Author: dailey.dai@openthinks.com

# Qt Quick Layout

There are two categories for Qt Quick layout.

1. Item Positioner
2. Item Layout manager

## Positioner

Positioner items are container items that manage the positions of items in a declarative user interface.

Positioner do not change the items in it.

A set of standard positioners are provided in the basic set of Qt Quick graphical types:

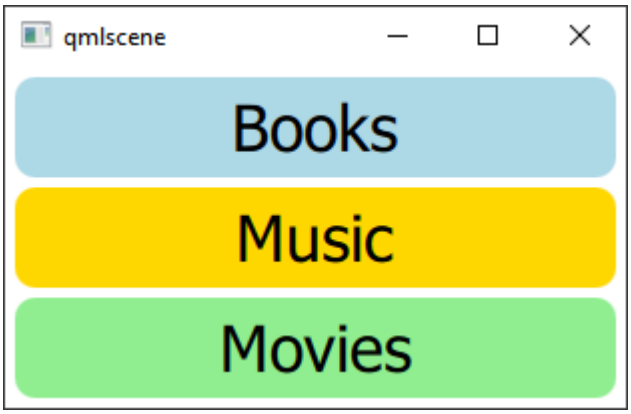
| Name | Description |
| --- | --- |
| Column | Positions its children in a column |
| Flow | Positions its children side by side, wrapping as necessary |
| Grid | Positions its children in grid formation |
| Row | Positions its children in a row |

## Column

Column items are used to vertically arrange items.

```
import QtQuick 2.0
Item {
    width: 310; height: 170
    Column {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
        spacing: 5
        Rectangle { color: "lightblue"; radius: 10.0
                    width: 300; height: 50
                    Text { anchors.centerIn: parent
                           font.pointSize: 24; text: "Books" } }
        Rectangle { color: "gold"; radius: 10.0
                    width: 300; height: 50
                    Text { anchors.centerIn: parent
                           font.pointSize: 24; text: "Music" } }
        Rectangle { color: "lightgreen"; radius: 10.0
                    width: 300; height: 50
                    Text { anchors.centerIn: parent
                           font.pointSize: 24; text: "Movies" } }
```

```
    }
}
```




## Row

Row items are used to horizontally arrange items.

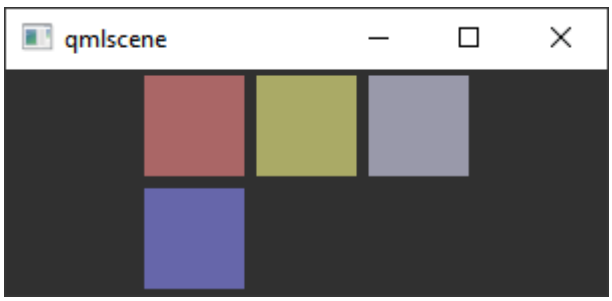
```
import QtQuick 2.0
Rectangle {
    width: 320; height: 110
    color: "#c0c0c0"
    Row {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
        spacing: 5
        Rectangle { width: 100; height: 100; radius: 20.0
                    color: "#024c1c" }
        Rectangle { width: 100; height: 100; radius: 20.0
                    color: "#42a51c" }
        Rectangle { width: 100; height: 100; radius: 20.0
                    color: "white" }
    }
}
```


## Grid

Grid items are used to place items in a grid or table arrangement.

```
import QtQuick 2.0
Rectangle {
    width: 300; height: 112
    color: "#303030"
    Grid {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
        columns: 2
        //columns:3
        //rows:2
        //flow:Grid.TopToBottom
        spacing: 6
        Rectangle { color: "#aa6666"; width: 50; height: 50 }
        Rectangle { color: "#aaaa66"; width: 50; height: 50 }
        Rectangle { color: "#9999aa"; width: 50; height: 50 }
        Rectangle { color: "#6666aa"; width: 50; height: 50 }
    }
}
```


when `columns=3`

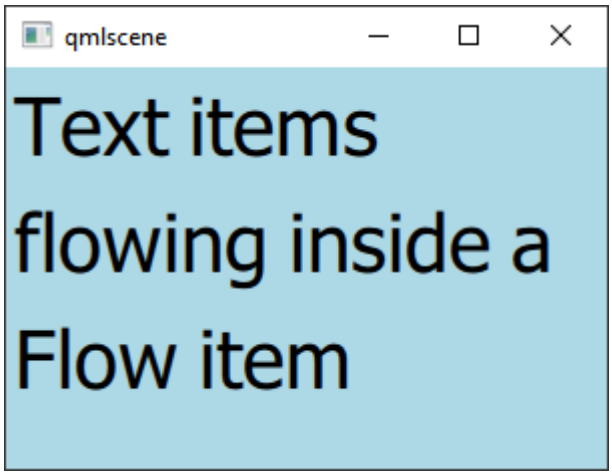


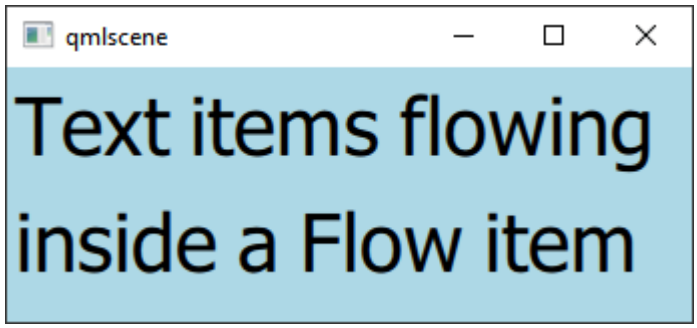

when `flow=Grid.TopToButton`


## Flow

Flow items are used to place items like words on a page, with rows or columns of non-overlapping items.

```
import QtQuick 2.0
Rectangle {
    color: "lightblue"
    width: 300; height: 200
    Flow {
        anchors.fill: parent
        anchors.margins: 4
        spacing: 10
        Text { text: "Text"; font.pixelSize: 40 }
        Text { text: "items"; font.pixelSize: 40 }
        Text { text: "flowing"; font.pixelSize: 40 }
        Text { text: "inside"; font.pixelSize: 40 }
        Text { text: "a"; font.pixelSize: 40 }
        Text { text: "Flow"; font.pixelSize: 40 }
        Text { text: "item"; font.pixelSize: 40 }
    }
}
```




## Nested positioner

```
import QtQuick 2.0
Rectangle {
    width: 200; height: 200
    color: "#c0c0c0"
    Row {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
```

```
        spacing: 5
        Column {
            spacing: 5
            Rectangle { color: "lightblue"; radius: 10.0
                        width: 50; height: 50
                        Text { anchors.centerIn: parent
                               font.pointSize: 24; text: "1-1" } }
            Rectangle { color: "gold"; radius: 10.0
                        width: 50; height: 50
                        Text { anchors.centerIn: parent
                               font.pointSize: 24; text: "1-2" } }
            Rectangle { color: "lightgreen"; radius: 10.0
                        width: 50; height: 50
                        Text { anchors.centerIn: parent
                               font.pointSize: 24; text: "1-3" } }
        }
        Column {
            spacing: 5
            Rectangle { color: "lightblue"; radius: 10.0
                        width: 50; height: 50
                        Text { anchors.centerIn: parent
                               font.pointSize: 24; text: "2-1" } }
            Rectangle { color: "gold"; radius: 10.0
                        width: 50; height: 50
                        Text { anchors.centerIn: parent
                               font.pointSize: 24; text: "2-2" } }
            Rectangle { color: "lightgreen"; radius: 10.0
                        width: 50; height: 50
                        Text { anchors.centerIn: parent
                               font.pointSize: 24; text: "2-3" } }
        }
    }
}
```

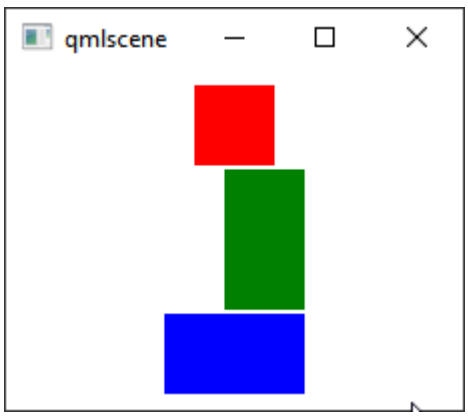

## Layout manager

Qt Quick Layouts are a set of QML types used to arrange items in a user interface. In contrast to positioners, Qt Quick Layouts can also resize their items. This makes them well suited for resizable user interfaces. Since layouts are items they can consequently be nested.

## ColumnLayout

Identical to GridLayout, but having only one column.

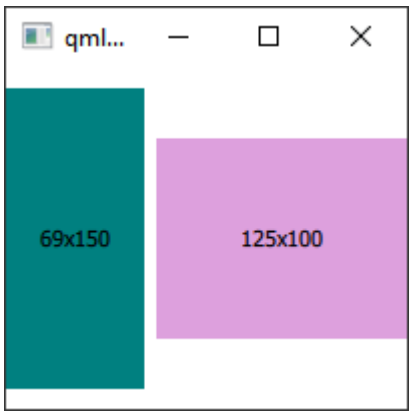
```
import QtQuick 2.0
import QtQuick.Layouts 1.2
Item {
    width: 200; height: 170
    ColumnLayout{
        spacing: 2
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
        Rectangle {
            Layout.alignment: Qt.AlignCenter
            color: "red"
            Layout.preferredWidth: 40
            Layout.preferredHeight: 40
        }
        Rectangle {
            Layout.alignment: Qt.AlignRight
            color: "green"
            Layout.preferredWidth: 40
            Layout.preferredHeight: 70
        }
        Rectangle {
            Layout.alignment: Qt.AlignBottom
            Layout.fillHeight: true
            color: "blue"
            Layout.preferredWidth: 70
            Layout.preferredHeight: 40
        }
    }
}
```

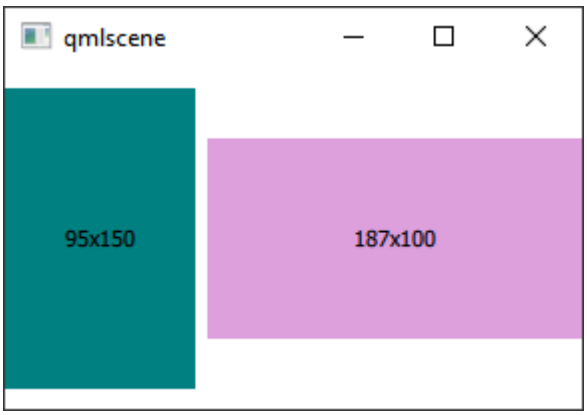



## RowLayout

Identical to GridLayout, but having only one row

```
import QtQuick 2.0
import QtQuick.Layouts 1.2
Item {
    width: 200; height: 170
    RowLayout {
        id: layout
        anchors.fill: parent
        spacing: 6
        Rectangle {
            color: 'teal'
            Layout.fillWidth: true
            Layout.minimumWidth: 50
            Layout.preferredWidth: 100
            Layout.maximumWidth: 300
            Layout.minimumHeight: 150
            Text {
                anchors.centerIn: parent
                text: parent.width + 'x' + parent.height
            }
        }
        Rectangle {
            color: 'plum'
            Layout.fillWidth: true
            Layout.minimumWidth: 100
            Layout.preferredWidth: 200
            Layout.preferredHeight: 100
            Text {
                anchors.centerIn: parent
                text: parent.width + 'x' + parent.height
            }
        }
    }
}
```



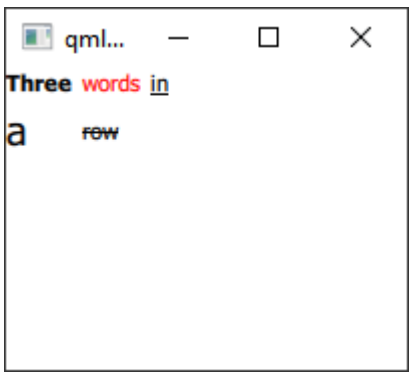



## GridLayout

Provides a way of dynamically arranging items in a grid. If the GridLayout is resized, all items in the layout will be rearranged.

```
import QtQuick 2.0
import QtQuick.Layouts 1.2
Item {
    width: 200; height: 150
    GridLayout {
        id: grid
        columns: 3

        Text { text: "Three"; font.bold: true; }
        Text { text: "words"; color: "red" }
        Text { text: "in"; font.underline: true }
        Text { text: "a"; font.pixelSize: 20 }
        Text { text: "row"; font.strikeout: true }
    }
}
```




## StackLayout

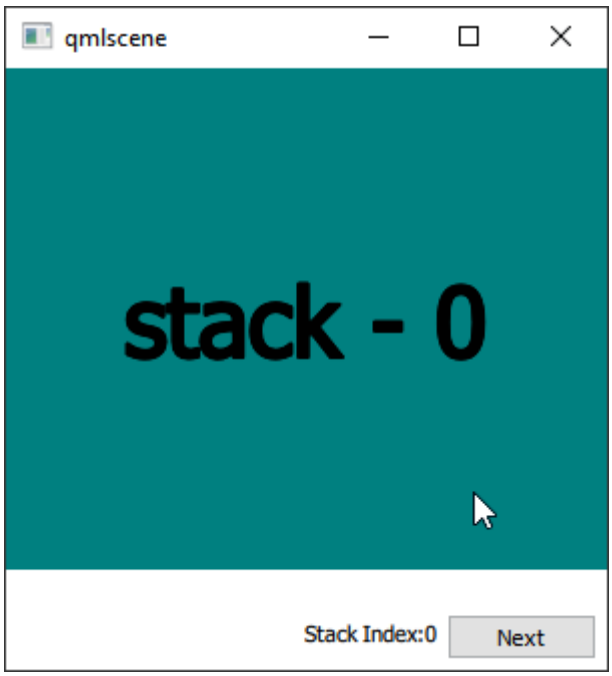
Stack of items where only one item is visible at a time.

The current visible item can be modified by setting the currentIndex property. The index corresponds to the order of the StackLayout's children.

```
import QtQuick 2.0
```

```
import QtQuick.Layouts 1.3
import QtQuick.Controls 1.2
Item {
    width: 300; height: 300;
    Rectangle{
        width:300;
        height:250;
        StackLayout {
            id: layout
            anchors.fill: parent
            currentIndex: 1
            Rectangle {
                color: 'teal'
                implicitWidth: 300
                implicitHeight: 200
                Text{
                    anchors.centerIn: parent;
                    text:"stack - 0";
                    font.pixelSize: 48
                    style: Text.Outline
                }
            }
            Rectangle {
                color: 'plum'
                implicitWidth: 300
                implicitHeight: 200
                Text{
                    anchors.centerIn: parent;
                    text:"stack - 1";
                    font.pixelSize: 48
                    style: Text.Outline
                }
            }
        }
    }
    Text{
        anchors.bottom: nextButton.baseline;
        anchors.right: nextButton.left;
        anchors.rightMargin: 5;
        text: "Stack Index:"+layout.currentIndex;
    }
    Button{
        id:nextButton;
        anchors.bottom: parent.bottom;
        anchors.right: parent.right;
        anchors.bottomMargin: 5;
        anchors.rightMargin: 5;
        text:"Next";
        onClicked: {
            layout.currentIndex= (layout.currentIndex+1) % layout.count;
        }
    }
}
```


qmlscene
stack - 1
Stack Index:1
Next


qmlscene
stack - 0
Stack Index:0
Next