How to create a Context Menu with QML

This article shows **how to create a reusable Context Menu UI component with QML**. A context menu is a UI element that appears upon user interaction (usually a long-click) and shows a set of options that are related to the clicked item or to the current state of the applicaiton.

The component structure

The context menu must appear **above any other UI element**, allowing the user to choose one of the presented options, or to hide the menu by clicking on the area outside the menu itself.

So, the component defines two main UI elements:

- a **root Rectangle**, that is used to cover the whole application UI
- an **inner Rectangle**, that contains the context menu options

The root rectangle is colored with a **semi-trasparent color**, in order to partially hide the application UI while the context menu is shown.

```
Rectangle {
    id: menuRoot

    opacity: 0
    anchors.fill: parent
    color: "#80000000"

    Rectangle {
 id: menu

 color: "#336699"
    }
}
```

Using a ListView for the options

Since the context menu must show a set of text items, a **ListView represents a good choice** to handle it. So, a ListView is added to the menu Rectangle, and two properties are added to the component:

- a **model** property that holds the model used by the ListView
- a **itemFontSize** int property that holds the font size used by the context menu options

```
Rectangle {
    id: menuRoot

    [...]

    property ListModel model
    property int itemFontSize : 25

    Rectangle {
 id: menu

 color: "#336699"

 ListView {
     id: menuList

     highlight: Rectangle { color: "lightsteelblue" }
     clip: true
     model: menuRoot.model;
     anchors.fill: parent

     delegate: Rectangle {
  id: listViewItem

  color: "transparent"
  border {width: 1;color: "black"}
  width: menu.width - 1
  height: itemText.height

  Text {
      id: itemText
      text: name
      width: listViewItem.width
      font.pointSize: itemFontSize
      elide: Text.ElideRight
  }
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }
 }
    }
}
```

The delegate above assumes that each ListElement of the ListModel has a '"name property**, that is used by the delegate's Text element**

Calculating the menu size

The **menu size** depends on two different factors:

- the width and height needed to display the menu options
- the width and height of the QML document's root (so, the **available space**)

Depending on these two factors, it is possible to **calculate the optimal width and height** for the menu.

First, let's **create a JavaScript "ContextualMenu.js" file**, and import it in the Component's qml file as follows:

```
import "ContextualMenu.js" as CtxMenu
```

Then, a **getDocRoot() utility function** is defined, that returns the root element of the QML document:

```
var docRoot = null;

function getDocRoot()
{
    if(!docRoot)
    {
 docRoot = menuRoot.parent;

 while(docRoot.parent)
 {
     docRoot = docRoot.parent;
 }
    }
    return docRoot;
}
```

Then, it is possible to define the following **getMenuFunction()** function, that returns the **width necessary to display all the menu options** or, if this is larger than the root element's width, the width of the root element minus a predefined padding. The padding is necessary to allow the user to click outside the context menu to dispose it.

The function creates a **temporary Text element** for each menu option, calculates the width of the Text element, and then destroy it. The docRootWidth variable is used to check if the **QML document's root width has changed**, for instance after a display rotation: in this case, the menu width is calculated again to better fit the new screen space.

```
var menuPadding = 64;
var menuWidth = -1;
var rowHeight = -1;
var docRootWidth = -1;

function getMenuWidth()
{
    if(menuWidth == -1 || (getDocRoot().width != docRootWidth))
    {
 docRootWidth = getDocRoot().width;

 for(var i = 0; i < model.count; i++)
 {
     var textItem = Qt.createQmlObject('import Qt 4.7; Text { font.pointSize:' + itemFontSize + '; text: "' + model.get(i).name + '"
  menuList);

     menuWidth = Math.max(menuWidth, textItem.width);
     rowHeight = textItem.height;

     textItem.destroy();
 }
    }
    menuWidth = Math.min(menuWidth, getDocRoot().width - menuPadding);

    return menuWidth;
}
```

Similarly, a **getMenuHeight()** function is defined:

```
function getMenuHeight()
{
    return Math.min(rowHeight * model.count + 1, getDocRoot().height - menuPadding);
}
```

The above functions are then used to properly **bind the menu width and height properties**, as follows:

```
Rectangle {
    id: menu

    width: CtxMenu.getMenuWidth() + 1
    height: CtxMenu.getMenuHeight()

    [...]
}
```

Showing and hiding the menu

(C) Copyright Nokia 2013. All rights reserved.

First, a **new State** is defined, to represent the menu in its visible state. A Transition is also used, in order to **animate the State transition**.

```
Rectangle {
    id: menuRoot

    [...]

    states: State {
 name: "visible"
 PropertyChanges {target: menuRoot; opacity: 1}
    }

    transitions: Transition {
 NumberAnimation {
     target:  menuRoot
     properties: "opacity"
     duration: 250
 }
    }
}
```

Rearranging the component

Now, before continuing, it is necessary to see for a moment **how the component will be used**. The below example shows a Context menu associated with a red Rectangle:

```
Rectangle {
    id: myRectangle
    color: "red"

    ContextualMenu {
 model: ListModel {
     ListElement {name: "Red Item 0"}
     ListElement {name: "Red Item 1"}
     ListElement {name: "Red Item 2"}
 }
    }
}
```

The context menu must be shown when the user performs a long click action on the **myRectangle** Rectangle, so a **MouseArea must be associated** with it in order to catch user interactions. On the other side, the context menu element must be associated with the QML document's root, since it must be displayed **above any other UI elements**: this means that the component must rearrange itself in order to properly display.

First, the component defines the MouseArea that will be associated with the component's parent element:

```
Rectangle {
    id: menuRoot

    [...]

    // intercepts press-and-hold interactions on the menu parent, that show the menu itself
    MouseArea {
 id: menuMouseArea
 anchors.fill: parent
 onPressAndHold: CtxMenu.showMenu(mouse)
    }
}
```

Then, a JavaScript function is defined, in order to perform the following operations:

- move the **menuMouseArea** to the component's parent element
- move the **component itself to the QML document's root**

```
function initializeMenu()
{
    // move the mouse area to the menu's parent element
    menuMouseArea.parent = menuRoot.parent;

    // move the contextual menu to the document's root
    menuRoot.parent = getDocRoot();
}
```

The above function will be called once the component had been completely loaded, using the Component.onCompleted signal:

```
Rectangle {
    id: menuRoot

    [...]

    Component.onCompleted: CtxMenu.initializeMenu()
}
```

Performing the state change

When the user long-clicks the component's parent element (the red Rectangle in the example above), the context menu must be visually placed above the element itself: this is performed by **catching the x and y coordinates of the originating mouse event**, and using them together with the menu width and height to place the menu itself above the parent element and within the document's root boundaries, as shown by the **showMenu()** JavaScript function below:

```
function showMenu(mouse)
{
    var mouseX = mouse.x;
    var mouseY = mouse.y;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    var absoluteX = mouseX;
    var absoluteY = mouseY;
    var element = menuMouseArea.parent;

    while(element != null)
    {
 absoluteX += element.x;
 absoluteY += element.y;

 element = element.parent;
    }

    menu.x = Math.max(0, Math.min(absoluteX - menu.width / 2, menuRoot.width - menu.width));
    menu.y = Math.max(0, Math.min(absoluteY - menu.height / 2, menuRoot.height - menu.height));

    menuRoot.state = "visible";
    menuList.forceActiveFocus();
}
function hideMenu()
{
    menuRoot.state = "";
}
```

The showMenu() function also **forces the focus to be moved to the ListView** (a FocusScope itself), so that it can appropriately handle the user interactions (for instance, the Key events).

Catching clicks outside the menu area

If the user clicks outside the context menu area, the context menu itself should be disposed: this is accomplished by defining another MouseArea in the component's root Rectangle, as follows:

```
Rectangle {
    id: menuRoot

    [...]

    // intercepts clicks outside the menu area, that hide the menu itself
    MouseArea {
 anchors.fill: parent
 onClicked: CtxMenu.hideMenu()
    }
}
```

Selecting a menu option

Once the menu is shown, the component must be able to notify which option was selected by the user. In order to do this, the component declares an **itemClicked** signal as follows:

```
Rectangle {
    id: menuRoot

    signal itemClicked(int index)

    [...]
}
```

This signal is called by the ListView when the user selects one of its items, with **both mouse or key interactions**:

```
ListView {
    id: menuList

    [...]

    delegate: Rectangle {
 id: listViewItem

 [...]

 MouseArea {
     anchors.fill: parent
     onClicked: {
  itemClicked(index);
  CtxMenu.hideMenu();
     }
 }
 Keys.onReturnPressed: {
     itemClicked(index);
     CtxMenu.hideMenu();
 }
    }
}
```

Using the Context Menu

The component can be easily used, as already seen above, by appending a ContextMenu item to the element that must be associated to the menu.

```
Rectangle {
    color: "red"
    x: 250
    y: 250
    width: 100
    height: 100
```


(C) Copyright Nokia 2013. All rights reserved.

```
    ContextualMenu {
  onItemClicked: console.log("clicked red item: " + index)

  model: ListModel {
      ListElement {name: "Red Item 0"}
      ListElement {name: "Red Item 1"}
      ListElement {name: "Red Item 2"}
  }
     }
 }
```

A video of this component in action can be sen below. The media player is loading...

Related content

The Qt Creator project containing the code presented in this article is available here: File:QMLContextualMenu.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.