Qt Quick examples for porting

This article features a collection of code snippets demonstrating various QML functionality. Effects and UI elements makes it easy to port highly branded UIs from other platforms, such as iPhone and Android.

23 Jan 2011

For complete QML application examples, see:

- QML Diner App
- QML RSS Reader

The code snippets should work with latest versions of Qt 4.7 (http://qt.gitorious.org/qt).

Basic Use Cases

Create a button

To create a clickable button you need to

- Create a `BorderImage` with the graphics you want for your button
- Add a `Text` field to show text inside the button
- Add a property to `BorderImage` (which acts as your button component) to access the text inside the text field
- Add a `MouseArea` inside it that fills the `BorderImage`
- Add a handler to `MouseArea`'s `onClicked` event.

```
function onMyButtonClicked() {
    console.log("onMyButtonClicked");
}

BorderImage {
    id: myButton
    width: 360
    height: 40
    source: "item.sci"
    property alias buttonText: myButtonText.text

    MouseArea {
        anchors.fill: parent
        onClicked: onMyButtonClicked();
    }

    Text {
        id: myButtonText
        text: "myButton"
        anchors.centerIn: parent
        font.pointSize: 16
        color: "white"
    }
}
```

Create a list

Lists are created using models and views. The model is typically provided by a `QAbstractListModel` "C++ model object", but can also be created directly in QML.

- First you need a model that contains the data to be shown on the list

```
import Qt 4.7

Rectangle {
    width: 360
    height: 640

    ListModel {
        id: myListModel
```


(C) Copyright Nokia 2014. All rights reserved.

```
        ListElement {
            item_index: 0
            item_name: "Item 1"
            item_description: "Description 1"
        }
        ListElement {
            item_index: 1
            item_name: "Item 2"
            item_description: "Description 2"
        }
        ListElement {
            item_index: 2
            item_name: "Item 3"
            item_description: "Description 3"
        }
    }
```

- Then you need a delegate that handles the drawing of each item in the list. Separate delegate is used for the current highlighted item if needed.

```
    Component {
        id: myListDelegate
        Item {
            id: wrapper
            width: parent.width
            height: 40
            Column {
                x: 5; y: 5
                Text { text: '<b>Name:</b> ' + item_name }
                Text { text: '<b>Description:</b> ' + item_description }
            }
            MouseArea {
                anchors.fill: parent
                onClicked: myList.currentIndex = item_index
            }
        }
    }

    Component {
        id: myListHilight
        Rectangle {
            width: parent.width
            height: 40
            color: "lightsteelblue"
            radius: 5
            y: SpringFollow {
                source: myList.currentItem.y
                spring: 3
                damping: 0.2
            }
        }
    }
```

- And finally the actual list that wires the model and the delegates together.

```
    // The actual list
    ListView {
        id: myList
        width: parent.width; height: parent.height
        model: myListModel
        delegate: myListDelegate
        highlight: myListHilight
        focus: true
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }
}
```

For a complete Qt Quick list application example, see List Application

Common Use Cases

BorderImage

The BorderImage element can be used to display an image (eg. a PNG-file) as a border and a background. It is defined as a sci-file where the image area is broken into border, corner, and middle sections using offsets from each respective side. The image is then drawn to the screen so that the borders and the middle part are scaled, but the corners are not. More info

```
BorderImage {
    x: 20; y: 20; width: 230; height: 240
    smooth: true
    source: "myimage.sci"
}
```

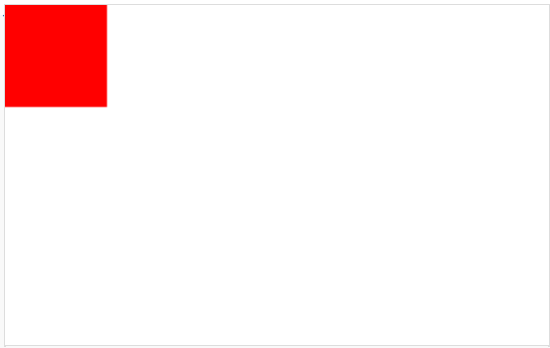
"myimage.sci" file contents:

```
border.left:30
border.top:30
border.right:30
border.bottom:30
source:myimage.png
```

States and Transitions

Elements in QML have states, such as user interface properties (e.g. color and position) that can be changed according to user input. When a state is changed to another, a state transition can be applied to animate the change and to add a visual effect. The following example shows a red rectangle which will turn into a green ball and move from corner to corner when clicked. More about states More about animations and transitions

```
Rectangle {
    id: myRect; width: 640; height: 400; state:
"OldState"
    Rectangle { id: movingRect; width: 60;
height: 60; color: "red"
        MouseArea { anchors.fill: parent;
                    onClicked: if (myRect.state
== "OldState") myRect.state = "NewState";
                               else myRect.state
= "OldState" }
    }
    states: [
        State {
            name: "NewState"
            PropertyChanges { target: movingRect; x: 580; y: 340; radius: 30; color:
"green" }
        }
    ]
    transitions: [
        Transition {
            from: "OldState"; to: "NewState"
            PropertyAnimation { properties: "x,y,color,radius"; duration: 1000 }
        },
        Transition {
            from: "NewState"; to: "OldState"
            PropertyAnimation { properties: "x,y,color,radius"; duration: 1000 }
        }
    ]
```

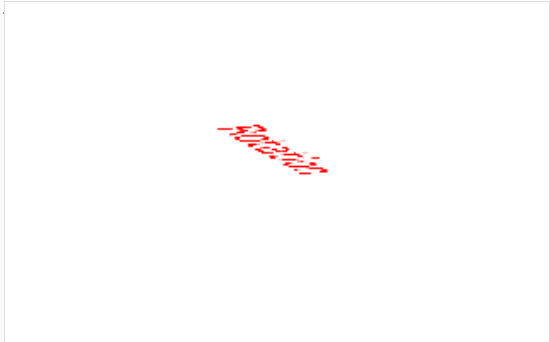


(C) Copyright Nokia 2014. All rights reserved.

```
}
```

Simple 3D animation

Items can be graphically rotated by manipulating the appropriate properties. The following snippet rotates a piece of text on screen around all three axes. More info



```
Text {
     id: myText
     property double xAngle: 0; property double
yAngle: 0; property double zAngle: 0
     text: "Rotation"; font.pointSize: 100;
color: "red"; x: 150; y: 100
     transform: Rotation { origin.x: 300;
origin.y: 80; axis {x:1; y:0; z:0}
angle:myText.xAngle }
     transform: Rotation { origin.x: 300;
origin.y: 80; axis {x:0; y:1; z:0} angle:myText.yAngle }
     transform: Rotation { origin.x: 300; origin.y: 80; axis {x:0; y:0; z:1}
angle:myText.zAngle }
}
Timer {
     interval: 30; running: true; repeat: true
     onTriggered: { myText.xAngle = myText.xAngle + 1;
                    myText.yAngle = myText.yAngle + 1.5;
                    myText.zAngle = myText.zAngle + 2.5 }
}
```

Fonts

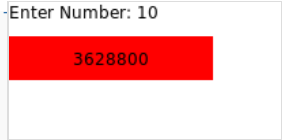
Fonts can be referenced via a font name (eg. "Courier"), loaded from a file, or a network url. The following snippet loads a font from the net and displays some text using it. The font loader also provides status information of the loading process. More info

```
Rectangle {
    width: 400; height: 200
    FontLoader { id: webFont; source:
"http://www.princexml.com/fonts/steffmann/Starburst.ttf" }
    Text {
        width: parent.width; font.family: webFont.name; font.pointSize: 30
        text: {
            if (webFont.status == 1) "Font loaded"
            else if (webFont.status == 2) "Loading"
            else if (webFont.status == 3) "Error"
        }
    }
}
```

Advanced UI

JavaScript code

JavaScript code can be embedded in QML to extend the QML functionalities. QML can access the JavaScript code, and vice versa: functions can be called when an event occurs (such as a mouse click), and QML elements can be accessed from functions. The script code can come from a file or from the Internet, using URLs. The following snippet sets up a text label, a text input field, and a button. When the button is clicked, the factorial of the text input field is calculated, and the result is placed inside the button. There are two JS functions, the mouse button callback and the factorial calculation. More info




```
Rectangle {
    width: 200; height: 100;
    Text {
        id: label; anchors.topMargin: 10; text: "Enter Number: "
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    TextInput {
        id: input; anchors.left: label.right; focus: true
    }
    Rectangle {
       anchors.top: label.bottom; anchors.topMargin: 10
       width: 150; height: 32; color: "red"
       Text { id:buttonText; anchors.centerIn: parent; text: "Calculate" }
       MouseArea { anchors.fill: parent; onClicked: onMyButtonClicked(); }
    }
    function factorial(a) {
        if (a <= 0) return 1;
        else return a * factorial(a - 1);
    }
    function onMyButtonClicked() {
        var result = factorial(input.text)
        buttonText.text = result
       console.log("factorial in console log: " + result);
    }
}
```

Paths

PathView lays items out along a path and allows scrolling them with flicking. The path is composed of path segments, eg. lines or curves. Item attributes can be modified along the path, for example scaling items bigger when they are moved to the foreground. The following snippet displays three fruit items (text and a picture) on a triangular path. More info

```
ListModel {
    id: model
    ListElement { fruit: "Bilberry";
                  picture:
"http://upload.wikimedia.org/wikipedia/commons/9/91/Norwegian_blueberries.jpg" }
    ListElement { fruit: "Kumquat";
                  picture:
"http://upload.wikimedia.org/wikipedia/commons/8/82/Kumquat.jpeg" }
    ListElement { fruit: "Grapefruit"; picture:
                  "http://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/
Table_grapes_on_white.jpg/250px-Table_grapes_on_white.jpg" }
}
Component {
    id: delegate
    Item {
        width: 100; height: 100
        scale: PathView.scale
        Column { Image { anchors.fill: parent; source: picture;  }
                 Text { text: fruit; font.pointSize: 24; }
        }
    }
}
PathView {
    anchors.fill: parent
    model: model;
    delegate: delegate
    path: Path {
        startX: 100; startY: 100
        PathAttribute { name: "scale"; value: 0.5 }
        PathLine { x:200; y:200 }
        PathAttribute { name: "scale"; value: 1.0 }
        PathLine { x:300; y:100 }
        PathLine { x:100; y:100 }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

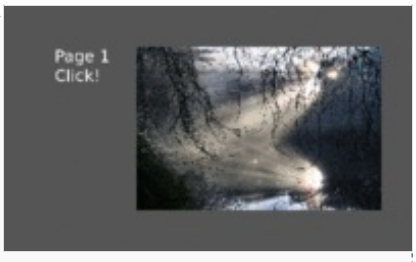
```
}
```

Porting Examples from iPhone

Slide View Transition

Apple iPhone has a build-in slide transition for changing UI views, which can be used by utilizing the UINavigationController class in the applications (See http://developer.apple.com/library/ios/#ANIMATION for more information). This transition can be seen in majority of iPhone applications.

If you want to have this effect also in your ported application, similar effect can be easily produced in QtQuick. Following code example shows an example how to do it.




```
import Qt 4.7

Flipable {
    id: flipable
    width: 800; height: 480;
    property bool flipped
    property int angle

    front: Rectangle {
        id: page1
        anchors.fill: parent
        color: "#555555";
        MouseArea { anchors.fill: parent; onClicked: main.state = 'page2' }
        Text { x:100; y:80; text: "Page 1\nClick!"; font.pointSize:24; color: "white"}
        Image {
source:"http://upload.wikimedia.org/wikipedia/commons/a/af/Crepuscular_rays_with_reflection_in_GGP.jpg"
                x:260; y:80; width:480; height:320; }
    }

    back: Rectangle {
        id: page2
        anchors.fill: parent
        color: "#004400";
        MouseArea { anchors.fill: parent; onClicked: main.state = 'page1' }
        Text { x:600; y:80; text: "Page 2\nClick!"; font.pointSize:24; color: "white"}
        Image {
source:"http://upload.wikimedia.org/wikipedia/commons/thumb/6/66/Thomas_Bresson_-_
Fort_de_Roppe_(15)_(by).JPG/800px-Thomas_Bresson_-_Fort_de_Roppe_(15)_(by).JPG";
                x:100; y:80; width:480; height:320; }
    }

    transform: Rotation {
         origin.x: flipable.width/2; origin.y: flipable.height/2
         axis.x: 1; axis.y: 0; axis.z: 0
         angle: flipable.angle
     }

    states: State {
         name: "back"
         PropertyChanges { target: flipable; angle: 180 }
         when: flipable.flipped
     }

     transitions: Transition {
         NumberAnimation { properties: "angle"; duration: 1500; easing.type: "InOutQuad"
}
     }

     MouseArea {
         anchors.fill: parent
```


(C) Copyright Nokia 2014. All rights reserved.

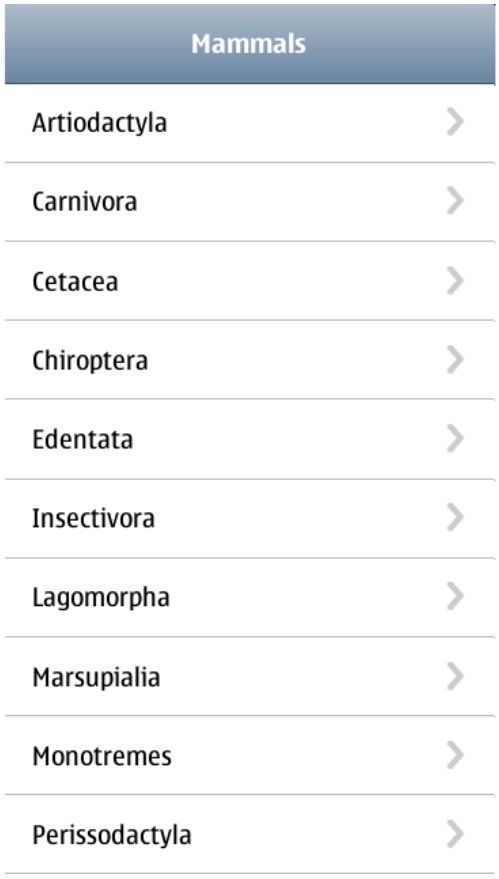
```
            onClicked: flipable.flipped = !flipable.flipped
        }

    }
```

List Application

Qt Quick List Application demonstrates how to migrate iPhone list application to Qt Quick and maintain original look and feel of the iPhone application. UI of the demonstration application consists of list, sub list and detail page.

Follow this link to download Qt Quick List Application source: media:MigrationToQt-QML-List-Sample.zip

For information about basic list implementation with Qt Quick, see section Create a list



(C) Copyright Nokia 2014. All rights reserved.