Filtering XmlListModel data

This article explains how XmlListModel data can be filtered by modifying list model's query property based on user input.

Introduction

XmlListModel offers an easy way to construct read-only model from XML data. Later that model can be presented with ListView. When there are lot of data entries in XML model, it is beneficial to let user search or filter the results.

Summary

In this example filtering is done by observing text property changes in QML TextInput element. Actual filtering in XmlLisMode side is done with simple XPath expression. XPath is a language that is crafted to navigate and find information in XML documents. As an example document, books list is used. It is available here: books.xml

Overview of books.xml

Books.xml has very simple structure. Root element is `catalog` and it can have multiple `book` child elements.

```
<?xml version="1.0"?>
<catalog>
   <book id="bk000">
      <author>Author</author>
      <title>Book Title</title>
      <genre>Genre</genre>
      <price>23</price>
      <publish_date>2011-11-16</publish_date>
      <description>Overview to XML document.</description>
   </book>
</catalog>
```

XPath queries

In the following QML source listing, pay attention to queries. Initially one has to specify a base query for XmlListModel. In this case we want to have all book elements from the books.xml. We can do this by XPath expression as follows:

```
query: "/catalog/book"
```

Next up is defining attributes for model items. This is done by specifying XmlRole Objects. Role name is used later on, in ListView delegate, to access the data in model items.

```
XmlRole {id: xmlRole; name: "author"; query: "author/string()"; }


XmlRole {name: "title"; query: "title/string()"; }
```

Filtering is done based on user input. This is the most complex XPath expression in this example. What we want to achieve is to compare if book author string contains the text user gave. Luckily XPath offers `contains(str1,str2)` funtion that can be used to check if a string contains another string. Contains-function is case sensitive, so we must convert author name and user input to lower case with `lower-case(str)` function. Note that for filtering we compare user input only to author element text. This is restricted with: `child::author`. We could use "." operator, but would cause entire book element text, with attributes and descendants, to be used in filtering. That would result into false positives.

```
xmlModel.query = "/catalog/book[contains(lower-case(child::author),lower-
case(\""+filter.text+"\"))]";
```

Last but not least, `xmlModel.reload()` is called to reload the model, now by using the new filtering query.

Source

```
import QtQuick 1.0

Rectangle {
    width: 360
    height: 480

    TextInput {
        id: filter
        text:"Author..."
        anchors.top: parent.top
        anchors.topMargin: 20
        width: parent.width

        //if user is typing fast, we don't want to search on every key-press
        Timer{
           id: filterTimer
           interval:500
           running: false
           repeat: false

           onTriggered: {
               console.log("triggered");
               xmlModel.query = "/catalog/book[contains(lower-case(child::author),lower-
case(\""+filter.text+"\"))]";
               xmlModel.reload();
           }
        }
        onTextChanged:{
            console.log(filter.text);
            if(filterTimer.running){
                console.log("restarted");
                filterTimer.restart()
            }else{
                console.log("started");
                filterTimer.start()
            }
        }
    }
    XmlListModel{
        id: xmlModel
        source: "books.xml"
        query: "/catalog/book"
        XmlRole {id: xmlRole; name: "author"; query: "author/string()";  }
        XmlRole {name: "title"; query: "title/string()"; }
        onStatusChanged: {
            console.log("Count: "+ xmlModel.count);
        }
    }
    ListView {
        anchors.top: filter.bottom
        anchors.topMargin: 20
        anchors.bottom: parent.bottom
        width:parent.width
        model: xmlModel
```

```
        delegate: Text { text: author + ": " + title }
    }
}
```