Creating a spinner list with QML

This article explains how to create a simple spinner list with QML.

Introduction

With this article I would like to show you a simple way to create your custom spinner list QML component. Widgets like this are already available on Symbian and Harmattan components sets, but they are more complex. If you need to implement a spinner list, which is platform indipendent, easy and simple to customize, this page is made for you.

What's a spinner list?

A spinner list is a widget which can replace combo box in UIs for touch devices. The list of options available are placed on a cylinder which can rotate on the horizonal axis, allowing users to select only one item at a time.

In the next paragraph I will show you how to implement this widget, which you can see in the video.

PLEASE NOTE: I had some problems recording the video. The little black squares you see are defects of the recording program I faced. I apologize for that.

The media player is loading...

Dissecting the widget

This widget is basically a simple ListView. This primitive element is flickable, so touch-friendly, and it allows the visualization of items stored in its model. An integer can be used as model if you want to display numbers from 0 to N. A QStringList could be used if you want to display strings. Of course it's possible to use more complex models, but this will make our delegate complex too. BTW you can read more about ListView here.

To "transform" the listview in a cilynder I applied some gradients on top of listview. The gradient doesn't consume mouse events so it's fine from this prospective, but the usage of translucents gradients have pros and cons:

- Pros: eye candy text and possibility to animate gradients
- Cons: translucent gradients are computionally expensive

Anyway, all current Nokia devices supporting QtQuick should be able to use gradients without big FPS loss. In case you need better performance, I suggest you to remove gradients and place a static image under the listview. This works fine since the delegate is a transparent rectangle or an Item derived element.

Another important thing for the implementation of this widget is its ability to update other elements when the user changes the selected item. For this purpose, the UI can listen for ListView::currentIndexChanged signal. The index doesn't change automatically but it's updated when move animations end.

Code

The code below is the one which you have seen running on the video above. The code has been explained in the previous paragraph and it's quite easy to understand. Gradient is declared after the listview in order to be placed above the list. onCurrentIndexChanged slot has been implemented just to show you a way to catch the currentIndexChanged signal. The index is calculated by multiplying list.visibleArea.yPosition (percentage of scrolled view) by list.count (number of items in the list). ListView::VisibleArea has to be initialized in order to work, probably because of a bug. For this reason you have to call it when the component is created.

```
// import QtQuick 1.0 // to target S60 5th Edition or Maemo 5
import QtQuick 1.1

Rectangle {
    width: 360
    height: 360
    color: "black"
```


(C) Copyright Nokia 2014. All rights reserved.

```
    Rectangle {
        id: mSpinner

        anchors.centerIn: parent
        width: 300
        height: width
        radius: 20
        color: "gray"

        ListView {
            id: list
            anchors.fill: parent
            clip: true

            snapMode: ListView.SnapToItem
            model: 10
            delegate: Rectangle{
              width: list.width
              height: list.height
              color: "transparent"
              rotation: (index % 2) ? -10 : 10

              Text {
                  anchors.centerIn: parent
                  horizontalAlignment: Text.AlignHCenter
                  text: index
                  font.pixelSize: list.width
              }
            }

            onCurrentIndexChanged: console.log("CURRENT INDEX " + list.currentIndex)
            onMovementEnded: list.currentIndex = list.visibleArea.yPosition * list.count

            // Workaround: Compute visibleArea
            Component.onCompleted: list.visibleArea
        }

        Rectangle {
            id: overlay
            anchors.fill: parent
            gradient: Gradient {
                     GradientStop { position: 0.0; color: "black" }
                     GradientStop { position: 0.18; color: "transparent" }
                     GradientStop { position: 0.33; color: "white" }
                     GradientStop { position: 0.66; color: "transparent" }
                     GradientStop { position: 1.0; color: "black" }
            }
        }
    }
}
```


(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.