Qt Quick for Qt Developers

Training Course

Visit us at `http://qt.digia.com`

Produced by Digia Plc.

Material based on Qt 5.0, created on September 27, 2012

Digia Plc.

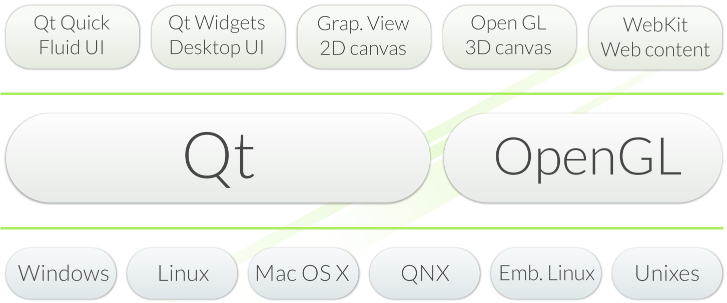
Module: Introduction to Qt Quick

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

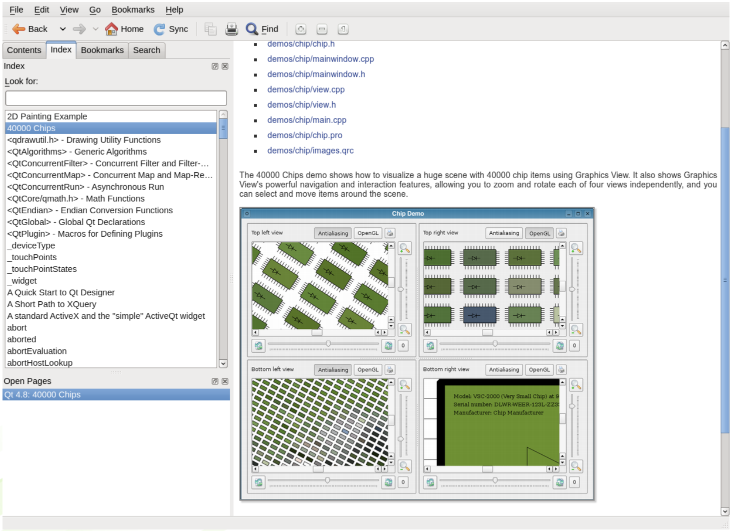
Objectives

- 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

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



30,000 feet Qt overview



- 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

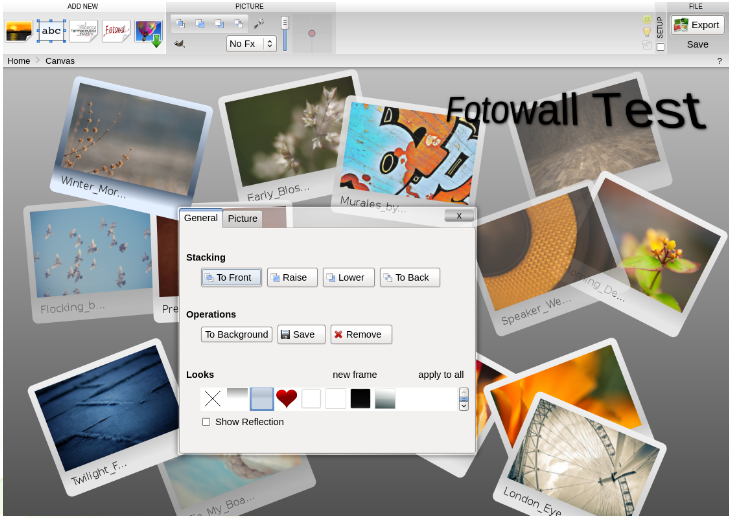
Qt 4.8: 40000 Chips

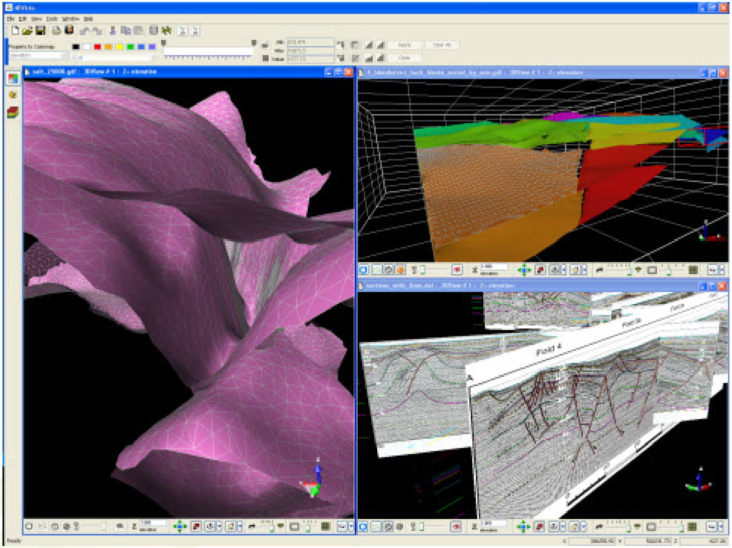
- demos/chip/chip.h
- demos/chip/mainwindow.cpp
- demos/chip/mainwindow.h
- demos/chip/view.cpp
- demos/chip/view.h
- demos/chip/main.cpp
- demos/chip/chip.pro
- demos/chip/images.qrc

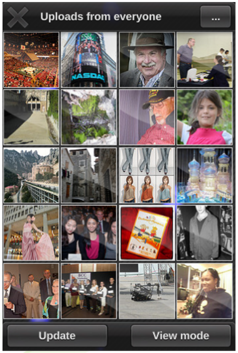
The 40000 Chips demo shows how to visualize a huge scene with 40000 chip items using Graphics View. It also shows Graphics View's powerful navigation and interaction features, allowing you to zoom and rotate each of four views independently, and you can select and move items around the scene.

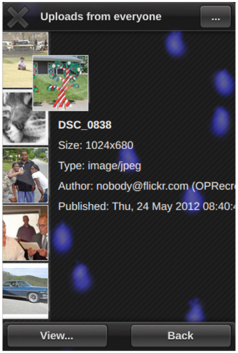
Model: VSC-2000 (Very Small Chip) at 9
Serial number: DLWR-WEER-123L-ZZ3
Manufacturer: Chip Manufacturer

The Graphics View World

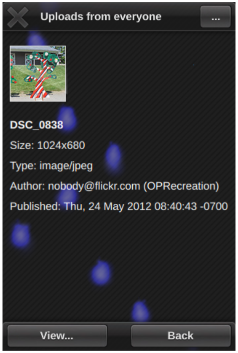



Fold 4





DSC_0838
Size: 1024x680
Type: image/jpeg
Author: nobody@flickr.com (OPRecr
Published: Thu, 24 May 2012 08:40:4



DSC_0838
Size: 1024x680
Type: image/jpeg
Author: nobody@flickr.com (OPRecreation)
Published: Thu, 24 May 2012 08:40:43 -0700

Qt Quick requirement

- Platform must support OpenGL ES2
- Needs at least QtCore, QtGui, QtV8 and QtDeclarative modules
- Other module can be used to add new features:
  - QtGraphicalEffects: add effects like blur, drop shadow...
  - Qt3D: 3D programming in QML
  - QtMultimedia: audio and video items
  - ...

Qt modules

The Qt framework is split into modules:

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

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

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
- Qt Creator IDE support for the QML language
- Graphical design tool

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

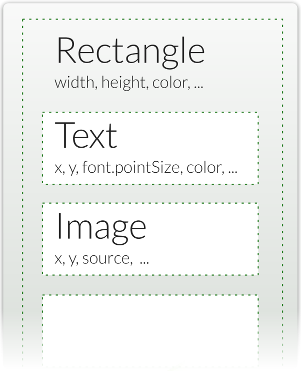
- 30,000 feet Qt overview
- Meet Qt Quick
- Concepts

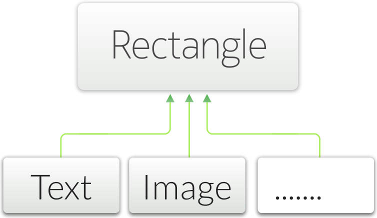
What is QML?

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 2.0

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

- 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
  - often used as the top-level element
  - `Rectangle`, `Text`, `TextInput`, ...
- Non-visual 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 2.0

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 re-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

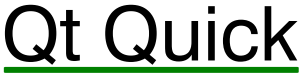
```
import QtQuick 2.0

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"
    }
}
```

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



Using Identities

```
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"

}
```

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

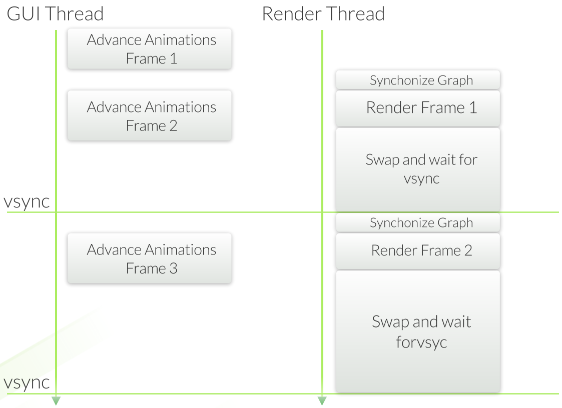
Basic Types

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

Behind the scene



Summary

- 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 And Answers

- 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?

Questions And Answers

- 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?

Questions And Answers

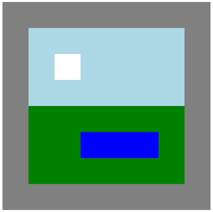
- How do you load a QML module?
- What is the difference between `Rectangle` and `width`?
- How would you create an element with an identity?
- What syntax do you use to refer to a property of another element?

Questions And Answers

- 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?

Exercise – Items

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.