How to build a tabbed UI with QML

This article shows **how to build a tabbed User Interface by using QML**. A tabbed UI shows a tab bar on the lower part of the screen, containing one or more tabs: each tab, once clicked by the user, shows a different view on the upper part of the UI.

The base structure

Start by creating an empty QML file, called **TabbedUI.qml**: this file will contain the component.

The base structure of this component can be composed of two different parts: an upper Rectangle, that will contain the views associated to each tab, and a lower Rectangle (the tab bar), that will contain the tabs. The component also defines two properties:

- `tabsHeight`: defines the height of the tab bar
- `tabIndex`: defines the index of the currently selected tab

A gradient is added to the tab bar, in order

```
Rectangle {
    // height of the tab bar
    property int tabsHeight : 64

    // index of the active tab
    property int tabIndex : 0

    anchors.fill: parent

    // will contain the tab views
    Rectangle {
 id: tabViewContainer
 width: parent.width

        anchors.top: parent.top
        anchors.bottom: tabBar.top
    }

    // the tab bar
    Rectangle {
 height: tabsHeight
 width: parent.width

 // take the whole parent width
 anchors.left: parent.left
 anchors.right: parent.right

 // attach it to the view bottom
 anchors.bottom: parent.bottom

 id: tabBar

 gradient: Gradient {
     GradientStop {position: 0.0; color: "#666666"}
     GradientStop {position: 1.0; color: "#000000"}
 }
    }
}
```

The Model

(C) Copyright Nokia 2013. All rights reserved.

In order to populate the TabbedUI, a Model will be used: specifically, the component will use a VisualItemModel, where each visual item represents the view associated to each tab. Each visual item will be a Tab object, as defined below:

```
Rectangle {
    // name of the tab
    property string name

    // icon to be displayed in the tab
    property string icon

    anchors.fill: parent
}
```

The Tab component simply extends the Rectangle object, and defines two properties that are used to create the tab on the TabbedUI's tab bar:

- `name`: the name of the tab
- `icon`: the icon displayed within the tab

By using a VisualItemModel and the Tab component, a model for the TabbedUI, containing 3 tabs, can be defined as follows:

```
VisualItemModel {
    id: tabsModel
    Tab {
 name: "Tab 1"
 icon: "pics/tab0.png"

 color: "yellow"
 Text {
     anchors.centerIn: parent
     text: "This is page 1"
 }
    }
    Tab {
 name: "Tab 2"
 icon: "pics/tab1.png"

 color: "green"
 Text {
     anchors.centerIn: parent
     text: "This is page 2"
 }
    }
    Tab {
 name: "Tab 3"
 icon: "pics/tab2.png"

 color: "red"
 Text {
     anchors.centerIn: parent
     text: "This is page 3"
 }
    }
}
```

Using the VisualItemModel

Let's take back the TabbedUI component, and add a new property that will hold the model:

```
Rectangle {

    [...]

    // the model used to build the tabs
    property VisualItemModel tabsModel

    [...]

}
```

Now, the **tabViewContainer** Rectangle can be populated by using a Repeater together with the provided model, as follows.

```
Rectangle {
    id: tabViewContainer
    width: parent.width

    anchors.top: parent.top
    anchors.bottom: tabBar.top

    // build all the tab views
    Repeater {
 model: tabsModel
    }
}
```

Similarly, the tab bar can be built with a Repeater. Since tabs have to be placed on a row, a Row positioner is added to the **tabBar** Rectangle:

```
// the tab bar
Rectangle {
    id: tabBar

    [...]

    // place all the tabs in a row
    Row {
 anchors.fill: parent

 id: tabs
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

To populate the Row added to the tab bar, let's define a Component that will be used as **delegate** to build a single tab bar Item.

This component is automatically sized depending on the number of items in the **tabsModel**, and is composed of:

- An Image object holding the tab icon
- A Text object holding the tab name
- A MouseArea that intercepts the mouse events and calls a **tabClicked()** function, that will be defined later

```
Component {
    id: tabBarItem

    Rectangle {
 height: tabs.height
 width: tabs.width / tabsModel.count

 color: "transparent"

 Image {
     source: tabsModel.children[index].icon
     anchors.horizontalCenter: parent.horizontalCenter
     anchors.top: parent.top
     anchors.topMargin: 4
 }

 Text {
     anchors.horizontalCenter: parent.horizontalCenter
     anchors.bottom: parent.bottom
     anchors.bottomMargin: 4
     color: "white"
     text: tabsModel.children[index].name
 }

 MouseArea {
     anchors.fill: parent
     onClicked: {
  tabClicked(index);
     }
 }
    }
}
```

Now, the **tabs** Row can be populated by using a Repeater as follows:

```
// the tab bar
Rectangle {
    id: tabBar

    [...]

    // place all the tabs in a row
    Row {
 anchors.fill: parent

 id: tabs

 Repeater {
     model: tabsModel.count

     delegate: tabBarItem
 }
    }
}
```

Selecting tabs

Now that we have the static component, some behavior needs to be added. When the user clicks on a tab, the Component must:

- **highlight** the clicked tab, and unselect the previous tab
- **show** the associated tab view, and hide the previously shown tab view

In order to to this a **tabClicked()** JavaScript function is defined:

```
function tabClicked(index)
{
    // unselect the currently selected tab
    tabs.children[tabIndex].color = "transparent";

    // hide the currently selected tab view
    tabsModel.children[tabIndex].visible = false;

    // change the current tab index
    tabIndex = index;

    // highlight the new tab
    tabs.children[tabIndex].color = "#30ffffff";

    // show the new tab view
    tabsModel.children[tabIndex].visible = true;
}
```

Component startup

When the component has been completely initialized, the default tab has to be selected. The Component.onCompleted signal can be used for this purpose:

```
Component.onCompleted:
{
    // hide all the tab views
```


(C) Copyright Nokia 2013. All rights reserved.

```
    for(var i = 0; i < tabsModel.children.length; i++)
    {
 tabsModel.children[i].visible = false;
    }
    // select the default tab index
    tabClicked(tabIndex);
}
```

Using the TabbedUI component

The TabbedUI component can be used as shown in the following code snippet:

```
Rectangle {
    width: 360
    height: 640

    id: root

    VisualItemModel {
 id: tabsModel
 Tab {
     name: "Tab 1"
     icon: "pics/tab0.png"

     color: "yellow"
            Text {
  anchors.centerIn: parent
                text: "This is page 1"
            }
        }
 Tab {
     name: "Tab 2"
     icon: "pics/tab1.png"

     color: "green"
            Text {
  anchors.centerIn: parent
                text: "This is page 2"
            }
        }
 Tab {
     name: "Tab 3"
     icon: "pics/tab2.png"

     color: "red"
            Text {
  anchors.centerIn: parent
                text: "This is page 3"
            }
        }
    }

    TabbedUI {
 tabsHeight: 72
 tabIndex: 1
        tabsModel: tabsModel
    }
}
```

The above code can be seen in action in the following video.

The media player is loading...

Related content

A Qt Creator project containing the source code presented in this article is available here: File:TabbedQuickApp.zip

A Symbian^3 installation file is also available here: File:Tabbedquickapp qt-4 7 0 exp m1 1 0 s3 v0 9.zip

QML Elements

Reference docs of the main QML Elements used in this article are available here: Rectangle, Text, Gradient, GradientStop, VisualItemModel, Component, MouseArea, Row, Image

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.