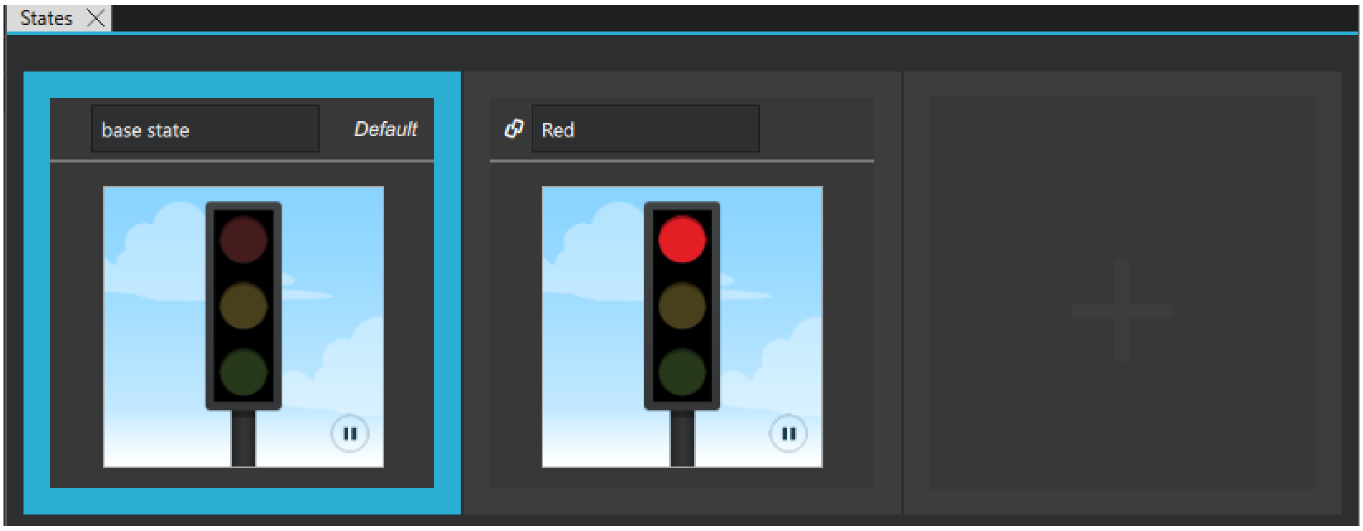
添加状态

可以通过选择 来定义"状态"视图中组件和组件实例的状态 + 。

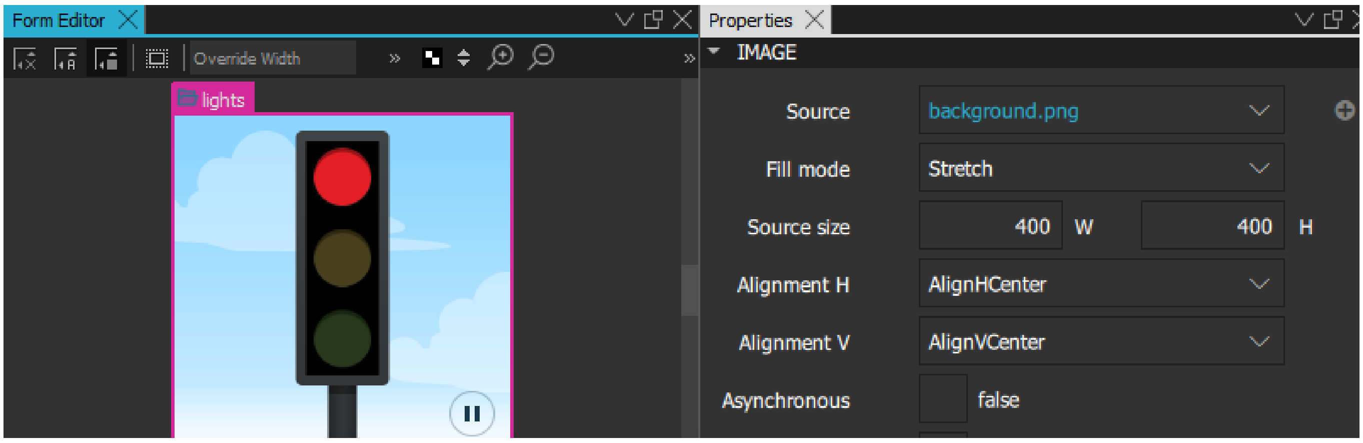


单击新状态以在 2D 视图中切换到该状态，然后在"属性"中修改组件或组件实例的属性值。

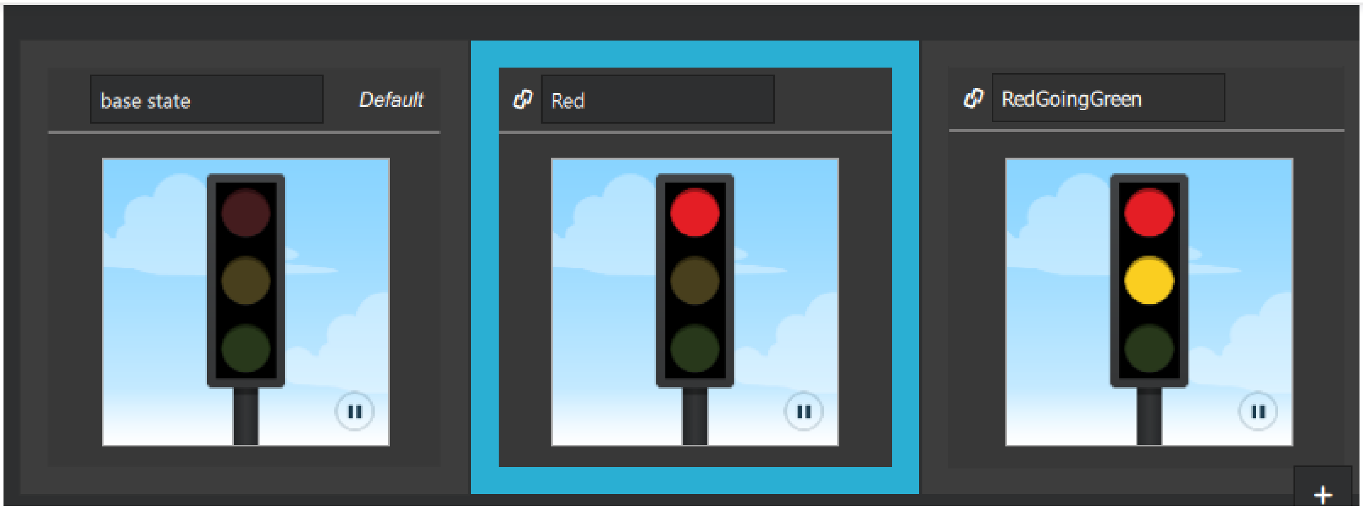
例如，若要更改按钮的外观，可以在按钮组件中定义状态以隐藏按钮图像并在其位置显示另一个图像，或者更改按钮背景或文本颜色。在其他组件中使用按钮实例时，可以通过隐藏或显示按钮组件实例来定义创建不同屏幕的状态。**组件** > Qt **快速**控件>控件中的预设按钮**控件**具有预定义的*正常*和*关闭*状态。

这也适用于可以使用向导模板创建的自定义按钮组件。有关编辑按钮组件中的状态以及隐藏和显示按钮以创建多个屏幕的详细信息，请参阅登录 UI - 组件和登录 UI - 状态。要向屏幕添加运动，可以在 2D 视图中更改组件实例的位置，然后将动画添加到状态之间的更改中。

在状态下更改的属性将以蓝色突出显示。在"代码"视图中，您可以看到记录为对基本状态的更改的更改。



Source	background.png
Fill mode	Stretch
Source size	400 W 400 H
Alignment H	AlignHCenter
Alignment V	AlignVCenter
Asynchronous	false



注意： 如果已在 Navigator 中锁定了某个组件，并且尝试删除更改其属性值的状态，系统将提示您确认删除。

有关详细信息，请观看以下视频：

Setting the Default State

To determine the startup state of the application, select ⚙ to open the **Actions** menu, and then select **Set as Default**.

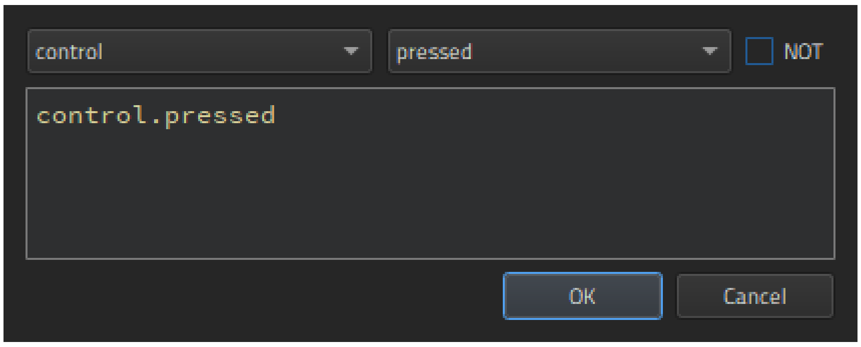
To reset the state later, select **Actions** > **Reset Default**.

Applying States

To determine when a state should be applied, select **Actions** > **Set when Condition**. In **Binding Editor**, specify a when property for the state. Set the value of the property to a boolean expression that evaluates to when you want the state to be applied.`true`

This enables you to evaluate the truthfulness of several components' properties and move the UI to the state in which these conditions apply. You can evaluate whether something is true or false, greater than or equal to something else, and so on. You can also use operators, such as AND or OR, to evaluate the truthfulness of several components.

In **Binding Editor**, select the component and property to create the expression. For example, to change the state when a button is pressed, you could select a button component and its pressed property.



```
control.pressed
```

When you compose the expressions in **Binding Editor**, the code completion feature lists the components and their properties you can use in the expressions.

Summary of Logical Operators

You can use the following logical operators in the expressions to combine several conditions in one expression:

Operator	Meaning	Evaluates to if `true`
!	NOT	The condition is not met.
&&	AND	Both conditions are met.
||	OR	Either of the conditions is met.
<	Less than	The left operand is less than the right operand.
>	Greater than	The left operand is greater than the right operand.
>=	Greater than or equal	The left operand is greater than or equal to the right operand.
<=	Less than or equal	The left operand is less than or equal to the right operand.
==	Equal	The operands are equal.
===	Strict equal	The operands are equal and of the same type.
!=	Not equal	The operands are not equal.
!==	Strict not equal	The operands are of the same type but not equal, or are of different type.

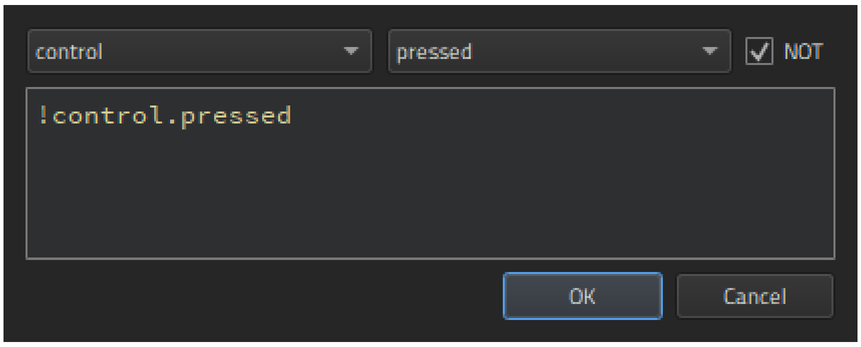
Alternatively, you can use **And Operator**, **Or Operator**, and **Not Operator** components to bind property values using the boolean AND, OR, and NOT operator. For more information, see Logic Helpers.

In addition, you can use arithmetic operators to compare numbers before checks. However, we recommend that you create separate properties for this purpose whenever possible.

Examples of when Conditions

To apply a state to a button when the button is pressed, you could simply write:

```
when: control.pressed
```


control
pressed
☑ NOT

```
!control.pressed
```

To apply a state when the button is not pressed, selected, nor hovered on, you could combine conditions, as follows:

```
when: !control.pressed && !control.checked && !control.hovered
```

To apply a state when the button is pressed or selected, but not hovered on, you could write:

```
when: control.pressed || control.checked && !control.hovered
```

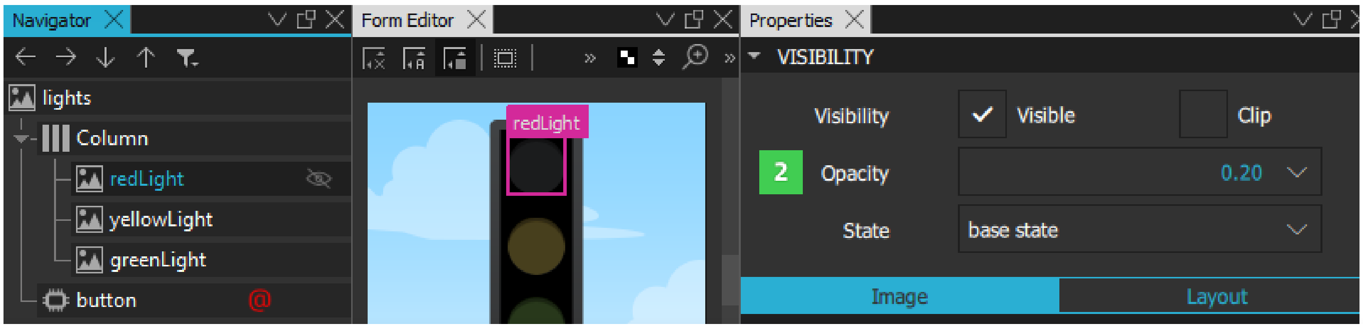
If you are not familiar with writing expressions, you can use preset logic helpers from **Components** > **Qt Quick Studio Logic Helper**.

Using States

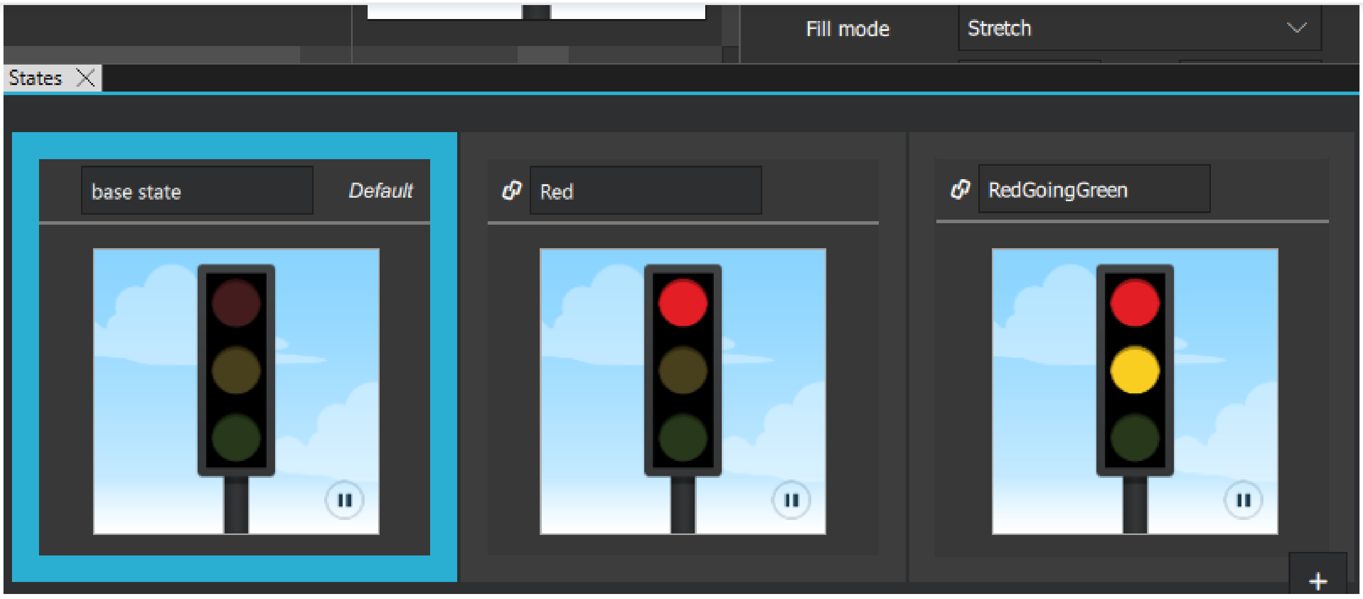
To keep the code clean, you should create a base state that contains all the components you will need in the application. You can then create states, in which you hide and show a set of components and modify their properties. This allows you to:

- Align components on different views with each other.
- Avoid excessive property changes. If a component is invisible in the base state, you must define all changes to its child components as property changes, which leads to complicated code.
- Minimize the differences between the base state and the other states to keep the code short and readable and to improve performance.
- Avoid problems when using transitions and animation when changing states.

To create views for an application by using states:



Visibility ✓ Visible Clip
Opacity 0.20
State base state



1. In the base state, add all components you will need in the application (1). While you work on one view, you can click the icon in Navigator to hide components on the canvas that are not part of a view.
2. In **States**, select the **+** symbol to create a new state and give it a name. For example, .`Normal`
3. In Properties (2), deselect the **Visibility** check box or set **Opacity** to 0 for each component that is not needed in this view. If you specify the setting for the parent component, all child components inherit it and are also hidden.
4. Create additional states for each view and set the visibility or opacity of the components in the view.
5. To determine which state is applied when the application starts, select **Actions** > **Set as Default**.

‹ Specifying Custom Properties   Validating with Target Hardware ›

Qt DOCUMENTATION

Support Services
Professional Services
Partners
Training

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

Feedback Sign In