ICS

QML Programming Fundamentals and Beyond

Introduction to QML

Material based on Qt 5.12

Course Outline

Session 1: April 28, Introduction to QML
- About QML
- Properties
- Basic Types

Session 2: May 1, QML Item Placement
- How to correctly size and place items
- When to use Anchors, Layouts and Positioners

Session 3: May 5, Touch Interaction
- QML Signals
- Touch Events
- Single and Multi-Touch
- Swipe and Pinch Gestures

Session 4: May 8, States & Transitions
- Creating and defining states
- Using Transitions

Session 5: May 15, Custom Items & Components
- Creating your own Components
- Creating a Module

Session 6: May 19, Model / View
- Model / View
- QML Models
- QML Views

Session 7: May 22, C++ Integration
- Why expose C++ to QML
- Exposing C++ Objects
- Exposing C++ Classes

About ICS

ICS Designs User Experiences and Develops Software for Connected Devices

- Largest source of independent Qt expertise in North America since 2002
- Headquartered in Waltham, MA with offices in California, Canada, Europe
- Includes Boston UX, ICS' UX design division

- Embedded, touchscreen, mobile and desktop applications
- Exclusive Open Enrollment Training Partner in North America

Qt Service Partner

BOSTON UX

UX/UI Design and Development for Connected Devices Across Many Industries

POLL QUESTIONS

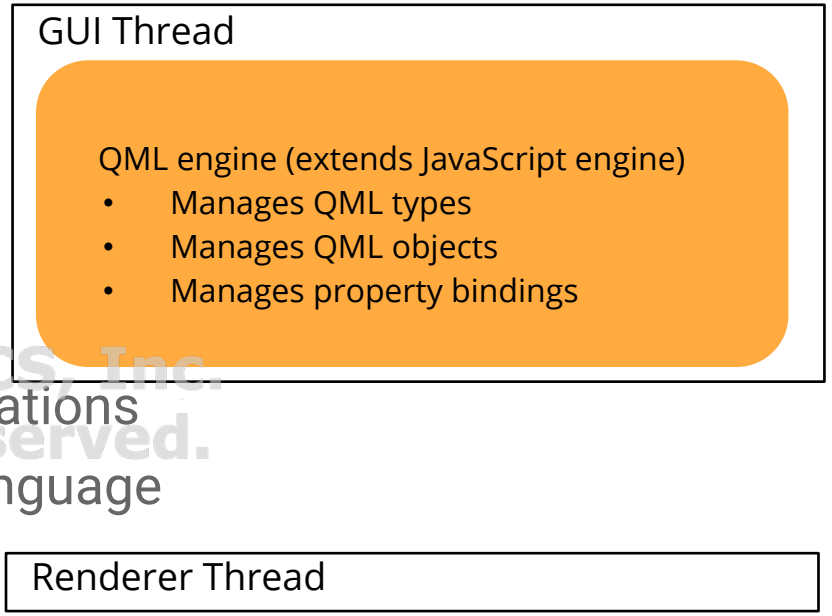
Agenda

- What is Qt Quick?
- What is QML and how it is structured:
  - QML Properties
  - QML Property Binding
  - QML Types
- Demo: Simple QML Project
- Questions ?

What is Qt Quick?

A set of technologies including:

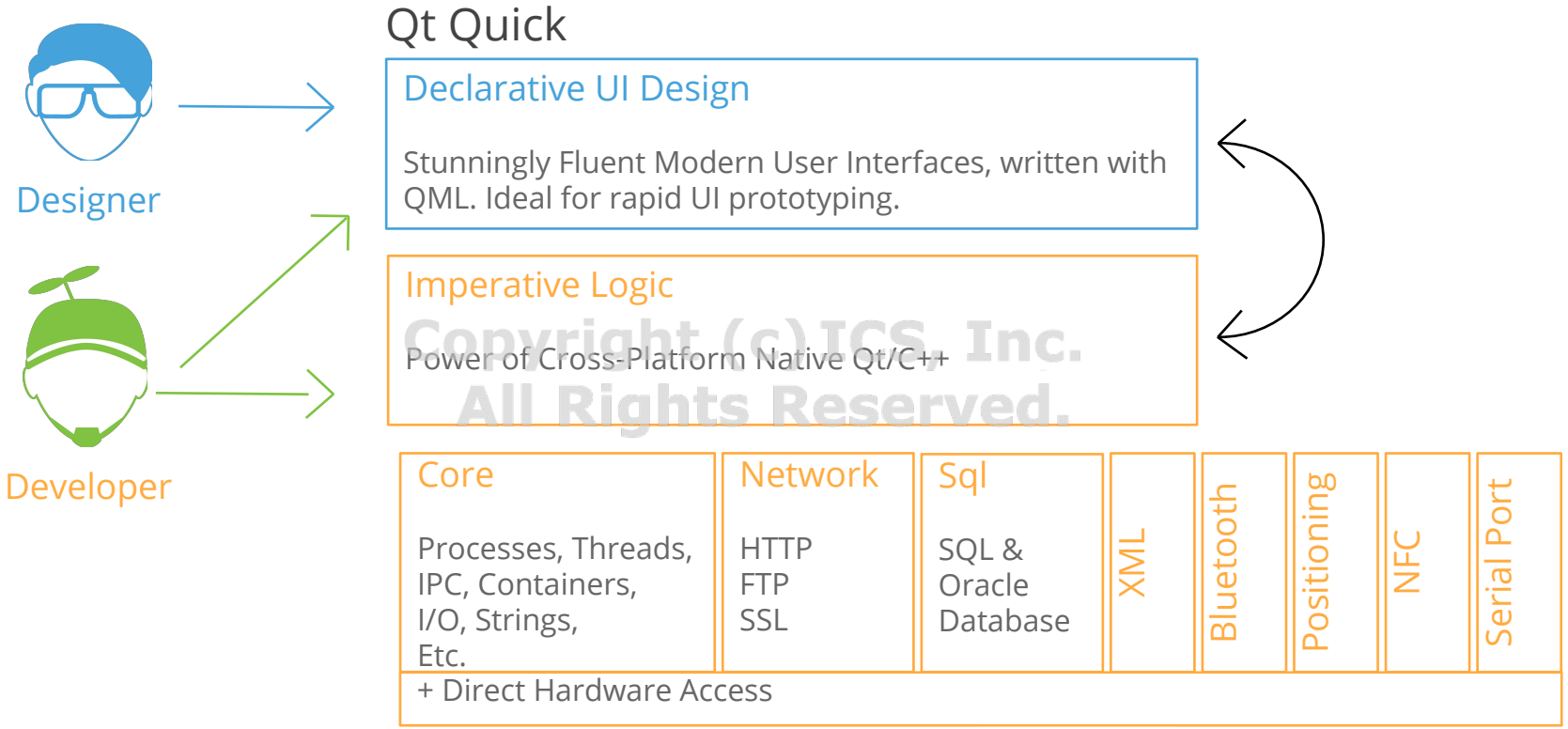
- Declarative markup language: QML
- Scripting Language: JavaScript
- Language runtime integrated with Qt
- C++ API for integration with Qt applications
- QtCreator IDE support for the QML language
  - Qt Quick Designer
  - Debugger
  - QML Profiler



Philosophy of Qt Quick

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

Rapid Workflow with Qt Quick



What Is QML?

Declarative language for User Interface building blocks

- Describes the user interface
  - What UI building blocks look like
  - How they behave
- UI specified as tree of QML objects with properties

Example file: MyRectangle.qml

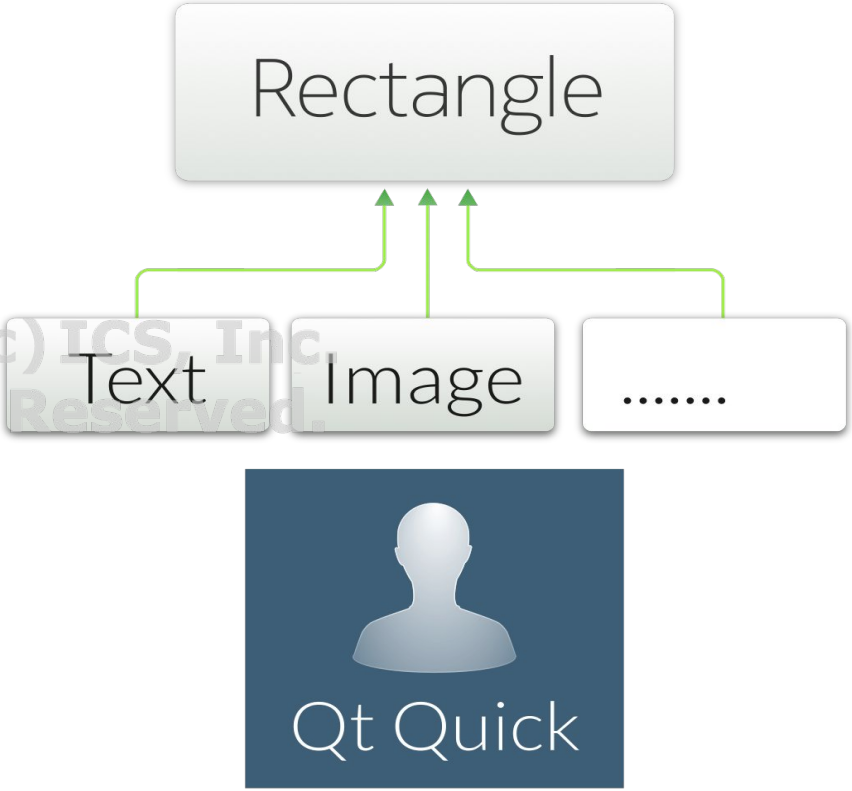
```
import QtQuick 2.12 // Loads version 2.12 of the QtQuick module

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

QML File Structure

- Root Item type
- Identifier
- Property declarations
- Signal declarations
- JavaScript functions
- Object properties
- Child objects
- States
- Transitions

```
import ModuleName VersionNumber
//Import modules as needed.
Item {
    id: exampleItem
    property var exampleProperty:42
    signal exampleSignal(var variantArgument)
    function example() { return 0; }
    width: background.width
    height: background.height
    Rectangle {
        id: background
        …
    }
    states: [ State { … }, … ]
    transitions: [ Transition { … }, … ]
}
```

A Tree of QML Objects



QML Types

- QML types are structures in the markup language
  - Represent visual and non-visual parts
- `Item` is the base type of visual types
  - Not visible itself
  - Has a position, dimensions, focus
  - Supports layering
  - Usually used to group visual types
  - `Rectangle`, `Text`, `TextInput`,...
- Non-visual QML types:
  - States, transitions, Models, paths, gradients, timers,...
- QML types contain properties
  - Extendable with custom properties

Properties

QML types 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 QML objects (id property)
  - Customizing their appearance
  - Changing their behavior

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

Property Examples

- Standard properties can be given values:

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


- Grouped properties keep related properties together: Use either below.

```
Text {
    font {
        family: "Helvetica"
        pointSize: 24
    }
}
```

```
Text {
    font.family: "Helvetica"
    font.pointSize: 24
}
```

Property Examples

- Attached properties are applied to QML objects without object creation:

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

- **KeyNavigation.tab** is not a standard property of **TextInput**. It is a property that is attached to objects

- Custom properties can be added to any object:

```
Rectangle {
    property real mass: 100.0
}

CircleButton {
    property alias bgColor: itemA.color
}
```

Binding

```
Window {
    visible: true; width: 400; height: 200
    Rectangle {
        x: 100; y: 50;
        width: height * 2
        height: parent.height / 2
        color: "lightblue"
    }
}
```

- Properties can contain JavaScript expressions
  - See above: `width` is twice the `height`
- Not just for initial assignments, expressions are re-evaluated when needed
- **Note!** The JavaScript assignment operator '=' is not a binding
  - Assignment: `width = height * 2 // No re-evaluation`
  - Assignment to a binding: `width = Qt.binding(function() { return height * 2; } )`

Identifying Objects

```
Text {
    id: title
    x: 50; y: 25
    text: "Qt Quick"
    font { family: "Helvetica"; pointSize: parent.width * 0.1 }
}

Rectangle {
    x: title.x
    y: title.y + title.height - height;
    height: 5
    width: title.width
    color: "green"
}
```

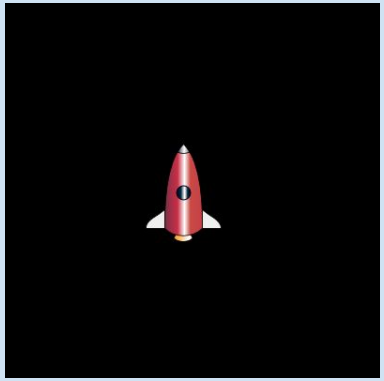

- **Text** item has the **id** `title`
- Properties **width**, **x**, **y** of **Rectangle** bound to the properties **x**, **y**, and **height** of `title` as shown.

Colors

```
Rectangle {
    id: redRect
    x: 0; y: 0;
    width: parent.width / 3; height: parent.height
    color: "#ff0000"
}
Rectangle {
    id: greenRect
    x: redRect.width; width: parent.width / 3; height: parent.height
    color: Qt.rgba(0,0.75,0,1)
}
Rectangle {
    x: redRect.width + greenRect.width;
    width: parent.width / 3;
    height: parent.height;
    color: "blue"
    opacity: 0.5
}
```

Images

```
Rectangle {
    width: 400; height: 400
    color: "black"
    Image {
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        source: "../images/rocket.png"
    }
}
```

- Property **source** contains a relative path
- Properties **width** and **height** are obtained from the image file
- Some Additional properties include **rotation**, **opacity**, **scale**

Text Type

```
Rectangle {
    width: 400; height: 400; color: "lightblue"
    Text {
        x: parent.width * 0.25; y: parent.height * 0.25
        text: qsTr("Qt Quick”)
        font { family: "Helvetica";
               pixelSize: parent.width * 0.1 }
    }
}
// fontSizeMode property is another way to do scaling
```


Qt Quick

- Can also use HTML tags in the text property:
  - `"<html><b>Qt Quick</b></html>"`
- Width and height can also be determined by the font metrics and text
- The rectangle's size could depend on the font size
  - `FontMetrics { id: metrics: font.family: “Courier” }`
  - `Rectangle { height: metrics.height * nofRows }`


TIME FOR A LIVE DEMO
38
Copyright (c) ICS, Inc.
All Rights Reserved.
WHAT COULD GO WRONG?
memegenerator.net

Q&A Session

If you have additional questions or feedback, please contact us at QtTraining@ics.com