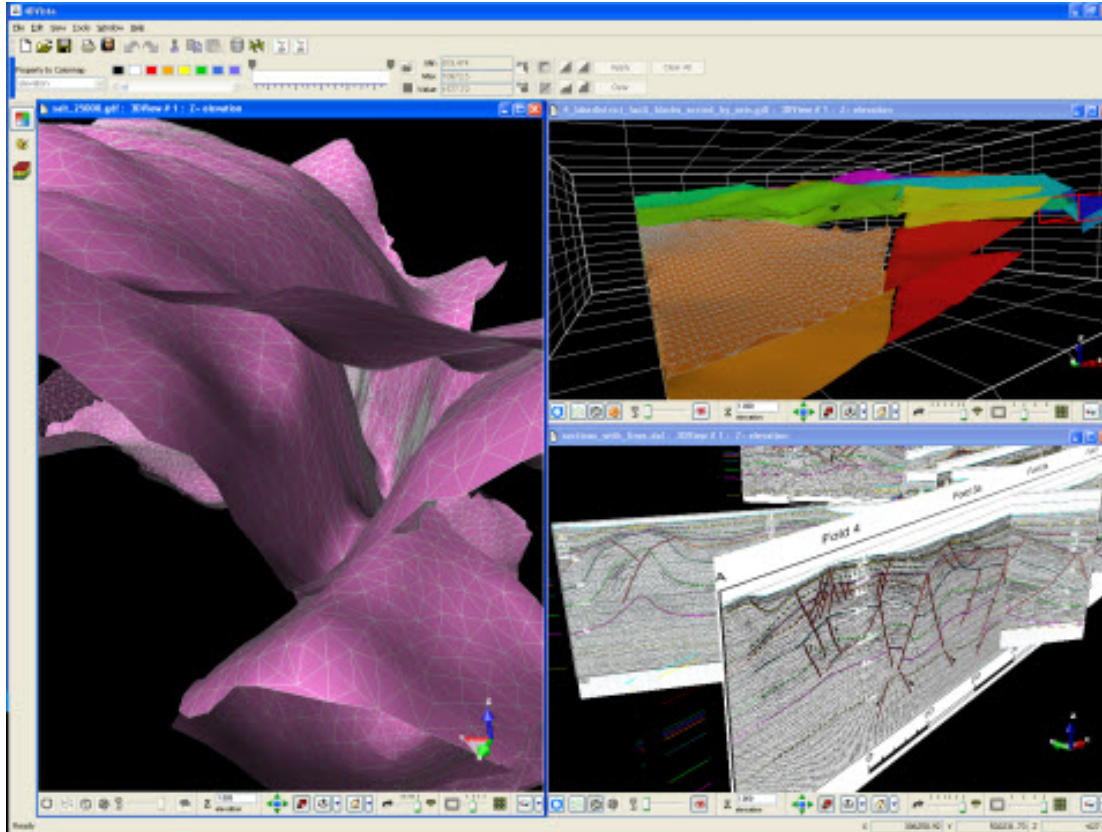
Bottom left view Antialiasing OpenGL

Bottom right view Antialiasing OpenGL

The Graphics View World



The OpenGL World







The Bolt

Published: 2008
Director: Byron Howard, Chris Williams
Cast: John Travolta, Miley Cyrus, Susie Essman

★★★★★☆☆☆☆ (7.1)

Description: Bolt, an American White Shepherd, has lived his whole life on the set of his action TV show, where he believes he has superpowers. When separated from the studio by accident, he meets a female alley cat named Mittens and a hamster named Rhino. He's trying to find the way home, to the studio. Along the way, he learns that he doesn't have superpowers and that the show is not real.

[Back](#)[Order](#)



The Cinematic Experience

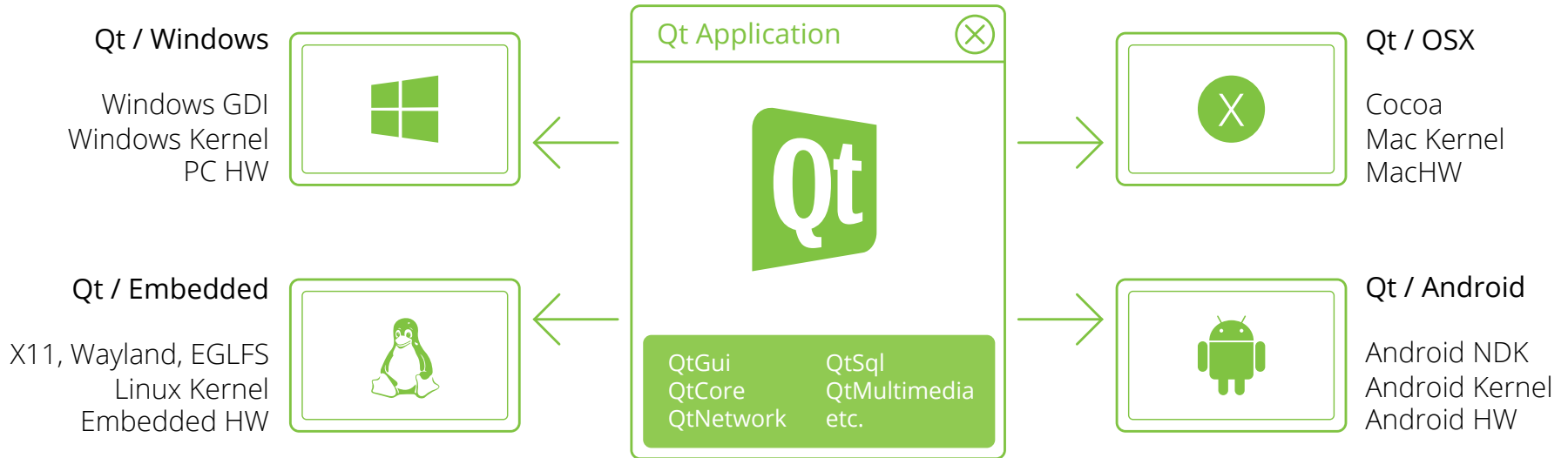
Welcome to 'Cinematic Experience' demo. This application demonstrates the power of Qt5 and few of the new additions available in QtQuick 2.0. Below is a short summary of those new features which have been used in this demo application.

Rendering
Qt5 has brand new rendering backend 'QML SceneGraph' which is optimized for hardware accelerated rendering. This allows to take full gains out of OpenGL powered GPUs on desktop and embedded devices. Not just performance, new Qt5 rendering backend also allows features which have not been possible earlier.

Particles
Qt5 comes with a fresh particles plugin 'QtQuick.Particles 2.0' which is superior compared to Qt4 particles. In this demo application, twinkling stars, shooting star and fog/smoke have been implemented using this new particles engine. Superb.



Qt Applications Are Native Applications



Qt Quick Requirements

- Platform must support OpenGL ES2
- Needs at least QtCore, QtGui, QtQml, and QtQuick modules
- Other modules can be used to add new features:
 - QtGraphicalEffects: add effects like blur, dropshadow...
 - Qt3D: 3D programming in QML
 - QtMultimedia: audio and video items, camera
 - QtWebEngine: web view
 - ...

The Qt framework is split into modules:

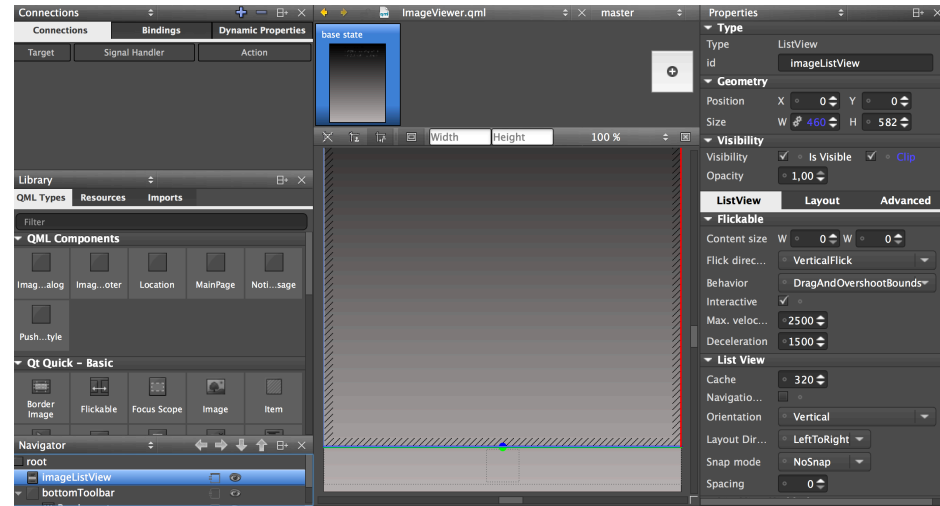
- Examples: QtCore, QtGui, QtWidgets, QtNetwork, QtMultimedia...
- Modules contain libraries, plugins and documentation.
- Libraries are linked to your applications
- Libraries group a set of common features (xml, dbus, network...)
- Qt Core is mandatory for all Qt applications

Meet Qt Quick

What is Qt Quick?

A set of technologies including:

- Declarative markup language: QML
- Imperative Language: JavaScript
- Language runtime integrated with Qt
- C++ API for integration with Qt applications
- QtCreator IDE support for the QML language



Philosophy of Qt Quick

- Intuitive User Interfaces
- Design-Oriented
- Rapid Prototyping and Production
- Easy Deployment
- Enable designer and developers to work on the same sources

Rapid Workflow with Qt Quick



Designer



Developer

Qt Quick

Declarative UI Design

Stunningly Fluent Modern User Interfaces, written with QML. Ideal for rapid UI prototyping.

Imperative Logic

Power of Cross-Platform Native Qt/C++

Core

Processes, Threads,
IPC, Containers,
I/O, Strings,
Etc.

Network

HTTP
FTP
SSL

Sql

SQL
&
Oracle
Databases

XML

Bluetooth

Positioning

NFC

Serial Port

+ Direct Hardware Access

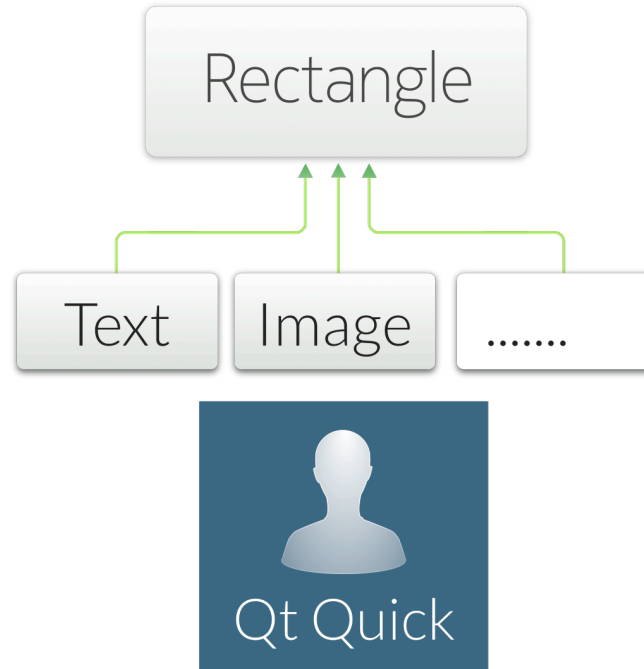
Concepts

What Is QML?

Declarative language for User Interface elements:

- Describes the user interface
 - What elements look like
 - How elements behave
- UI specified as tree of elements with properties

A Tree of Elements



- Let's start with an example...

Viewing an Example

```
import QtQuick 2.4

Rectangle {
    width: 400;
    height: 400
    color: "lightblue"
}
```

- Locate the example: `rectangle.qml`
- Launch the QML runtime:
`qmlscene rectangle.qml`

Demo: `qml-intro/ex-concepts/rectangle.qml`

- Elements are structures in the markup language
 - Represent visual and non-visual parts
- `Item` is the base type of visual elements
 - Not visible itself
 - Has a position, dimensions
 - Usually used to group visual elements
 - `Rectangle`, `Text`, `TextInput`,...
- Non-visual elements:
 - States, transitions,...
 - Models, paths,...
 - Gradients, timers, etc.
- Elements contain properties
 - Can also be extended with custom properties

See Documentation: [QML Elements](#)

- Elements are described by properties:
- Simple name-value definitions
 - `width, height, color,...`
 - With default values
 - Each has a well-defined type
 - Separated by semicolons or line breaks
 - Used for
 - Identifying elements (`id` property)
 - Customizing their appearance
 - Changing their behavior

Property Examples

- Standard properties can be given values:

```
Text {  
    text: "Hello world"  
    height: 50  
}
```

- Grouped properties keep related properties together:

```
Text {  
    font.family: "Helvetica"  
    font.pixelSize: 24  
    // Preferred syntax  
    // font { family: "Helvetica"; pixelSize: 24 }  
}
```

- Identity property gives the element a name:

- Identifying elements (`id` property)
- Customizing their appearance
- Changing their behavior

```
Text {  
    id: label  
    text: "Hello world"  
}
```

Property Examples

- Attached properties are applied to elements:

```
TextInput {  
    text: "Hello world"  
    KeyNavigation.tab: nextInput  
}
```

- `KeyNavigation.tab` is not a standard property of `TextInput`
- Is a standard property that is attached to elements

- Custom properties can be added to any element:

```
Rectangle {  
    property real mass: 100.0  
}  
  
Circle {  
    property real radius: 50.0  
}
```


Binding Properties

```
Item {  
    width: 400; height: 200  
    Rectangle {  
        x: 100; y: 50; width: height * 2; height: 100  
        color: "lightblue"  
    }  
}
```

- Properties can contain expressions
 - See above: `width` is twice the `height`
- Not just initial assignments
- Expressions are re-evaluated when needed



Demo: [qml-intro/ex-concepts/expressions.qml](#)

Identifying Elements

The `id` property defines an identity for an element

- Lets other elements refer to it
 - For relative alignment and positioning
 - To use its properties
 - To change its properties (e.g., for animation)
 - For re-use of common elements (e.g., gradients, images)
- Used to *create relationships* between elements

See Documentation: [Property Binding](#)

Using Identities

```
Item {  
    width: 300; height: 115  
    Text {  
        id: title  
        x: 50; y: 25 text: "Qt Quick"  
        font.family: "Helvetica"; font.pixelSize: 50  
    }  
    Rectangle {  
        x: 50; y: 75; height: 5  
        width: title.width  
        color: "green"  
    }  
}
```



Qt Quick

Demo: [qml-intro/ex-concepts/identity.qml](#)

Viewing an Example

```
Text {  
    id: title  
    x: 50; y: 25 text: "Qt Quick"  
    font.family: "Helvetica"; font.pixelSize: 50  
}  
  
Rectangle {  
    x: 50; y: 75; height: 5  
    width: title.width  
    color: "green"  
}
```

The text "Qt Quick" is displayed in a large, bold, black serif font. Below the text is a thick, solid green horizontal line.

- Property `Text` element has the identity, `title`
- Property `width` of `Rectangle` bound to `width` of `title`

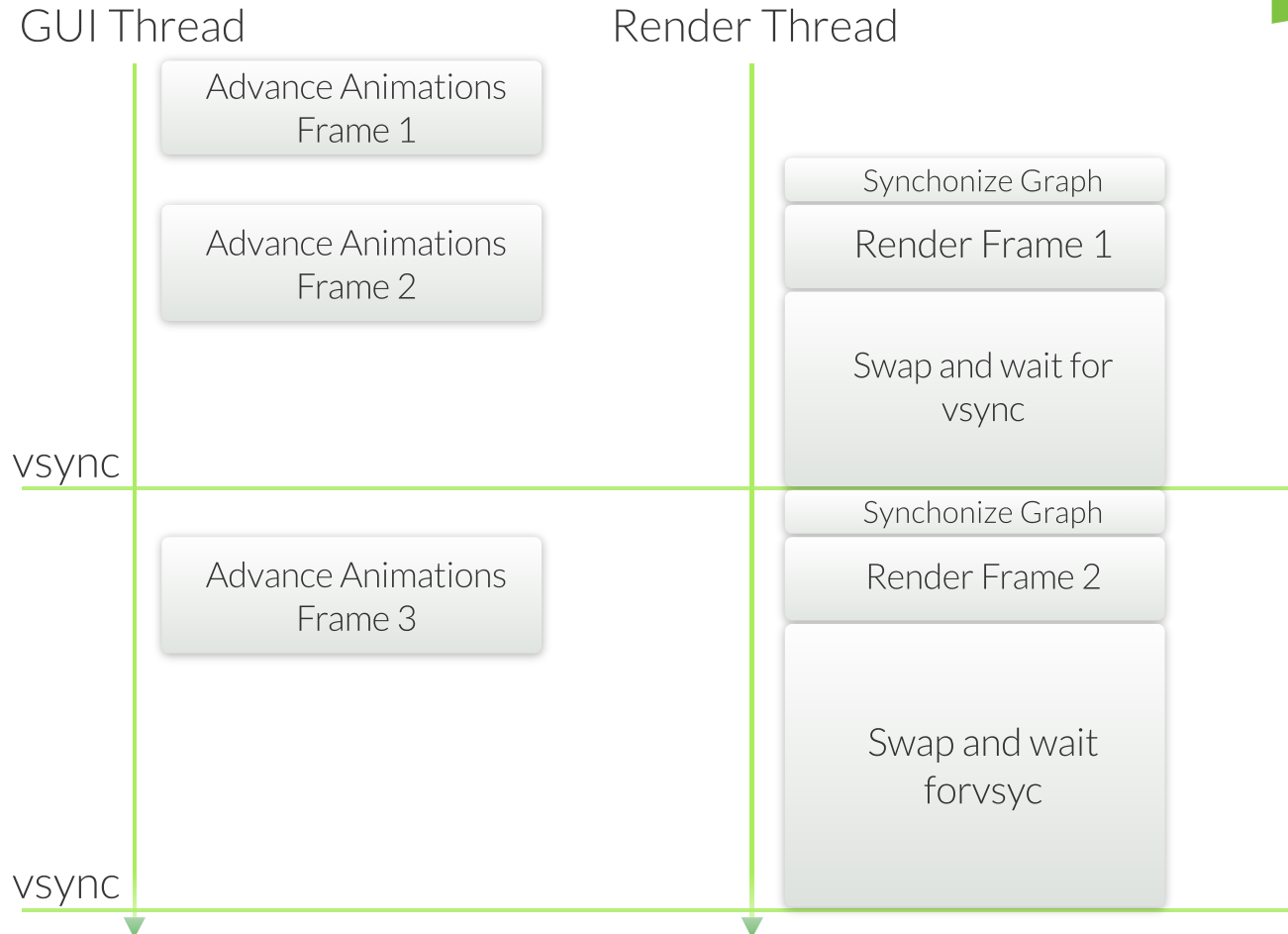
Property values can have different types:

- Numbers (int and real): 400 and 1.5
- Boolean values: `true` and `false`
- Strings: `"HelloQt"`
- Constants: `AlignLeft`

- Lists:[...]
 - One item lists do not need brackets
- Scripts:
 - Included directly in property definitions
- Other types:
 - colors, dates, rects, sizes, 3Dvectors,...
 - Usually created using constructors

See Documentation: [QML Types](#)

Behind the Scene



- QML defines user interfaces using elements and properties
 - Elements are the structures in QML source code
 - Items are visual elements
- Standard elements contain properties and methods
 - Properties can be changed from their default values
 - Property values can be expressions
 - `Id` properties give identities to elements
- Properties are bound together
 - When a property changes, the properties that reference it are updated
- Some standard elements define methods
- A range of built-in types is provided

Questions

- How do you load a QML module?
- What is the difference between `Rectangle` and `width`?
- How would you create an element with an identity?
- What syntax do you use to refer to a property of another element?

The image on the right shows two items and two child items inside a 400×400 rectangle.

1. Recreate the scene using Rectangle items.
2. Can items overlap? Experiment by moving the light blue or green rectangles.
3. Can child items be displayed outside their parents? Experiment by giving one of the child items negative coordinates.

