Author: dailey.dai@openthinks.com

Qt Quick Control

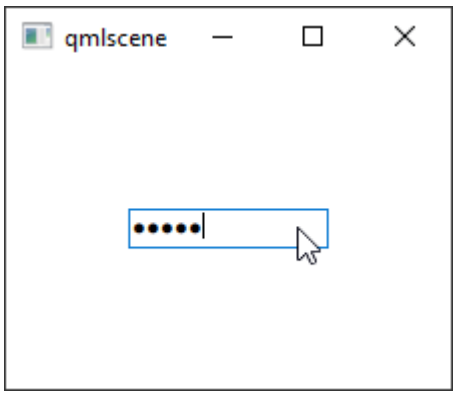
Line Editor

TextInput

The `TextInput` type displays a single line of editable plain text.

`TextInput` is used to accept a line of text input. Input constraints can be placed on a `TextInput` item (for example, through a validator or `inputMask`, and setting `echoMode` to an appropriate value enables `TextInput` to be used for a password input field.

```
import QtQuick 2.0
Item{
    Rectangle{
        border.width: 1
        border.color: "#147fd4";
        anchors.centerIn: parent;
        width:100;height:20;
        TextInput{
            width:100;height:20;
            anchors.fill:parent;anchors.margins: 2;
            echoMode:TextInput.Password;
        }
    }
}
```

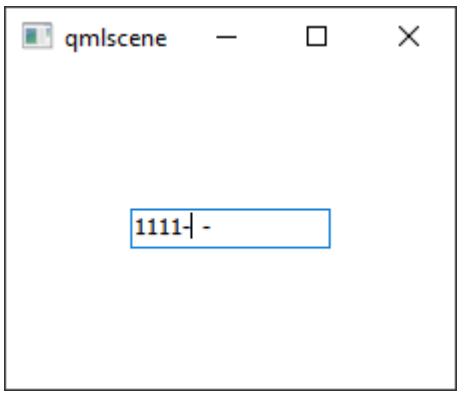


`echoMode` : Specifies how the text should be displayed in the `TextInput`

- `TextInput.Normal` Displays the text as it is. (Default)
- `TextInput.Password` Displays platform-dependent password mask characters instead of the actual characters.
- `TextInput.NoEcho` Displays nothing.

- `TextInput.PasswordEchoOnEdit` Displays characters as they are entered while editing, otherwise identical to `TextInput.Password`.

```
import QtQuick 2.0
Item{
    Rectangle{
        border.width: 1
        border.color: "#147fd4";
        anchors.centerIn: parent;
        width:100;height:20;
        TextInput{
            width:100;height:20;
            anchors.fill:parent;anchors.margins: 2;
            inputMask: "0000-00-00";
        }
    }
}
```



`inputMask` : Allows you to set an input mask on the `TextInput`, restricting the allowable text inputs.

Character	Meaning
A	ASCII alphabetic character required. A-Z, a-z.
a	ASCII alphabetic character permitted but not required.
N	ASCII alphanumeric character required. A-Z, a-z, 0-9.
n	ASCII alphanumeric character permitted but not required.
X	Any character required.
x	Any character permitted but not required.
9	ASCII digit required. 0-9.
0	ASCII digit permitted but not required.
D	ASCII digit required. 1-9.
d	ASCII digit permitted but not required (1-9).
#	ASCII digit or plus/minus sign permitted but not required.
H	Hexadecimal character required. A-F, a-f, 0-9.
h	Hexadecimal character permitted but not required.
B	Binary character required. 0-1.
b	Binary character permitted but not required.
>	All following alphabetic characters are uppercased.
<	All following alphabetic characters are lowercased.
!	Switch off case conversion.
[ ] { }	Reserved.
\	Use \ to escape the special characters listed above to use them as separators.

```
import QtQuick 2.0
TextInput{
    validator: IntValidator{bottom: 11; top: 31;}
    focus: true
}
```

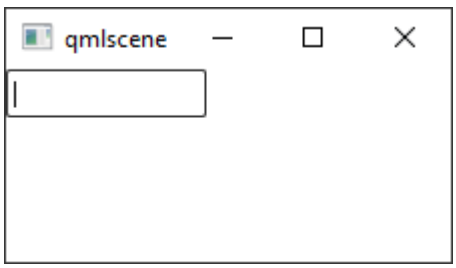
`validator` : When a validator is set the `TextInput` will only accept input which leaves the text property in an acceptable or intermediate state.

- `IntValidator` : provides a validator for integer values
- `DoubleValidator` : provides a validator for non-integer numbers
- `RegExpValidator` : provides a validator, which counts as valid any string which matches a specified regular expression

TextField

Same as `TextInput` mostly, but `TextField` could define background.

```
import QtQuick 2.0
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4
Item{
    TextField{
        style:TextFieldStyle{
            textColor:"black"
            background:Rectangle{
                radius:2
                implicitWidth: 100
                implicitHeight: 24
                border.color:"#333"
                border.width:1
            }
        }
    }
}
```



Text Block

TextEdit

The `TextEdit` item displays a block of editable, formatted text.

```
import QtQuick 2.0

TextEdit {
    width: 240
    text: "<b>Hello</b> <i>World!</i>"
    font.family: "Helvetica"
    font.pointSize: 20
    color: "blue"
    focus: true
    textFormat:TextEdit.RichText
}
```

`textFormat` : The way the text property should be displayed.

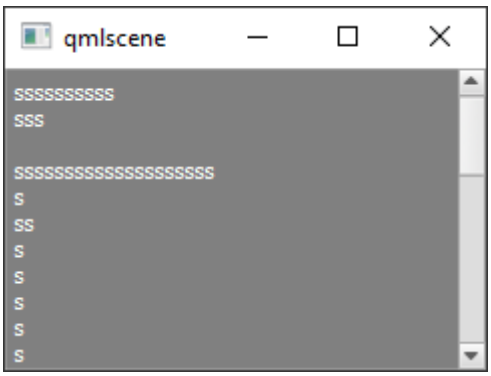
- TextEdit.AutoText : automatically determine whether the text should be treated as rich text
- TextEdit.PlainText : (default)
- TextEdit.RichText

TextArea

Same as `TextEdit` mostly, but `TextArea` could define background and support scroll

```
import QtQuick 2.0
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4

TextArea{
    wrapMode:TextArea.WordWrap;
    style:TextAreaStyle{
        backgroundColor: "gray"
        textColor: "white"
        selectionColor: "steelblue"
        selectedTextColor: "#a00000"
    }
}
```

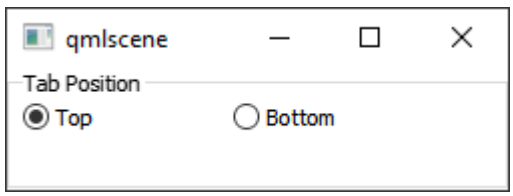


ExclusuveGroup

`ExclusiveGroup` can contain several `Action` items, and those will automatically get their `Action::exclusiveGroup` property assigned.

```
import QtQuick 2.0
import QtQuick.Layouts 1.2
import QtQuick.Controls 1.4
GroupBox {
    id: group2
    title: qsTr("Tab Position")
    Layout.fillWidth: true
```

```
    RowLayout {
        ExclusiveGroup { id: tabPositionGroup }
        RadioButton {
            id: topButton
            text: qsTr("Top")
            checked: true
            exclusiveGroup: tabPositionGroup
            Layout.minimumWidth: 100
        }
        RadioButton {
            id: bottomButton
            text: qsTr("Bottom")
            exclusiveGroup: tabPositionGroup
            Layout.minimumWidth: 100
        }
    }
}
```

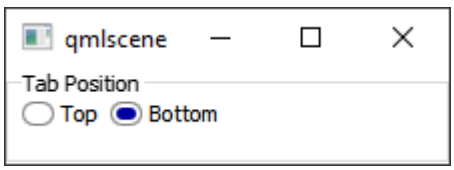


RadioButton

A `RadioButton` is an option button that can be switched on (checked) or off (unchecked). Radio buttons typically present the user with a "one of many" choices. In a group of radio buttons, only one radio button can be checked at a time; if the user selects another button, the previously selected button is switched off.

```
import QtQuick 2.0
import QtQuick.Layouts 1.2
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4
GroupBox {
     title: "Tab Position"
     Component{
        id:radioStyle
        RadioButtonStyle{
            indicator: Rectangle{
                implicitWidth: 16;implicitHeight: 12;radius: 6;
                border.color: control.hovered?"darkblue":"gray";border.width:1;
                Rectangle{
                    anchors.fill: parent;visible: control.checked;
                    color:"#0000A0";radius: 5;
                    anchors.margins: 3;
                }
            }
        }
     }
     RowLayout {
         ExclusiveGroup { id: tabPositionGroup }
```

```
        RadioButton {
            text: "Top"
            checked: true
            exclusiveGroup: tabPositionGroup
            style: radioStyle
        }
        RadioButton {
            text: "Bottom"
            exclusiveGroup: tabPositionGroup
            style: radioStyle
        }
    }
}
```

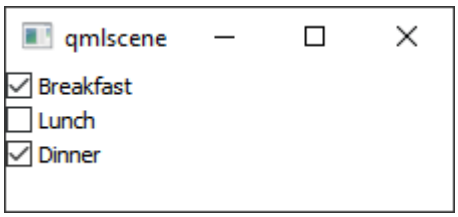


CheckBox

A `CheckBox` is an option button that can be toggled on (checked) or off (unchecked). Checkboxes are typically used to represent features in an application that can be enabled or disabled without affecting others.

```
import QtQuick 2.0
import QtQuick.Layouts 1.2
import QtQuick.Controls 1.4

Column {
    CheckBox {
        text: qsTr("Breakfast")
        checked: true
    }
    CheckBox {
        text: qsTr("Lunch")
    }
    CheckBox {
        text: qsTr("Dinner")
        checked: true
    }
}
```

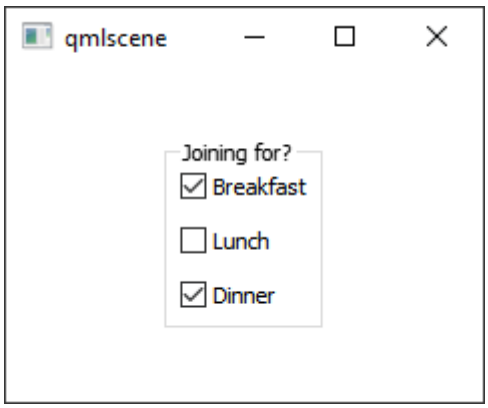


GroupBox

A group box provides a frame, a title on top and displays various other controls inside itself. Group boxes can also be checkable

```
import QtQuick 2.0
import QtQuick.Layouts 1.2
import QtQuick.Controls 1.4
Item{
    GroupBox {
         title: "Joining for?"
         anchors.centerIn: parent;
         Column {
             spacing: 10

             CheckBox {
                 text: "Breakfast"
                 checked: true
             }
             CheckBox {
                 text: "Lunch"
                 checked: false
             }
             CheckBox {
                 text: "Dinner"
                 checked: true
             }
         }
     }
}
```



ComboBox

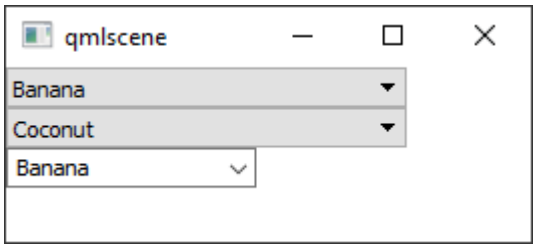
Provides a drop-down list functionality.

Add items to the `ComboBox` by assigning it a `ListModel`, or a list of strings to the `model` property.

```
import QtQuick 2.0
import QtQuick.Controls 1.4

Column{
    ComboBox {
```

```
        width: 200
        model: [ "Banana", "Apple", "Coconut" ]
    }
    ComboBox {
        currentIndex: 2
        model: ListModel {
            id: cbItems
            ListElement { text: "Banana"; color: "Yellow" }
            ListElement { text: "Apple"; color: "Green" }
            ListElement { text: "Coconut"; color: "Brown" }
        }
        width: 200
        onCurrentIndexChanged: console.debug(cbItems.get(currentIndex).text + ", " +
cbItems.get(currentIndex).color)
    }
    ComboBox {
        editable: true
        model: ListModel {
            id: model
            ListElement { text: "Banana"; color: "Yellow" }
            ListElement { text: "Apple"; color: "Green" }
            ListElement { text: "Coconut"; color: "Brown" }
        }
        onAccepted: {
            if (find(currentText) === -1) {
                model.append({text: editText})
                currentIndex = find(editText)
            }
        }
    }
}
```

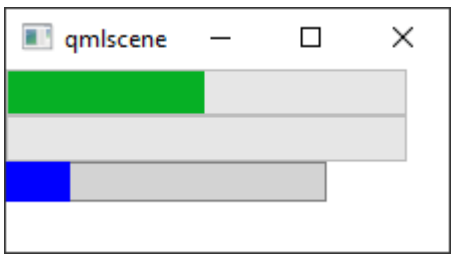


ProgressBar

The `ProgressBar` is used to give an indication of the progress of an operation. `value` is updated regularly and must be between `minimumValue` and `maximumValue`.

```
import QtQuick 2.0
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4
Column {
    ProgressBar {
        value: 0.5
```

```
    }
    ProgressBar {
        indeterminate: true
    }
    ProgressBar{
        minimumValue: 0;maximumValue: 1;
        value:0.2;
        width: 160;height: 20;
        style: ProgressBarStyle{
            background: Rectangle{
                implicitWidth: 200;implicitHeight: 20;
                border.width: 1;border.color: control.hovered?"green":"gray";
                color:"lightgray";
            }
            progress: Rectangle{
                color: "blue";
            }
        }
    }
}
```



TabView

A control that allows the user to select one of multiple stacked items

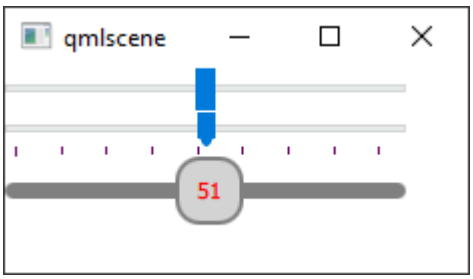
```
import QtQuick 2.0
import QtQuick.Controls 1.4
TabView {
     Tab {
         title: "Red"
         Rectangle { color: "red" }
     }
     Tab {
         title: "Blue"
         Rectangle { color: "blue" }
     }
     Tab {
         title: "Green"
         Rectangle { color: "green" }
     }
 }
```

Slider

Provides a vertical or horizontal slider control.

The slider is the classic control for providing a bounded value. It lets the user move a slider handle along a horizontal or vertical groove and translates the handle's position into a value within the legal range.

```
import QtQuick 2.0
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4
Column {
    Slider {
        value: 0.5
    }
    Slider {
        value: 0.5
        tickmarksEnabled: true
    }
    Slider{
        minimumValue: 0;maximumValue: 100;
        stepSize:1
        style: SliderStyle{
            groove: Rectangle{
                implicitWidth: 200;implicitHeight: 8;
                color: "gray";radius: 8;
            }
            handle: Rectangle{
                anchors.centerIn: parent;
                color: control.pressed?"white":"lightgray";
                border.color: "gray";border.width: 2;
                width: 34;height: 34;radius: 12;
                Text{
                    anchors.centerIn: parent
                    text:control.value
                    color:"red"
                }
            }
        }
    }
}
```

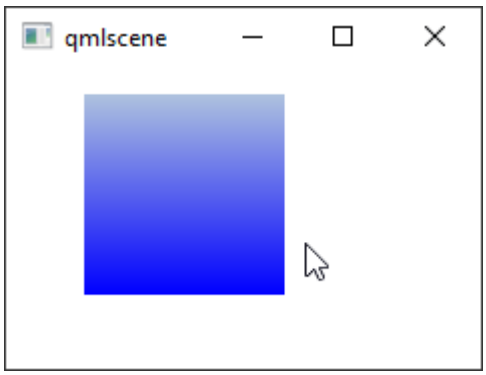


Flickable

Provides a surface that can be "flicked".

The Flickable item places its children on a surface that can be dragged and flicked, causing the view onto the child items to scroll.

```
import QtQuick 2.0
Flickable {
    width: 150
    height: 150
    contentWidth: 300
    contentHeight: 300
    rebound: Transition {
        NumberAnimation {
            properties: "x,y"
            duration: 1000
            easing.type: Easing.OutBounce
        }
    }
    Rectangle {
        width: 100
        height: 100
        gradient: Gradient {
            GradientStop {
                position: 0.0
                color: "lightsteelblue"
            }
            GradientStop {
                position: 1.0
                color: "blue"
            }
        }
    }
}
```



Screen

The Screen attached object provides information about the Screen an Item or Window is displayed on.

The Screen attached object is valid inside Item or Item derived types, after component completion. Inside these items it refers to the screen that the item is currently being displayed on.

```
import QtQuick 2.0
import QtQuick.Window 2.3

Item {
    Text{
        id:info
    }
    function showScreenInfo(){
        var screenMap ={
            "desktopAvailableWidth":Screen.desktopAvailableWidth,
            "desktopAvailableHeight":Screen.desktopAvailableHeight,
            "devicePixelRatio":Screen.devicePixelRatio
        };
        var sInfo="";
        for(var key in screenMap){
            sInfo = sInfo + key+"="+screenMap[key]+"\n";
        }
        info.text=sInfo;
    }
    Component.onCompleted: {
       showScreenInfo();
    }
}
```