Qt Quick for Qt Developers

Qt Essentials - Training Course

Produced by Nokia, Qt Development Frameworks

Material based on Qt 4.7, created on January 18, 2011

http://qt.nokia.com

Module: Introduction to Qt Quick

- Meet Qt Quick
- Concepts

Objectives

- 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

Module: Introduction to Qt Quick

- Meet Qt Quick
- Concepts

What is Qt Quick?

A set of technologies including:

- Declarative markup language: QML
- Language runtime integrated with Qt
- Qt Creator IDE support for the QML language
- Graphical design tool
- C++ API for integration with Qt applications

Philosophy of Qt Quick

- Intuitive User Interfaces
- Design-Oriented
- Rapid Prototyping and Production
- Easy Deployment

Design Criteria: Intuitive User Interfaces

- More natural ways to interact with applications
- Easier to predict what elements will do
- Smooth motion like real-world objects

Design Criteria: Design-Oriented

Traditional Qt way:

- Design tools for developers
- Rich, platform-native widgets

The Qt Quick way:

- Development tools and processes that are accessible to designers
- Lightweight, customizable elements

Design Criteria: Rapid Prototyping and Production

- No compilation step
- JavaScript used as scripting language
  - See *JavaScript: The Definitive Guide*
  - See https://developer.mozilla.org/en/JavaScript
- Helpful to understand:
  - HTML, CSS
  - ... but not required
- Requires little or no programming experience

Design Criteria: Easy Deployment

- Self-contained packages
- Installation is optional
- Network transparent
  - allows deployment over a network
  - rich client front end to online services

Module: Introduction to Qt Quick

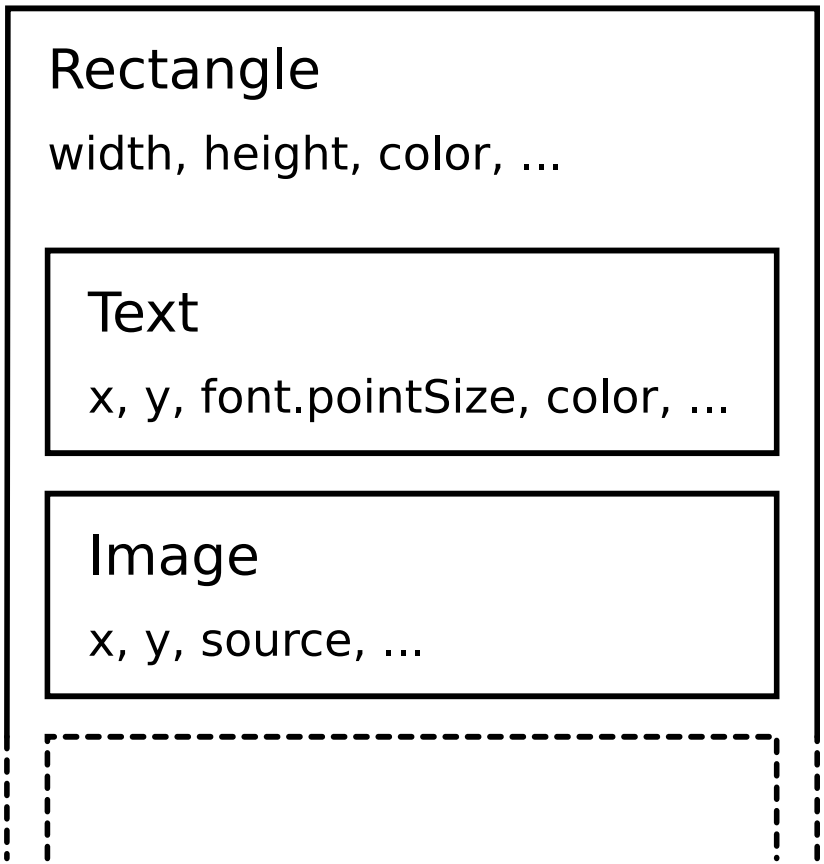
- Meet Qt Quick
- Concepts

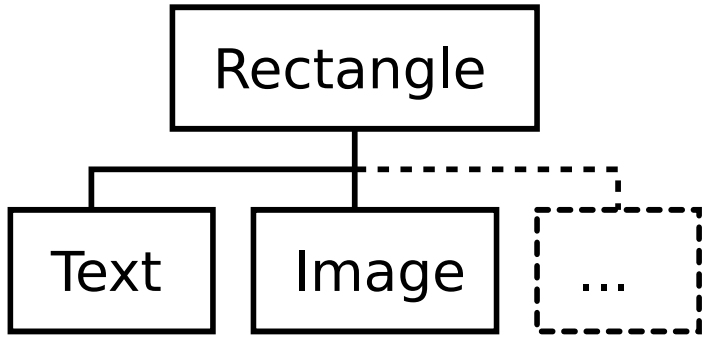
What is QML?

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 1.0

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

- Locate the example: `rectangle.qml`
- Launch the QML viewer:

```
qmlviewer rectangle.qml
```

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

Example Concepts

```
import QtQuick 1.0

// Define a light blue square

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

- To use Qt's features, `import` the `Qt` module
- Specify a version to get the features you want
  - only imports the features from that version
  - will not use features from later versions, even if available
  - will not fall back to features from an earlier version
- Guarantees that the behavior of the code will not change
  - modules include support for multiple versions
  - an upgraded module retains support for older versions of itself

Example Concepts

```
import QtQuick 1.0

// Define a light blue square

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

- Use `//` to add single line comments
- Put multi-line comments inside `/*` and `*/`

Example Concepts

```
import QtQuick 1.0

// Define a light blue square

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

- Declare the elements you want to use
- Each element has a body between { and }
- A set of default elements are included in the `Qt` module

Example Concepts

```
import QtQuick 1.0

// Define a light blue square

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

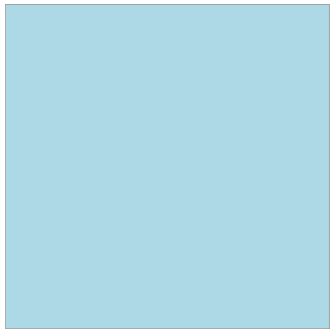
- Elements contain properties
- Each property is defined using its name and a value
- `name : value`

Example Summary

```
import QtQuick 1.0

// Define a light blue square

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



- A `Rectangle` element
  - with a body: `{ ... }`
  - containing `width`, `height` and `color` properties
  - separated by line breaks or semicolons
- Running the example in the `qml` viewer
  - the viewer window will be 400 by 400

Elements

- Elements are structures in the markup language
  - represent visible and non-visible parts
- `Item` is the base type of visible elements
  - not visible itself
  - has a position, dimensions
  - usually used to group visual elements
  - often used as the top-level element
  - `Rectangle`, `Text`, `TextInput`, ...
- Non-visible elements:
  - states, transitions, ...
  - models, paths, ...
  - gradients, timers, etc.
- Elements contain properties
  - can also be extended with custom properties

See QML Elements Documentation

Properties

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
}
```

- **Identity property** gives the element a name:

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

Property Examples

- **Attached properties** are applied to elements:

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

  - `KeyNagivation.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

```
import QtQuick 1.0

Item {
    width: 400; height: 200

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


Demo qml-intro/ex-concepts/expressions.qml

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

See Property Binding Documentation

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

Using Identities

```
import QtQuick 1.0

Item {
    width: 300; height: 115

    Text {
        id: text_element
        x: 50; y: 25
        text: "Qt Quick"
        font.family: "Helvetica"; font.pixelSize: 50
    }

    Rectangle {
        x: 50; y: 75; height: 5
        width: text_element.width
        color: "green"
    }
}
```

Demo qml-intro/ex-concepts/identity.qml

Using Identities

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

- `Text` element has the identity, `textElement`
- `width` of `Rectangle` bound to `width` of `textElement`
- Try using `TextInput` instead of `Text`

Methods

- Most features are accessed via properties
- Some actions cannot be exposed as properties
- Elements have methods to perform actions:
  - `TextInput` has a `selectAll()` method
  - `Timer` has `start()`, `stop()` and `restart()` methods
  - `Particles` has a `burst()` method
- All methods are public in QML
- Other methods are used to convert values between types:
  - `Qt.formatDateTime(datetime, format)`
  - `Qt.md5(data)`
  - `Qt.tint(baseColor, tintColor)`

Basic Types

Property values can have different types:

- Numbers (int and real): `400` and `1.5`
- Boolean values: `true` and `false`
- Strings: `"Hello Qt"`
- Constants: `AlignLeft`
- Lists: `[ ... ]`
  - lists with one item can be written as just the item itself
- Scripts:
  - included directly in property definitions
- Other types:
  - colors, dates, times, rects, points, sizes, 3D vectors, ...
  - usually created using constructors

See QML Types Documentation

Summary

- QML defines user interfaces using elements and properties
  - elements are the structures in QML source code
  - items are visible 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

Exercise

1. How do you request features from a certain version of Qt?
2. What is the difference between `Rectangle` and `width`?
3. How would you create an element with an identity?
4. What syntax do you use to refer to a property of another element?