ICS

QML Programming Fundamentals and Beyond

Custom Items and Components

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

Agenda

- Custom QML Items
- Signals & Handlers
- Alias Properties

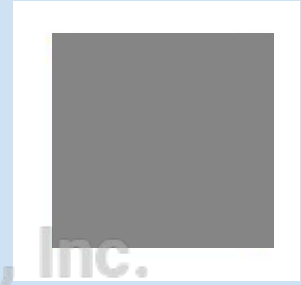
Custom Items and Components

Two ways to create reusable user interface components:

- Custom items
  - Defined in separate files
  - One main item per file
  - Used in the same way as standard items
  - Can have an associated version number
- Components
  - Used with models and views
  - Used with generated content
  - Defined using the `Component` item
  - Used as templates for items

Defining a Custom Item

```
Rectangle {
    color: mouseArea.pressed ? "lightgrey" : "grey"
    width: 100; height: 100
    MouseArea {
        id: mouseArea
        anchors.fill: parent
    }
}
```



A simple button

- A **Rectangle** with a **MouseArea**
- Stored in file named **SimpleButton.qml**

QML Structures: Using a Custom Item

```
Window {
   visible: true; width: 640; height: 480
   SimpleButton {
      anchors.centerIn: parent
      height: parent.height / 3
      width: parent.width / 3
   }
}
```


- **SimpleButton.qml** is in the same directory as the **main.qml**
- Item within the created file is automatically available as **SimpleButton**

Recap - Adding Custom Properties

- Create a custom property
  - Syntax:

property <type> <name>[: <value>]

```
property string product: "Qt Quick"
property int count: 123
property real slope: 123.456
property bool condition: true

// Read-only property
readonly property url address: "http://qt.io/"
```


Documentation: QML Object Attributes

Adding Custom Properties

The shown button lacks any text, after adding a **Text** item we can display the text.
To set/get the text outside of the **SimpleButton** object we must add a property.

```
Rectangle {
    property string buttonLabelText: buttonLabel.text
    …
    Text {
        id: buttonLabel
        …
        text: "Button text"
    }
}
```


- The custom property, **`buttonLabelText`**, *binds to* **`buttonLabel.text`**
- Setting a value to the custom property
  - Changes the binding
  - It no longer refers to **`buttonLabel.text`**

Using Alias Properties

If a custom property was created for binding to the property of a child item, it is recommended to use the property alias instead :

```
Rectangle {
    property alias buttonLabelText: buttonLabel.text
    …
    Text {
        id: buttonLabel
        …
        text: "Button"
    }
}
```


- Custom property, **buttonLabelText**, *binds to* **buttonLabel.text**
- Setting a value to the property changes the text of **buttonLabel**

Using Alias Properties

```
Window {
   visible: true; width: 640; height: 480
   SimpleButton {
      anchors.centerIn: parent
      height: parent.height / 3
      width: parent.width / 3
      buttonLabelText: "some alternate text"
   }
}
```

Adding Custom Signals

- Standard items define signals and handlers
  - e.g., **MouseArea** items can use **onClicked**
- Signal syntax: **signal <Name> [(<type> <value>, ...)]**
- Handler syntax: **on<Signal Name>: <expression>**
- Examples of signals and handlers:
  - signal **clicked**()
    - Handled by **onClicked**
  - signal **selectedRow**(**int index, int tableId**)
    - Handled by **onSelectedRow**
    - Argument passed as **index** and **tableId**
- Custom items can also define their own signals

Making and Emitting Custom Signals

- Assign an **id** for your root item to make it easy to call the signals internally
- Define with : signal <name> (<var_type> <var_name>)
- Signal names can not begin with an uppercase character
- Emit the signal by calling it.

```
Rectangle {
   id: buttonRoot
// With a return :
   signal clicked(bool checked)
   property bool isSelected: false

// Without a return :
   signal selected()
   ...
   MouseArea {
      ...
      onClicked: {
            isSelected =! isSelected
            buttonRoot.clicked(isSelected)
            buttonRoot.selected()
}}}
```

Receiving Signals

Rules:

- Follows the format **on<SignalName>**
- on* handlers are automatically created for custom signals
- The values supplied using the name defined in the signal (checked)

Signal without a return

```
...// signal clicked()
 SimpleButton {
      text: “click me”
      onClicked: {
          console.log(“clicked”)
 }}}
```

Signal with a return

```
... // signal clicked(bool checked)
 SimpleButton {
      text: “click me”
      onClicked: {
          console.log(“checked:” + checked)
 }}}
```

Connections

Items can be Connected to other items in Qml using a Connections Item

- This allows you to connect the parent of the Connections Item to a target items signal.
- In the example below, when myButton is clicked it's parent will log to the console.

```
...
    SimpleButton {
        id: myButton
        text: "click me"
        onClicked: someFunction()
    }

    Connections{
        target: mybutton
        onClicked: console.log("My button clicked")
    }
}
```

QML Structures: Collections of Items

```
import "items"

Window {
    ...
    SimpleButton { ... }
    NewButton { ... }
}
```


- Import “items” directory
- Includes all the files (e.g. items/SimpleButton.qml)
- Useful to organize your application

Importing into a Namespace

```
import "items" as MyItems
Rectangle {
    width: 250; height: 100; color: "lightblue"
    MyItems.SimpleButton {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
    }
}
```


- Importing a collection of items from a path
- Avoids potential naming clashes with items from other collections and modules


TIME FOR A LIVE DEMO
38
© Integrated Computer Solutions, 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

COMING SOON!

Hands-on Virtual Training:
Building an Embedded Device Application with Qt

Course begins July 14

More details and registration available early June