

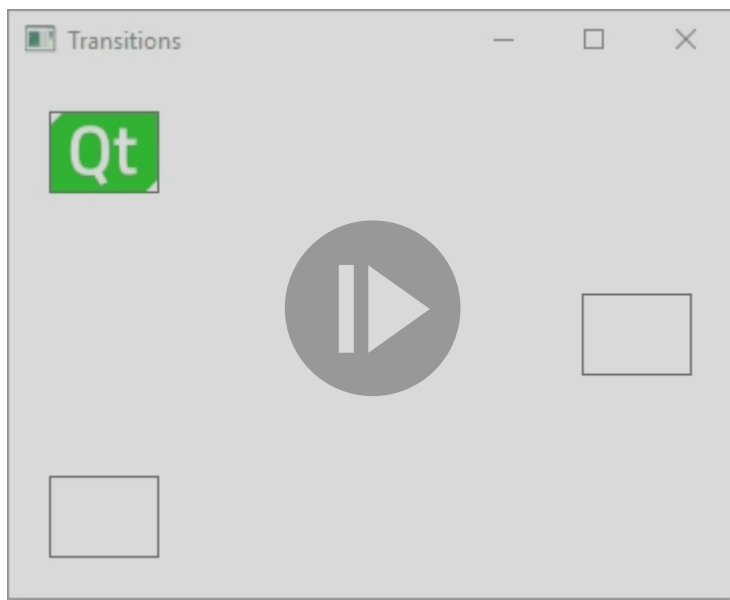
Qt 创建者手册 > [创建Qt快速应用程序](#)

创建Qt快速应用程序

本教程阐释了 [Qt 快速](#) 的基本概念。有关您拥有的 UI 选项的详细信息，请参阅[用户界面](#)。

本教程介绍如何在使用 Qt 6 作为最低 Qt 版本和 CMake 作为构建系统时使用 Qt 创建器实现[状态](#)和[转换](#)。

我们使用[编辑模式](#)创建一个应用程序，当您单击三个矩形时，该应用程序会在三个矩形之间移动Qt徽标。



有关更多示例，请参阅 [Qt 快速示例和教程](#)。

您也可以在Qt设计工作室中开发Qt快速应用程序。有关更多信息，请参阅[Qt设计工作室手册](#)。

创建项目

1. 选择“**文件**>**新项目**>**应用程序 (Qt)** > **Qt 快速应用程序**”。
2. 选择“**选择**”以打开“**项目位置**”对话框。
3. 在“**名称**”字段中，输入应用程序的名称。在命名您自己的项目时，请记住，以后无法轻松重命名它们。
4. 在“**创建位置**”字段中，输入项目文件的路径。您可以稍后移动项目文件夹而不会出现问题。
5. 选择“**下一步**”（或在 macOS 上选择“**继续**”）以打开“**定义生成系统**”对话框。
6. 在“**生成系统**”字段中，选择“**CMake**”作为用于生成和运行项目的生成系统。

注意： 如果选择 [qmake](#)，则配置项目的说明将不适用。

7. 选择“**下一步**”以打开“**定义项目详细信息**”对话框。

10. 选择“**下一步**”以使用默认设置并打开“**工具包选择**”对话框。
11. 要为其构建应用程序的平台选择 Qt 6.2 或更高版本的**工具包**。要为移动设备构建应用程序，请选择也适用于安卓和 iOS 的工具包。

注意：如果在“**编辑**>**首选项**>**工具包**”（在 Windows 和 Linux 上）或“**Qt 创建者**>**首选项**>**工具包**”（在 macOS 上）中指定了工具包，则会列出**这些工具包**。有关详细信息，请参阅**添加工具包**。

12. 选择“**下一步**”以打开“**项目管理**”对话框。
13. 查看项目设置，然后选择“**完成**”（或在 macOS 上**选择“完成”**）以创建项目。

有关跳过的设置和其他可用向导模板的更多信息，请参阅[创建 Qt 快速应用程序](#)。

Qt 创建器生成一个组件文件 *main.qml*，并在**编辑**模式下打开它。

部署应用程序

应用程序的主视图在视图左上角的矩形内显示一个 Qt 徽标和两个空矩形。

我们在本教程中使用 *qt-logo.png* 图像，但您也可以改用任何其他图像或组件。

若要在运行应用程序时显示该图像，必须在向导为您创建的 *CMakeLists.txt* 文件中将其指定为资源：RESOURCES

```
qt_add_qml_module(apptransitions
    URI transitions
    VERSION 1.0
    QML_FILES main.qml Page.qml
    RESOURCES qt-logo.png
)
```

创建自定义 QML 类型

Because the **Window** QML type requires states to be added into child components, we use the wizard to create a custom QML type called *Page* that we will refer to from *main.qml*.

To create a custom QML type:

1. Select **File > New File > Qt > QML File (Qt Quick 2)**.
2. Select **Choose** to open the **Location** dialog.
3. In the **File name** field, enter a name for the custom QML type. In this example, we call the type *Page*.
4. Select **Next** to open the **Project Management** dialog.
5. Select **Finish** to create *Page.qml*.


Qt Creator opens *Page.qml* in the **Edit** mode. It contains a root item of the type **Item** that we replace with a **Rectangle** type. We give the type the ID *page*, anchor it to the parent item on all sides, and set its color to white:

```
import QtQuick

Rectangle {
```

Because we develop with Qt 6, where version numbers are not used with modules, we remove the version number from the import statement.

When you start typing the QML type name, Qt Creator suggests available types and properties to **complete the code**.

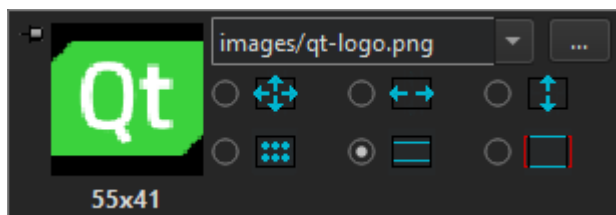
Select the light bulb icon  next to the type name to open the **Qt Quick Toolbar for rectangles**. You can use it to specify rectangle properties, such as color, transparency, and gradients.



Next, we add an **Image** type with *qt-logo.png* as the source. We position the image in the top-left corner of the rectangle:

```
Image {
    id: icon
    x: 20
    y: 20
    source: "qt-logo.png"
}
```

You can use the **Qt Quick Toolbar for images** to specify image properties, such as source file and fill mode.



We now create the rectangles that the image will move between. Their size should match the image size and they should be transparent, so that the image is visible. We set the border color to light gray to make the rectangles visible:

```
Rectangle {
    id: topLeftRect
    width: 55
    height: 41
    color: "#00ffffff"
    border.color: "#808080"
```

```
anchors.left: parent.left
anchors.top: parent.top
anchors.leftMargin: 20
anchors.topMargin: 20
```

We add a **MouseArea** type to make the rectangle clickable by users:

```
MouseArea {
    id: mouseArea
    anchors.fill: parent
```

To check your code, you can compare it with the *Page.qml* example file.

Next, we will make the image move between the rectangles when users click them, by adding states and by connecting mouse clicks to state changes.

Connecting Mouse Clicks to State Changes

To make the image move between the rectangles when users click them, we add states to the *Page* component, where we change the values of the and properties of *icon* to match those of the middle right and top left rectangles. To make sure that the image is displayed within the rectangle when the view is scaled on different sizes of screens, we *bind* the values of the and properties of *icon* to those of the rectangles:xyxy

```
...
states: [
    State {
        name: "State1"
    },
    State {
        name: "State2"

        PropertyChanges {
            target: icon
            x: middleRightRect.x
            y: middleRightRect.y
        }
    },
    State {
        name: "State3"

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

```

Connections {
    target: mouseArea
    function onClicked()
    {
        page.state = "State1"
    }
}

```

Because we develop with Qt 6, we must specify the connections as functions.

Adding Page to the Main View

We now open *main.qml* for editing and add an instance of the Page custom component to it:

```

import QtQuick

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Transitions")

    Page {
        id: page
        anchors.fill: parent
    }
}

```

Press **Ctrl+R** to run the application, and click the rectangles to move the Qt logo from one rectangle to another.

Animating Transitions

We will now create transitions to apply animation to the image. For example, the image bounces back when it moves to *middleRightRect* and eases into *bottomLeftRect*.

We specify transitions for switching from each state to the other two states:

```

transitions: [
    Transition {
        id: toState1
        ParallelAnimation {
            SequentialAnimation {
                PauseAnimation {
                    duration: 0
                }

                PropertyAnimation {
                    target: icon
                    property: "x"
                }
            }
        }
    }
]

```

```

    }

    SequentialAnimation {
        PauseAnimation {
            duration: 0
        }

        PropertyAnimation {
            target: icon
            property: "x"
            duration: 200
        }
    }
}
to: "State1"
from: "State2,State3"
},

```

We change the easing curve type for transitions to *State2* from linear to to create the bounce effect: `Easing.OutBounce`

```

Transition {
    id: toState2
    ParallelAnimation {
        SequentialAnimation {
            PauseAnimation {
                duration: 0
            }

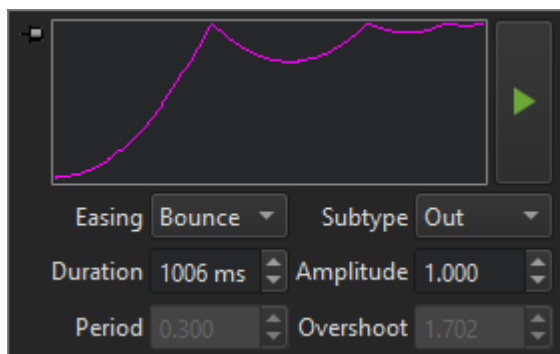
            PropertyAnimation {
                target: icon
                property: "y"
                easing.type: Easing.OutBounce
                duration: 1006
            }
        }

        SequentialAnimation {
            PauseAnimation {
                duration: 0
            }

            PropertyAnimation {
                target: icon
                property: "x"
                easing.type: Easing.OutBounce
                duration: 1006
            }
        }
    }
    to: "State2"
    from: "State1,State3"
}

```

Note: You can use the [Qt Quick Toolbar for animation](#) to specify the easing curve type and animation duration.



Then, we change the easing curve type for transitions to *State2* from linear to to create the easing effect: `Easing.InOutQuad`

```
Transition {
    id: toState3
    ParallelAnimation {
        SequentialAnimation {
            PauseAnimation {
                duration: 0
            }

            PropertyAnimation {
                target: icon
                property: "y"
                easing.type: Easing.InOutQuad
                duration: 2000
            }
        }

        SequentialAnimation {
            PauseAnimation {
                duration: 0
            }

            PropertyAnimation {
                target: icon
                property: "x"
                easing.type: Easing.InOutQuad
                duration: 2000
            }
        }
    }
    to: "State3"
    from: "State1,State2"
}
]
```

Files:

- transitions/CMakeLists.txt
- transitions/Page.qml
- transitions/main.qml

Images:

- transitions/qt-logo.png

< Tutorials

Creating a Qt Widget Based Application >

© 2022 The Qt Company Ltd. Documentation contributions included herein are the copyrights of their respective owners. The documentation provided herein is licensed under the terms of the [GNU Free Documentation License version 1.3](#) as published by the Free Software Foundation. Qt and respective logos are trademarks of The Qt Company Ltd in Finland and/or other countries worldwide. All other trademarks are property of their respective owners.



Contact Us

Company

- About Us
- Investors
- Newsroom
- Careers
- Office Locations

Licensing

- Terms & Conditions
- Open Source
- FAQ

Support

- Support Services
- Professional Services
- Partners
- Training

For Customers

- Support Center
- Downloads
- Qt Login
- Contact Us
- Customer Success

Community

- Contribute to Qt



Downloads

Marketplace

© 2022 The Qt Company

[Feedback](#) [Sign In](#)