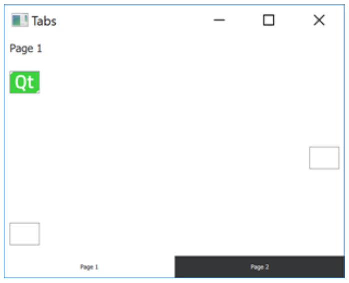
# Creating a Qt Quick Application

This tutorial uses built-in QML types and illustrates basic concepts of Qt Quick. For more information about the UI choices you have, see User Interfaces.

This tutorial describes how to use Qt Creator to implement Qt Quick states and transitions. We create an application that displays a Qt logo that moves between three rectangles on the page when you click them.



For more information about using Qt Quick Designer, see Developing Qt Quick Applications.

For examples of using Qt Quick Controls 2, see Qt Quick Controls 2 Examples.

## Creating the Project

1. Select **File** > **New File or Project** > **Application** > **Qt Quick Application - Swipe** > **Choose**.
2. In the **Name** field, enter a name for the application.
3. In the **Create in** field, enter the path for the project files, and then select **Next** (or **Continue** on macOS).
4. In the **Build system** field, select the build system to use for building and running the project: qmake, CMake, or Qbs.
5. In the **Qt Quick Controls 2 Style** field, select one of the predefined UI styles to use, and then select **Next**.
6. Select kits for the platforms that you want to build the application for. To build applications for mobile devices, select kits for Android ARM and iPhone OS, and click **Next**.

   **Note:** Kits are listed if they have been specified in **Tools** > **Options** > **Build & Run** > **Kits** (on Windows and Linux) or in **Qt Creator** > **Preferences Build & Run** > **Kits** (on macOS).
7. Select **Next**.
8. Review the project settings, and click **Finish** (or **Done** on macOS).

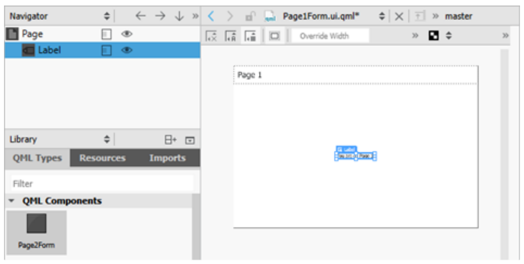
Qt Creator generates two UI files, *Page1Form.ui.qml* and *Page2Form.ui.qml*, and a QML file, *main.qml*. You can modify *Page1Form.ui.qml* in the **Form Editor** to create the application main view and *main.qml* in the **Text Editor** to add the application logic. For the purposes of this example, you can ignore *Page2Form.ui.qml*.

## Creating the Main View

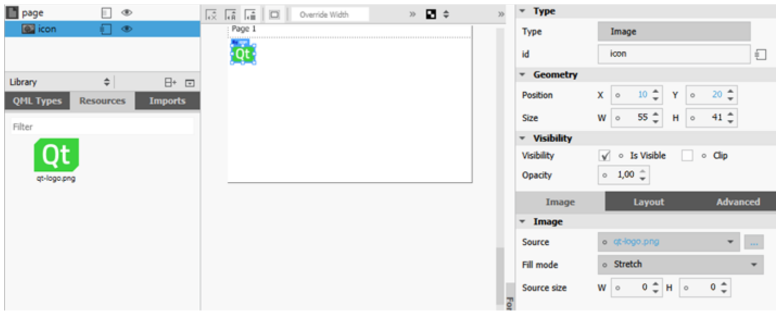
The main view of the application displays a Qt logo in the top left corner of the screen and two empty rectangles.

To use the *qt-logo.png* image in your application, you must copy it from the Qt examples directory to the project directory (same subdirectory as the QML file). The image appears in **Resources**. You can also use any other image or a QML type, instead.

1. In the **Projects** view, double-click the *Page1Form.ui.qml* file to open it in Qt Quick Designer.



2. In the **Navigator**, select **Label** and press **Delete** to delete it.
3. Select **Page** in the navigator, and enter *page* in the **Id** field.
4. In **Library** > **Resources**, select qt-logo.png and drag and drop it to the *page* in the navigator.

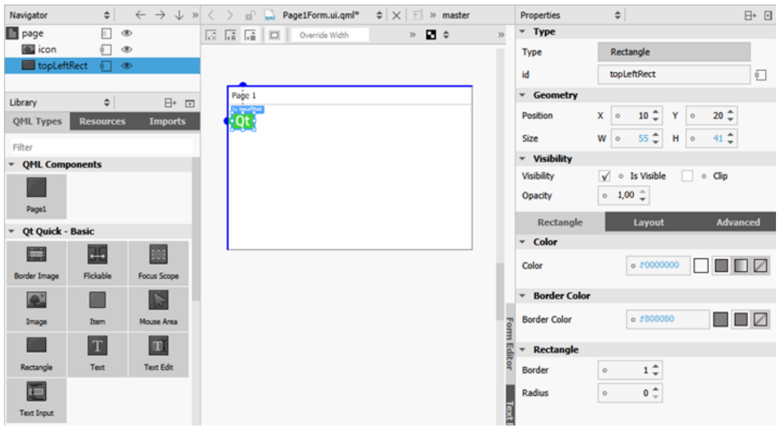


Type: Image
id: icon
Geometry
Position X 10 Y 20
Size W 55 H 41
Visibility: ✓ Is Visible, Clip
Opacity 1,00
Image
Source: qt-logo.png
Fill mode: Stretch
Source size W 0 H 0

a. In the **Id** field, enter *icon*.

b. In the **Position** field, set **X** to *10* and **Y** to *20*.

5. Right-click the resource file, qml.qrc, in the **Projects** view, and select **Add Existing File** to add qt-logo.png to the resource file for deployment.
6. Drag and drop a **Rectangle** to *page* in the navigator and edit its properties.



Type: Rectangle
id: topLeftRect
Geometry
Position X 10 Y 20
Size W 55 H 41
Visibility: ✓ Is Visible, Clip
Opacity 1,00
Color
Color #000000
Border Color
Border Color #808080
Rectangle
Border 1
Radius 0

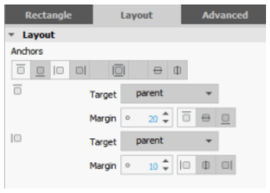
a. In the **Id** field, enter *topLeftRect*.

b. In the **Size** field, set **W** to *55* and **H** to *41*, for the rectangle size to match the image size.

c. In the **Color** field, click the (**Transparent**) button to make the rectangle transparent.

d. In the **Border color** field, set the border color to *#808080*.

e. Click **Layout**, and then click the top and left anchor buttons to anchor the rectangle to the top left corner of the page.



Target: parent
Margin: 20
Target: parent
Margin: 10

f. In the **Margin** field, select *20* for the top anchor and *10* for the left anchor.

7. Drag and drop a **Mouse Area** type from the **Library** to *topLeftRect* in the navigator.
8. Click **Layout**, and then click the (**Fill to Parent**) button to anchor the mouse area to the rectangle.
9. In the **Navigator**, copy topLeftRect (by pressing **Ctrl+C**) and paste it to the *page* in the navigator twice (by pressing **Ctrl+V**). Qt Creator renames the new instances of the type topLeftRect1 and topLeftRect2.
10. Select topLeftRect1 and edit its properties:

    a. In the **Id** field, enter *middleRightRect*.

    b. In **Layout**, select the vertical center anchor button and then the right anchor button to anchor the rectangle to the middle right margin of the screen.

    c. In the **Margin** field, select *10* for the right anchor and *0* for the vertical center anchor.

11. Select topLeftRect2 and edit its properties:

    a. In the **Id** field, enter *bottomLeftRect*.

    b. In **Layout**, select the bottom and left anchor buttons to anchor the rectangle to the bottom left margin of the screen.

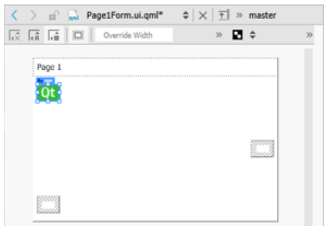
    c. In the **Margin** field, select *20* for the bottom anchor and *10* for the left anchor.

12. In the **Navigator**, select the (**Export**) button for each type to export all types as properties. This enables you to use the properties in the *main.qml* file.
13. Press **Ctrl+S** to save the changes.

To check your code, you can view *Page1Form.ui.qml* in the **Text Editor** and compare it with the Page1Form.ui.qml example file.

The new project wizard adds boilerplate code to the *Page1.qml* file to create menu items and push buttons. Modify the boilerplate code by removing obsolete code. You removed the push buttons from the UI form, so you also need to remove the corresponding code from *Page1.qml* (or the application cannot be built).

The UI is now ready and you can switch to editing the *main.qml* file in the **Text Editor** to add animation to the application, as described in the following section.



## Adding Application Logic

Edit the *main.qml* file to add pointers to two additional states: *State1* and *State2*. You cannot use the **Form Editor** to add states for a Window QML type. Use the **Text Editor** to add the states inside a StateGroup QML type and refer to them by using the id of the state group.

1. Specify the window size and background color as properties of the ApplicationWindow type:

```
ApplicationWindow {
    visible: true
    width: 640
    height: 480
    title: qsTr("Tabs")
```

2. Specify an id for the Page1 type to be able to use the properties that you exported in *Page1Form.ui.qml*:

```
    SwipeView {
        id: swipeView
        anchors.fill: parent
        currentIndex: tabBar.currentIndex

        Page1Form {
            id: page
```

3. Add a pointer to the clicked expressions in **mouseArea**:

```
            mouseArea {
            onClicked: stateGroup.state = ' '
            }
```

   The expression sets the state to the base state and returns the image to its initial position.

4. Add a pointer to a clicked expression to **mouseArea1** to set the state to *State1*:

```
            mouseArea1 {
            onClicked: stateGroup.state = 'State1'
            }
```

5. Add a pointer to a clicked expression to **mouseArea2** to set the state to *State2*:

```
            mouseArea2 {
            onClicked: stateGroup.state = 'State2'
            }
        }
```

6. Bind the position of the Qt logo to the rectangle to make sure that the logo is displayed within the rectangle when the view is scaled on different sizes of screens. Set expressions for the x and y properties, as illustrated by the following code snippet:

```
    StateGroup {
        id: stateGroup
        states: [
            State {
                name: "State1"

                PropertyChanges {
                    target: page.icon
                    x: page.middleRightRect.x
                    y: page.middleRightRect.y
                }
            },
            State {
                name: "State2"

                PropertyChanges {
                    target: page.icon
                    x: page.bottomLeftRect.x
                    y: page.bottomLeftRect.y
                }
            }
        ]
```

7. Press **Ctrl+R** to run the application.

Click the rectangles to move the Qt logo from one rectangle to another.

## Adding Animation to the View

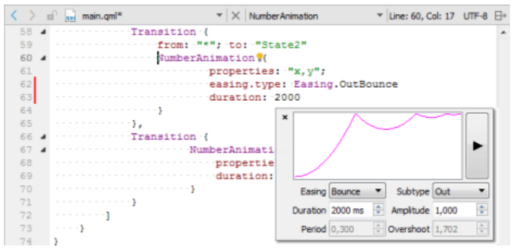
Add transitions inside the state group to define how the properties change when the Qt logo moves between states. The transitions apply animations to the Qt logo. For example, the Qt logo bounces back when it moves to the middleRightRect and eases into bottomLeftRect.

1. In the **Text Editor**, add the following code to specify that when moving to State1, the x and y coordinates of the Qt logo change linearly over a duration of 1 second:

```
        transitions: [
            Transition {
                from: "*"; to: "State1"
                NumberAnimation {
                    easing.type: Easing.OutBounce
                    properties: "x,y";
                    duration: 1000
                }
            },
```

2. You can use the Qt Quick toolbar for animation to change the easing curve type from linear to OutBounce:

   a. Click **NumberAnimation** in the **Text Editor** to display the 💡 icon, and then click the icon to open the toolbar:



main.qml* | NumberAnimation | Line: 60, Col: 17 UTF-8

```
            Transition {
                from: "*"; to: "State2"
                NumberAnimation {
                        properties: "x,y";
                        easing.type: Easing.OutBounce
                        duration: 2000
                }
            },
            Transition {
                    NumberAnimati
                        propertie
                        duration:
                    }
            }
        ]
    }
}
```

Easing: Bounce | Subtype: Out
Duration: 2000 ms | Amplitude: 1,000
Period: 0,300 | Overshoot: 1,702

   b. In the **Easing** field, select **Bounce**.

   c. In the **Subtype** field, select **Out**.

3. Add the following code to specify that when moving to State2, the x and y coordinates of the Qt logo change over a duration of 2 seconds, and an InOutQuad easing function is used:

```
    ...
            Transition {
                from: "*"; to: "State2"
                NumberAnimation {
                        properties: "x,y";
                        easing.type: Easing.InOutQuad;
                        duration: 2000
                }
            },
```

4. Add the following code to specify that for any other state changes, the x and y coordinates of the Qt logo change linearly over a duration of 200 milliseconds:

```
    ...
            Transition {
                     NumberAnimation {
                         properties: "x,y";
                         duration: 200
                     }
            }
        ]
```

5. Press **Ctrl+R** to run the application.

Click the rectangles to view the animated transitions.

Files:

- transitions/Page1Form.ui.qml
- transitions/Page2Form.ui.qml
- transitions/main.qml
- transitions/qml.qrc
- transitions/transitions.pro

Images:

- transitions/qt-logo.png

< Tutorials                Creating a Qt Widget Based Application >

| Download | Product | Services | Developers | About us |
|---|---|---|---|---|
| Start for Free | Qt in Use | Technology Evaluation | Documentation | Training & Events |
| Qt for Application Development | Qt for Application Development | Proof of Concept | Examples & Tutorials | Resource Center |
| Qt for Device Creation | Qt for Device Creation | Design & Implementation | Development Tools | News |
| Qt Open Source | Commercial Features | Productization | Wiki | Careers |
| Terms & Conditions | Qt Creator IDE | Qt Training | Forums | Locations |
| Licensing FAQ | Qt Quick | Partner Network | Contribute to Qt | Contact Us |

Qt The Qt Company

Sign In   Feedback   © 2018 The Qt Company