Using QML ListView

Overview

19 Dec 2010

ListView is one of the standard QML elements made available in the Qt Declarative module. It is a powerful component for displaying an arbitrary list of information, but it can be tricky to understand at first. Items in a ListView are laid out horizontally or vertically. List views are inherently flickable because ListView inherits from Flickable. Unlike many other components, the ListView requires combining a few separate components in order to make it work:

- **ListModel** containing the data for the list
- **Delegate** component, which provides the formatting for individual list items
- **ListView** which is the actual displayed item and refers to the other two

Example Usage

There are several moving parts here, but when assembled properly they provide a set of components that cleanly separates data and presentation. To create a demonstration, we start with a new QML file:

```
import Qt 4.7

Item {
    width: 300
    height: 200
}
```

We are choosing a small size because this could be a subcomponent in a larger QML project (and because this lets us demonstrate the built-in flickable scrolling).

The first thing we will add is our ListModel, which contains our data.

```
Item {
    width: 300
    height: 200

    ListModel {
        id: beerModel
        ListElement {
            beerNumber: 1
            beerName: "Lager"
            beerColor: "light"
        }
    }
}
```

The list model contains a single ListItem entry (which QML refers to as an element), with a few pieces of data. QML calls the data labels *Roles* (e.g. "beerNumber" or "beerName" is a Role). Roles must start with a lower case letter, and (though the documentation doesn't mention this) they should be different from reserved words such as common property names like "id" and "name" and "color" to avoid confusion. I think it is always safer to use longer concatenated names in programming, whether you prefer camel casing (like used here) or underscores (like_this), to avoid stepping on your language keywords--others prefer to pepper in underscores or use other methods to ensure distinctiveness. In any case, you can add additional list elements like so:

```
Item {
    width: 300
    height: 200

    ListModel {
        id: beerModel
        ListElement {
            beerNumber: 1
            beerName: "Lager"
            beerColor: "light"
        }
        ListElement {
            beerNumber: 2
            beerName: "Porter"
            beerColor: "dark"
        }
        ListElement {
            beerNumber: 3
            beerName: "Stout"
            beerColor: "really dark"
        }
        ListElement {
            beerNumber: 4
            beerName: "Pale Ale"
            beerColor: "deep amber"
        }
        ListElement {
            beerNumber: 5
            beerName: "Weiss"
            beerColor: "white"
        }
        ListElement {
            beerNumber: 6
            beerName: "Lambic"
            beerColor: "varied"
        }
        ListElement {
            beerNumber: 7
            beerName: "Amber"
            beerColor: "amber"
```


(C) Copyright Nokia 2013. All rights reserved.

```
        }
        ListElement {
            beerNumber: 8
            beerName: "Pilsner"
            beerColor: "yellow"
        }
        ListElement {
            beerNumber: 9
            beerName: "Light"
            beerColor: "water"
        }
    }
}
```

Note that we have both numeric data and string data--we are not restricted to only string data.

Now that we have our data, the next step is the delegate component that displays it. Think of the delegate component as if you were writing a single QML component to display a single ListElement entry. This will be repeated over and over with each ListElement entry. For starters, we are going to create a very simple delegate:

```
    Component {
        id: beerDelegate
        Text {
            text: beerName + " - " + beerColor
        }
    }
```

Notice that we have access directly to our Roles ("beerName" and "beerColor") as data elements, and that there is no requirement to use everything (we are not using "beerNumber" in this particular example.) This component should follow the ListModel component in our example, but the order is not important as long as the ListView (which we are about to create) can find the ListModel and the delegate component somewhere.

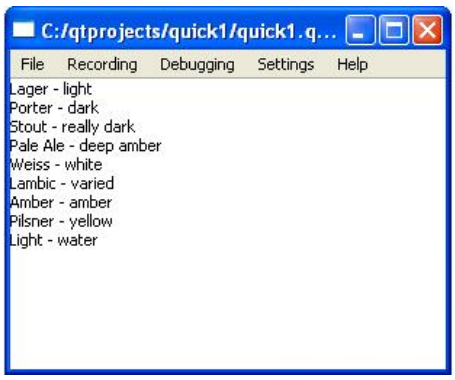
Now we need a ListView object to tie it all together. This is the component that will actually be visible, and which will refer to the other two, but it is still extremely simple. We give it an id and anchors (though it could also have had a height and width of its own instead) and assign the model and delegate:

```
    ListView {
        id: beerList
        anchors.fill: parent
        model: beerModel
        delegate: beerDelegate
    }
```

In case you are wondering how all the code goes together, here is the basic structure (with the full sections above replacing the ...):

```
import Qt 4.7

Item {
    width: 300
    height: 200

    ListModel {
        ...
    }

    Component {
        ...
    }

    ListView {
       ...
    }
}
```

Run this in qmlviewer (save it as BeerList.qml and run *qmlviewer BeerList.qml*) and you should see a simple list of the text. Click and drag up or down and you will see the flickable behavior that you get without adding any code. Otherwise, though, it doesn't do much right now. And it's ugly.

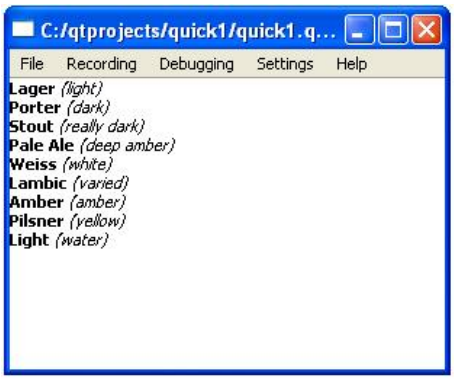


So lets gussy it up a little. If we want to format each item a little, we only need to change the delegate component. We can change it to the following:

```
    Component {
        id: beerDelegate
        Text {
            text: "<b>" + beerName + "</b> <i>(" + beerColor + ")</i>"
        }
    }
```

Now run it again in qmlviewer and you will see a little formatting applied.

(C) Copyright Nokia 2013. All rights reserved.



But it's still ugly, and it still doesn't let us do anything interesting. First we'll address the ugliness:

```
    Component {
        id: beerDelegate
        Rectangle {
            id: beerDelegateRectangle
            height: beerDelegateText.height * 1.5 + blueline.height
            width: parent.width
            Rectangle {
                id: blueline
                width: parent.width
                height: 2
                anchors.horizontalCenter: parent.horizontalCenter
                anchors.top: parent.top
                color: "navy"
            }
            Text {
                id: beerDelegateText
                anchors.verticalCenter: parent.verticalCenter
                anchors.horizontalCenter: parent.horizontalCenter
                text: "<b>" + beerName + "</b> <i>(" + beerColor + ")</i>"
            }
        }
    }
```

Now run it again in qmlviewer and it looks a bit better.

We've spaced it out a bit more, centered the text, and added a separator between each row. You should also be able to see now what it will look like when the list is bigger than the space provided...how it scrolls up and down.

Now we will add some functionality. We'll fix it so that it operates like a selectable listbox--where users can click on a row and select an item.

Add a property for selectedBeerNumber to our mail Item:

```
Item {
    width: 300
    height: 200
    property int selectedBeerNumber: -1
    ...
}
```

Then update the beerDelegate component as follows:

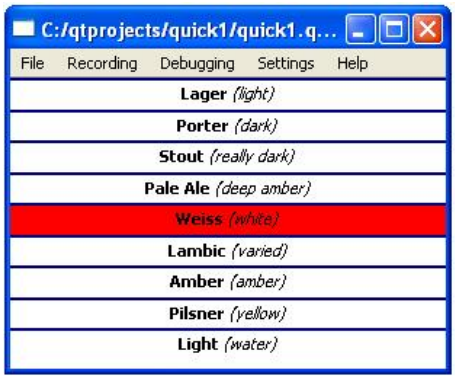
```
    Component {
        id: beerDelegate
        Rectangle {
            id: beerDelegateRectangle
            height: beerDelegateText.height * 1.5 + blueline.height
            width: parent.width
            Rectangle {
                id: blueline
                width: parent.width
                height: 2
                anchors.horizontalCenter: parent.horizontalCenter
                anchors.top: parent.top
                color: "navy"
            }
            Text {
                id: beerDelegateText
                anchors.verticalCenter: parent.verticalCenter
                anchors.horizontalCenter: parent.horizontalCenter
                text: "<b>" + beerName + "</b> <i>(" + beerColor + ")</i>"
```


(C) Copyright Nokia 2013. All rights reserved.

```
            }
            MouseArea {
                anchors.fill: parent
                onClicked: {
                    selectedBeerNumber = beerNumber;
                }
            }
            states: [
                State {
                    name: "selected"
                    when: (beerNumber==selectedBeerNumber)
                    PropertyChanges {target: beerDelegateRectangle; color: "red"}
                }
            ]
        }
    }
```

We've added a MouseArea which sets the selectedBeerNumber property, and we have added a state to show that the item is selected. The state is only activated when the *when* clause is true, so only the item whose beerNumber matches the selectedBeerNumber property will take on the "selected" state--any others will revert to the default state, which is simply the way the object is originally defined (e.g. without the red).

Now if we run the qmlviewer, we see the list, and when we click our favorite beer it will be highlighted in red.



Here is the complete file in its final version:

```
import Qt 4.7

Item {
    id: beerListMain
    width: 300
    height: 200
    property int selectedBeerNumber: -1

    ListModel {
        id: beerModel
        ListElement {
            beerNumber: 1
            beerName: "Lager"
            beerColor: "light"
        }
        ListElement {
            beerNumber: 2
            beerName: "Porter"
            beerColor: "dark"
        }
        ListElement {
            beerNumber: 3
            beerName: "Stout"
            beerColor: "really dark"
        }
        ListElement {
            beerNumber: 4
            beerName: "Pale Ale"
            beerColor: "deep amber"
        }
        ListElement {
            beerNumber: 5
            beerName: "Weiss"
            beerColor: "white"
        }
        ListElement {
            beerNumber: 6
            beerName: "Lambic"
            beerColor: "varied"
        }
        ListElement {
            beerNumber: 7
            beerName: "Amber"
            beerColor: "amber"
        }
        ListElement {
            beerNumber: 8
            beerName: "Pilsner"
            beerColor: "yellow"
        }
        ListElement {
            beerNumber: 9
            beerName: "Light"
            beerColor: "water"
        }
    }

    Component {
        id: beerDelegate
        Rectangle {
            id: beerDelegateRectangle
            height: beerDelegateText.height * 1.5 + blueline.height
            width: parent.width
            Rectangle {
                id: blueline
                width: parent.width
```


(C) Copyright Nokia 2013. All rights reserved.

```
                height: 2
                anchors.horizontalCenter: parent.horizontalCenter
                anchors.top: parent.top
                color: "navy"
            }
            Text {
                id: beerDelegateText
                anchors.verticalCenter: parent.verticalCenter
                anchors.horizontalCenter: parent.horizontalCenter
                text: "<b>" + beerName + "</b> <i>(" + beerColor + ")</i>"
            }
            MouseArea {
                anchors.fill: parent
                onClicked: {
                    selectedBeerNumber = beerNumber;
                }
            }
            states: [
                State {
                    name: "selected"
                    when: (beerNumber==selectedBeerNumber)
                    PropertyChanges {target: beerDelegateRectangle; color: "red"}
                }
            ]
        }
    }

    ListView {
        id: beerList
        anchors.fill: parent
        model: beerModel
        delegate: beerDelegate
    }
}
```

Further Reading

- QML ListView Element
- QML Elements

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.