QML Tag Cloud

A "tag cloud" could be used in a selection dialog, where the user chooses tags to assign to an item from the list; or in a search dialog, where the user chooses the tags they want to search for. This tutorial shows how to create a simple *tag cloud* UI in less than 100 lines of QML.

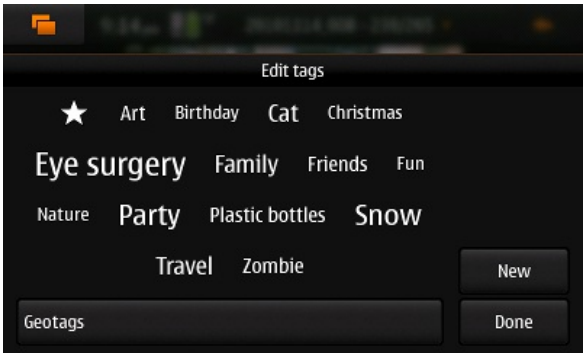
12 Jun 2011

Tags and tag clouds

Tags are short fragments of text (usually one or two words) that can be attached to items as an informal classification method.

A popular interface for displaying a list of tags is the "tag cloud". A tag cloud displays the list of available tags ordered alphabetically, with the most commonly used or important tags appearing in a larger font size to make them easier to find. A tag cloud could be used in a selection dialog, where the user chooses tags to assign to an item from the list; or in a search dialog, where the user chooses the tags they want to search for.

A good use of the tag cloud interface appears in the Maemo 5 Photos application.



We will attempt to create something similar in QML.

A QML Tag Cloud Component

We will create a QML component that displays the contents of a Data Model. In a real application this might be a XmlListModel or a C++ QAbstractItemModel. For demonstration purposes we will use a QML ListModel like this:

```
    ListModel {
        id: photoTags

        ListElement {
            name: "art"       //the name of the tag
            score: 2          //a count of tag uses
            selected: false  //set if tag is selected
        }
        ListElement {
            name: "birthday"
            score: 18
            selected: false
        }
        ListElement {
            name: "family"
            score: 2
            selected: true
        }
        // etc...
    }
```

As already mentioned, tags are meant to be alphabetically ordered in a cloud. We will assume that the Data Model has already done this, rather than attempt to sort them in our component.

Code

For the impatient, I will give the complete code for the TagCloud.qml component first, and then explain parts of it in more detail:

```
import QtQuick 1.0

Rectangle {
    id: tagCloud
    SystemPalette { id: palette }   //we get the system default colors from this

    //these properties are the public API
    property variant model
    property color baseColor: palette.base
```


(C) Copyright Nokia 2013. All rights reserved.

```
    property color textColor: palette.text
    property color hightlightedTextColor: palette.highlightedText
    property color highlightColor: palette.highlight

    color: baseColor
    Flickable {
        id: flickable
        anchors.fill: parent
        boundsBehavior: Flickable.DragOverBounds
        Flow {
            id: flow
            width: parent.width
            spacing: 10
            anchors.margins: 4

            property int maxHeight:0
            Repeater {
                id: repeater
                model: tagCloud.model
                property int minScore   //initially undefined
                property int maxScore:0
                Rectangle {
                    Text {
                        id: textBlock
                        text: name
                        color: selected ? hightlightedTextColor : textColor;
                        anchors.centerIn: parent

                        //QML won't allow "onScoreChanged" for some reason, so
                        // we create a local ref to score and put a change handler on that
                        property int itemScore: score
                        onItemScoreChanged: {
                            repeater.minScore = (repeater.minScore == undefined) ? itemScore: Math.min(itemScore, repeater.minScore)
                            repeater.maxScore = Math.max(itemScore, repeater.maxScore)
                            console.log(index + " minScore:" + repeater.minScore + " maxScore:" + repeater.maxScore)
                        }

                        property double scale:  ((itemScore - repeater.minScore)/(repeater.maxScore - repeater.minScore))
                        font.pointSize:  Math.round(scale*16) + 10

                        onHeightChanged: {
                            flow.maxHeight = Math.max(height, flow.maxHeight)
                        }

                    }
                    radius: 3
                    width: textBlock.width + 4
                    height: flow.maxHeight + 2  //all rows are the same height
                    color: selected ? highlightColor : baseColor

                    MouseArea {
                        anchors.fill: parent
                        onClicked:  {
                            tagCloud.model.setProperty(index, "selected", !selected)
                        }
                    }
                }
            }
        }
        contentWidth: parent.width;
        contentHeight: flow.childrenRect.height

    }

    //optional - scroll indicator
    Rectangle {
        id: scrollIndicator
        anchors.right: flickable.right
        width:4
        height:flickable.visibleArea.heightRatio * flickable.height
        y:flickable.visibleArea.yPosition * flickable.height
        color:textColor
        opacity:0
        states: State {
            name: "ShowBars"
            when: flickable.movingVertically
            PropertyChanges { target:scrollIndicator; opacity: 1 }
        }
        transitions: Transition {
            NumberAnimation { properties: "opacity"; duration: 400 }
        }
    }
}
```

Example main.qml to test it out:

```
import QtQuick 1.0

Rectangle {
    id: main
    width: 360
    height: 200

    ListModel {
        id: photoTags

        //example items
        ListElement {
            name: "animals"
            score: 2
            selected: false
        }
        ListElement {
            name: "friends"
            score: 60
            selected: false
        }
        ListElement {
            name: "night"
            score: 80
            selected: false
        }
        ListElement {
            name: "party"
            score: 140
            selected: false
```


(C) Copyright Nokia 2013. All rights reserved.

```
        }
        ListElement {
            name: "people"
            score: 120
            selected: false
        }
        ListElement {
            name: "summer"
            score: 32
            selected: false
        }
        ListElement {
            name: "sun"
            score: 27
            selected: false
        }
        ListElement {
            name: "travel"
            score: 35
            selected: false
        }
        ListElement {
            name: "winter"
            score: 21
            selected: false
        }
    }

    TagCloud {
        model: photoTags
        width: parent.width
        height: parent.height
        baseColor: "black"
        textColor: "white"
        highlightColor: "steelblue"
        hightlightedTextColor: textColor
    }
}
```

You should see something like this in qmlviewer (I've added a few extra tags to those above to make it look more interesting):

How it works

Each displayed tag is a implemented as a Text element inside a Rectangle. We use the Repeater element to create a tag rectangle delegate for each element in the Data Model and use the Flow layout to arrange these items. Finally, the Flow is wrapped inside a Flickable so we can flick-scroll through the tags if we have more than will fit on screen at once.

Lets look at the inner Text component first:

```
    Text {
      id: textBlock
      text: name
      color: selected ? hightlightedTextColor : textColor;
      anchors.centerIn: parent

      property int itemScore: score
      onItemScoreChanged: {
         repeater.minScore = (repeater.minScore == undefined) ? itemScore: Math.min(itemScore, repeater.minScore)
         repeater.maxScore = Math.max(itemScore, repeater.maxScore)
         console.log(index + " minScore:" + repeater.minScore + " maxScore:" + repeater.maxScore)
      }
      property double scale:  ((itemScore - repeater.minScore)/(repeater.maxScore - repeater.minScore))
      font.pointSize:  Math.round(scale*16) + 10


      onHeightChanged: {
        flow.maxHeight = Math.max(height, flow.maxHeight)
      }

    }
```

The properties `name`, `selected` and `score` come from the data model item that we are displaying. `text` is "wired up" to `name`. We use the value of `selected` to choose which text color to use.

A local property, `scale`, is used to calculate `font.pointSize`. To work out the value of `scale`, we need 3 things: the `score` of the current item, and the maximum and minimum scores of all items in the Data Model.

The ListModel we are using is just a simple data structure and doesn't know the maximum and minimum scores of its contents, so we need to work this

(C) Copyright Nokia 2013. All rights reserved.

out for ourselves. We use the delegates to do this.

Here is how this works:

Inside the Repeater element we have `minScore` and `maxScore` properties.

```
    Repeater {
      id: repeater
      model: tagCloud.model
      property int minScore   //initially undefined
      property int maxScore:0
      //...
```

When each delegate is created its `score` property is set to the value from the model item. This triggers a property change signal, which we trap using the `onItemScoreChanged` function (due to QTBUG-17965 we can't set a handler directly on `score` so we create a new `itemScore` property which is bound to `score` and set a handler on that).

Inside `onItemScoreChanged` we can compare `score` against `repeater.minScore` and `repeater.maxScore` and change them as appropriate. `repeater.minScore` is initially undefined because we don't know what a sensible initial value would be. We test for `repeater.minScore` being undefined, which will only happen for the first delegate that is created, and assign the current `score` to it if so.

Once the Repeater has finished creating all the tag delegates the values of `minScore` and `maxScore` will contain the actual maximum and minimum values. The `scale` property of each item will automatically be recalculated each time `minScore` and `maxScore` change.

Note in the code above we are using a simple linear calculation to work out `scale`. An alterative which might give better results when dealing with large variations in score is

```
property double scale: (Math.log(score) - Math.log(flow.minScore))/(Math.log(flow.maxScore) - Math.log(flow.minScore))
```

`scale` determines the font size, which in turn determines the `height` of the Text element.

The height of the Rectangles must be bigger than the height of the Text items they contain. We could just set the Rectangle height to the Text height plus some padding, but it looks better (IMHO) if all the tag Rectangles in the cloud are the same height, i.e. big enough to hold the largest font size. To track the maximum height we have a `maxHeight` property in the Flow element, and we set this in each delegate's `onHeightChanged` function. This property is then used to work out the value of the Rectangle height property.

To make the tags clickable, we add a MouseArea inside the Rectangle, and we toggle the `selected` property inside the `onClicked` handler.

```
    MouseArea {
        anchors.fill: parent
        onClicked:  {
            tagCloud.model.setProperty(index, "selected", !selected)
        }
    }
```

Download

Here is a zip file of a Qt creator project containing TagCloud.qml and a sample main.qml qmltagcloud.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.