How to create a Collapsible Panel with QML

This article shows how to create and use a **Collapsible Panel** component in QML. You can directly use this component in your Qt Quick applications - the component and the demo application are available for download in this article. Download it from here.

20 Mar 2011

Introduction

Collapsible panels have two states, collapsed and expanded. When collapsed the panel occupies a single line - the user can *toggle* the panel to make the content in the panel visible (and invisible). Collapsible panels provide a very efficient use of screen space and are typically used when a lot of data is to be made available but only a subset of the data is needed by the user at a time. For example, application settings are often kept in collapsible sections.

The Collapsible Panel component makes it easy for you to add collapsible sections to your applications.

Collapsible Panel QML Component

The Collapsible Panel demonstrated below is created as a reusable QML component. The following sections teach you how this component is coded and how to use it in your applications.

Basic Structure

(C) Copyright Nokia 2014. All rights reserved.

The Collapsible Panel component has two main elements,

- **Title/heading**
- **List view**

So the main.qml at a higher level would look like the following

```
// CollapsiblePanel.qml
import QtQuick 1.0

Rectangle{
 id: myRect

 Rectangle{
  id: titleRect
         Text{
                    id: titleTxt
                 }
 }

 ListView{
 id: listView
 }
}
```

Allowing customization

Since this is a component, we allow the developer to use their own model and delegate for the ListView, as well as his custom text for the `titleRect`. We can use property aliases for these properties for a direct reference to existing property. Refer to the documentation for more info on Property aliases.

Since we are using the `ListView`, which is going to be collapsed, it makes sense to also let the developer know which item in the list was selected when the panel was expanded. So our component should also have a signal which is emitted when a list item is selected along with the index of the item.

With the above two points in consideration, our `CollapsiblePanel` component now looks something like this:

```
// CollapsiblePanel.qml
import QtQuick 1.0

Rectangle{
 id: myRect

 signal itemselected(int index)

    property alias titleText: titleTxt.text
    property alias customModel: listView.model
    property alias customDelegate: listView.delegate


 Rectangle{
  id: titleRect
         Text{
            id: titleTxt
            }
 }

 ListView{
```


(C) Copyright Nokia 2014. All rights reserved.

```
  id: listView
  model: customModel
  delegate: customDelegate
  }
 }
```

Implementing the logic, states and transitions

With the above basic structure in place, we now have to implement the collapse and expand logic, when the `titleRect` is clicked. But before that, lets add some anchors to the `ListView`, `titeRect` and choose font, colors for the text of the tile. Further, we add a graphical representation of the state of the Panel to the `titleRect`. The `titleRect` now becomes more presentable.

```
    Rectangle{
        id: titleRect
        width: parent.width
        height: 40
        gradient: Gradient {
            GradientStop { position: 0.0; color: "#1141a7" }
            GradientStop { position: 1.0; color:  "#0f1273"}
        }
        Image {
            id: titleArrow
            width:  20
            height:  20
            anchors.left: parent.left
            anchors.leftMargin: 5
            anchors.verticalCenter: parent.verticalCenter
            source: "images/arrowdown.png"

        }
        Text{
            id: titleTxt
            anchors.left: titleArrow.right
            anchors.leftMargin: 10
            anchors.verticalCenter: parent.verticalCenter
            font.bold: true
            font.pixelSize: 18
            color:  "#c1c3c8"
        }
    }
```

We add a MouseArea to the `titleRect` to detect mouse events or touch events. When the title bar is clicked we want the list to collapse or expand - so why not add a "collapsed" state which would change the properties to make the listView collapsed.

In the collapsed state, we need to do two things:

1. **Make the list view collapse**
2. **Change the image in the title bar**

We can hide the listView by making it invisible (opacity = 0); however we want the list to look like stacking or collapsing - so we also set its height to 0.

The image in the title bar is an arrow pointing downwards. Instead of changing the image in the title bar, wouldn't it make sense to rotate the same image towards the right?

```
 states: [
  State {
```


(C) Copyright Nokia 2014. All rights reserved.

```
  name: "collapsed"
  PropertyChanges {
   target: listView
   height: 0
   opacity: 0
  }
  PropertyChanges {
   target: titleArrow
   rotation: -90
  }
 }
]
```

Along with the property changes, we also add some transitions to improve the component's look and feel. We animate all the properties (height, opacity, rotation) that we changed in the collapsed state, so that the interaction looks natural.

```
transitions: [
 Transition {
  NumberAnimation { target: listView; property: "height"; duration: 400 }
  NumberAnimation { target: listView; property: "opacity"; duration: 400 }
  NumberAnimation { target: titleArrow; property: "rotation"; duration: 300 }
 }
]
```

Well, we are almost done! The collapsible panel is now a usable component, provided with a valid list model and delegate. However, we need to keep in mind that since this is a component, its a good idea to consider its size to be dynamic i.e. the size of the component should decrease when it is collapsed and increase when it is expanded, so that it can be used with other components easily. To achieve this we just need to make sure that when the component is loaded, it adapts to the height according to the number of elements the `listView` contains.

```
Component.onCompleted:  {
 myRect.height = titleRect.height+listView.contentHeight
}
```

We also want the component to shrink its size when collapsed, so we add one more `PropertyChange` to the "collapsed" state - which would change the component's height to the height of titleRect.

```
PropertyChanges {
 target: myRect
 height: titleRect.height
}
```

Using the Collapsible Panel

Now that the component is ready to be used, lets try it out in one of our demo applicatiosn. We need to have a custom model (we'll use a `ListModel` here) and delegate to use the component.

Let's begin by defining 3 ListModels - date and time model (**d_t_SettingsModel**), **callSettingsModel** and **phoneSettingsModel**.

```
   // List Model 1
    ListModel {
        id: d_t_SettingsModel
        ListElement {
```


(C) Copyright Nokia 2014. All rights reserved.

```
            settingNumber: 1
            settingName: "Time"
            settingValue: "18:03"
        }
        ListElement {
            settingNumber: 2
            settingName: "Time zone"
            settingValue: "GMT +5:30 Mumbai"
        }
        ListElement {
            settingNumber: 3
            settingName: "Date"
            settingValue: "28 February"
        }
        ListElement {
            settingNumber: 4
            settingName: "Date format"
            settingValue: "dd mm yy"
        }
        ListElement {
            settingNumber: 5
            settingName: "Time format"
            settingValue: "24 hour"
        }
        ListElement {
            settingNumber: 6
            settingName: "Clock type"
            settingValue: "Analogue"
        }
    }

    // List Model 2
    ListModel {
        id: callSettingsModel
        ListElement {
            settingNumber: 1
            settingName: "Send my caller ID"
            settingValue: "No"
        }
        ListElement {
            settingNumber: 2
            settingName: "Call Waiting"
            settingValue: "On"
        }
        ListElement {
            settingNumber: 3
            settingName: "Video in call"
            settingValue: "Ask first"
        }
        ListElement {
            settingNumber: 4
            settingName: "Show call duration"
            settingValue: "Yes"
        }
        ListElement {
            settingNumber: 5
            settingName: "Show call summary"
```


(C) Copyright Nokia 2014. All rights reserved.

```
            settingValue: "Yes"
        }
        ListElement {
            settingNumber: 6
            settingName: "Active profile"
            settingValue: "General"
        }
    }

   // List Model 3
    ListModel {
        id: displaySettingsModel
        ListElement {
            settingNumber: 1
            settingName: "Light Sensor"
            settingValue: "20%"
        }
        ListElement {
            settingNumber: 2
            settingName: "Font Size"
            settingValue: "Normal"
        }
        ListElement {
            settingNumber: 3
            settingName: "Screen/keylock time-out"
            settingValue: "1 minute"
        }
        ListElement {
            settingNumber: 4
            settingName: "Welcome note/logo"
            settingValue: "Default"
        }
        ListElement {
            settingNumber: 5
            settingName: "Light time-out"
            settingValue: "5 seconds"
        }
    }
```

Now for the delegate of the `CollapsiblePanel`. We need to implement a `MouseArea` in our delegate which would emit the `itemselected(index)` signal for the `CollapsiblePanel` component. Note that, we use a `statusBar` in the following delegate which is declared below.

```
    Component
    {
        id: myDelegate
        Rectangle{
            id: myDelegateItem
            width: parent.width
            height:  30
            gradient:unpressedDelegateItem
            MouseArea{
                anchors.fill:  parent
                onPressed: {
                    myDelegateItem.gradient = pressedDelegateItem
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    border.width = 2
                    border.color = "yellow"
                    statusBar.text = settingName + " Clicked" // statusBar declared
later
                    myRect.itemselected(index)
                }
                onReleased: {
                    border.width = 0
                    statusBar.text= "Item selection triggers a signal too!" // statusBar
declared later
                    myDelegateItem.gradient = unpressedDelegateItem
                }
            }
            Text {
                anchors.left: parent.left
                anchors.leftMargin: 10
                anchors.verticalCenter: parent.verticalCenter
                text: settingName
                color: "#c1c3c8"
                font.bold: true
                font.pixelSize: 14
            }
            Text {
                anchors.right: parent.right
                anchors.rightMargin: 10
                anchors.verticalCenter: parent.verticalCenter
                text: settingValue
                color: "#c1c3c8"
                font.bold: true
                font.pixelSize: 14
            }
            Gradient {
                id: unpressedDelegateItem
                GradientStop { position: 0.0; color: "#1753ce" }
                GradientStop { position: 1.0; color:  "#15159c"}
            }
            Gradient {
                id: pressedDelegateItem
                GradientStop { position: 0.0; color:  "#15159c"}
                GradientStop { position: 1.0; color:  "#1753ce"}
            }
        }
    }
```

Finally, we can use the `CollapsiblePanel` component, by specifying the model and delegate. We can also implement the `onItemclicked` function if required.

```
CollapsiblePanel
{
 id: panel1
 width: parent.width
 height: parent.height
 anchors.top: titleBar.bottom // declared further below
 titleText: "Date & Time Settings"
 customModel: d_t_SettingsModel
 customDelegate: myDelegate
```


(C) Copyright Nokia 2014. All rights reserved.

```
 onItemselected:{
  console.log(index)
 }

}
CollapsiblePanel
{
 id: Panel2
 width: parent.width
 height: parent.height
 anchors.top: panel1.bottom
 titleText: "Call Settings"
 customModel: callSettingsModel
 customDelegate: myDelegate
 onItemselected:{
  console.log(index)
 }

}
CollapsiblePanel
{
 id: panel3
 width: parent.width
 height: parent.height
 anchors.top: panel2.bottom
 titleText: "Display Settings"
 customModel: displaySettingsModel
 customDelegate: myDelegate
 onItemselected:{
  console.log(index)
 }

}
```

We also add a title bar and a status bar for the demo, as follows.

```
    Rectangle{
        id: titleBar
        width: parent.width
        height: 50
        color: "skyblue"
        Text{
            anchors.centerIn: parent
            text: "Collapsible Panels Demo"
            color: "darkblue"
        }
    }

    Rectangle{
        id: statusBar
        width: parent.width
        height: 50
        anchors.bottom: parent.bottom
        color: "skyblue"

        property alias text: statusText.text
```


(C) Copyright Nokia 2014. All rights reserved.

```
        Text{
            id: statusText
            anchors.centerIn: parent
            color: "darkblue"
            text: "Item selection triggers a signal too!"
        }
    }
```

So the demo app is ready and let's see how it looks!

Video Demo

The video below shows the demo app that we created above in the above section. The media player is loading...

Source code

You can download the source code for the above demo application here. You can also find the **Collapsible Panel** component in it to use for your projects!

- Download source code - File:CollapsiblePanel.zip

(C) Copyright Nokia 2014. All rights reserved.