How to create a Page Control component in QML

This article shows a simple Qt Quick UI Component for switching between application pages/views. The *PageControl* manages a set of pages, allowing the user to flick between them using horizontal sliding gestures or by pressing left and right keys. The component includes an indicator panel that highlights the current page within the set.

06 Mar 2011

The base structure

The component is structured in two main elements:

- a root Rectangle, that contains the whole component
- a PathView that places and animates the single page elements

Since the `PathView` must be populated with a ListModel, the component declares a property that allows to set the model itself.

The base structure can be written in a file named **PageControl.qml** as:

```
import QtQuick 1.0

Rectangle {
    id: pageControl

    property ListModel listModel

    clip: true

    PathView {
 id: myPathView

 model: listModel
 anchors.fill: parent
 focus: true
 interactive: true
    }
}
```


(C) Copyright Nokia 2014. All rights reserved.

The PathView

The `PathView` must allow to switch from a page to another with right/left sliding movements, so it can be defined with a straight horizontal Path, composed of just one PathLine. The length of this `PathLine` must be proportional to the number of pages of the component itself: since each page must have the same width (and height) of the `PageControl` component, the length of this line will be equal to the width of one page multiplied by the number of pages.

```
PathView {
    id: myPathView

    preferredHighlightBegin: 0.5
    preferredHighlightEnd: 0.5

    [...]

    path: Path {
 startX: - pageControl.width * listModel.count / 2 + pageControl.width / 2
 startY: pageControl.height / 2
 PathLine {
     x: pageControl.width * listModel.count / 2 + pageControl.width / 2
     y: pageControl.height / 2
 }
    }
}
```

Since the current page must be shown in the middle of the Path defined above, the **preferredHighlightBegin** and **preferredHighlightEnd** properties are both defined equal to 0.5 (the middle value between 0.0 and 1.0).

The PathView delegate

The `PathView` delegate represents a single page shown by the PathView itself. This article uses a simple delegate Component, composed of a base Rectangle and a Text element placed in the middle of it. More complex UI elements can be easily defined by modifying the delegate itself.

```
Component {
    id: appDelegate

    Rectangle {
 smooth: true
 width: pageControl.width
 height: pageControl.height
 color: pageColor

 Text {
     anchors.centerIn: parent
     color: "white"
     text: name
 }
    }
}
```

An appropriate model for this delegate must at least contain two properties:

- a `pageColor` property, defining the color of the base `Rectangle`
- a `name` property, defining the text to be displayed by the `Text` element

Below is a sample model:

(C) Copyright Nokia 2014. All rights reserved.

```
ListModel {
    id: appModel
    ListElement { name: "This is page 1"; pageColor: "blue" }
    ListElement { name: "This is page 2"; pageColor: "green" }
    ListElement { name: "This is page 3"; pageColor: "red" }
}
```

The `PathView` element can now be modified to use the `appDelegate` Component defined above:

```
PathView {
    id: myPathView

    [...]

    delegate: appDelegate
}
```

Adding key-based navigation

To allow '*key-based navigation*, the `PageControl` component should:

- show the next page when the **right** key is pressed
- show the previous page when the **left** key is pressed

To implement this behavior, Keys event handlers are used:

```
PathView {
    id: myPathView

    Keys.onRightPressed: if (!moving && interactive) incrementCurrentIndex()
    Keys.onLeftPressed: if (!moving && interactive) decrementCurrentIndex()

    [...]
}
```

Modifying the PathView deceleration

The `PathView` declares a `flickDeceleration` property that allows you to specify the rate at which a flick movement decelerates: the higher the `flickDeceleration` value, the greater the deceleration. In order to prevent several pages being flicked at once (the `PageControl` component allows switching only between adjacent pages), a high deceleration value must be used. In the example below a value of 500 is used:

```
PathView {
    id: myPathView

    [...]

    flickDeceleration: 500
}
```

Adding a signal on page change

When the current page changes, it would be useful to have a signal notifying about this change. This can be easily implemented by adding a signal to the `PageControl` component, and by using the `PathView`'s `onCurrentIndexChanged` signal to call it with the

(C) Copyright Nokia 2014. All rights reserved.

new page index as argument.

```
Rectangle {
    id: pageControl

    [...]

    signal pageChanged(int page)

    PathView {
 id: myPathView

 onCurrentIndexChanged: pageChanged(currentIndex)

 [...]
    }
}
```

The page indicators

The `PageControl` component also defines a lower UI indicator that shows:

- the total number of available pages
- the position of the current page in the set of pages

This information is represented with a set of visual elements (Rectangle elements), with the element representing the current page highlighted with a different style (for instance, a different color). The component is then modified to include, in the lower area, a Rectangle that will contain the page indicators:

```
import QtQuick 1.0

Rectangle {
    id: pageControl

    [...]

    PathView {
 id: myPathView

 [...]
    }

    Rectangle {
 id: pageIndicator

 width: parent.width
 height: 40
 anchors.bottom: parent.bottom

 color: "white"
    }
}
```

The page indicators must be placed horizontally, so a Row element can be used. Since the indicators must be dynamically generated, depending on the number of pages, a Repeater is added to the `Row` element, with a delegate that builds the Rectangles representing the various page indicators.

(C) Copyright Nokia 2014. All rights reserved.

```
Rectangle {
    id: pageIndicator

    width: parent.width
    height: 40
    anchors.bottom: parent.bottom

    Row {
 anchors.centerIn: parent
 Repeater {
     model: listModel.count
     delegate: Rectangle {
  width: 20
  height: pageIndicator.height
  color: "white"
  Rectangle {
      anchors.centerIn: parent
      width: 10
      height: 10
      color: (myPathView.currentIndex == index ? "black" : "gray")
  }
     }
 }
    }

    color: "white"
}
```

How to use the PageControl

The video below shows the component in action on a Nokia N8 device

The media player is loading...

The PageControl component can be easily used by defining a model and, optionally, by handling the `onPageChanged` signal as shown below:

```
Rectangle {
    width: 360
    height: 360

    Rectangle {
 id: myRect
 color: "lightblue"
 height: 40
 width: parent.width

 Text {
     id: myText
     anchors.centerIn: parent
     color: "black"
 }
    }

    ListModel {
 id: appModel
```


(C) Copyright Nokia 2014. All rights reserved.

```
  ListElement { name: "This is page 1"; pageColor: "blue" }
  ListElement { name: "This is page 2"; pageColor: "green" }
  ListElement { name: "This is page 3"; pageColor: "red" }
  ListElement { name: "This is page 4"; pageColor: "gray" }
  ListElement { name: "This is page 5"; pageColor: "black" }
  ListElement { name: "This is page 6"; pageColor: "purple" }
     }


     PageControl {
  width: parent.width
  anchors.top: myRect.bottom
  anchors.bottom: parent.bottom
  listModel: appModel

  onPageChanged: myText.text = "Page changed: " + page
     }
 }
```

Related content

The Qt Creator project containing the code presented in this article is available here: File:QMLPageControl.zip

(C) Copyright Nokia 2014. All rights reserved.