Qt Quick for Qt Developers

States and Transitions

Based on Qt 5.4 (QtQuick 2.4)

Contents

The Qt Company

- States
- State Conditions
- Transitions

Objectives

The Qt Company

Can define user interface behavior using states and transitions:

- Provides a way to formally specify a user interface
- Useful way to organize application logic
- Helps to determine if all functionality is covered
- Can extend transitions with animations and visual effects

States and transitions are covered in the Qt documentation

States

States

The Qt Company

States manage named items

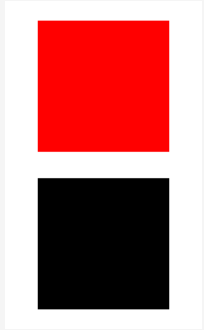
- Represented by the `State` element
- Each item can define a set of states
  - With the `states` property
  - Current state is set with the `state` property
- Properties are set when a state is entered
  - Can also modify anchors
  - Change the parents of items
  - Run scripts

See Documentation: QML States

States Example

The Qt Company

```
Rectangle {
    width: 150; height: 250
    Rectangle {
        id: stopLight
        x: 25; y: 15; width: 100; height: 100
    }
    Rectangle {
        id: goLight
        x: 25; y: 135; width: 100; height: 100
   }
}
```



- Prepare each item with an `id`
- Set up properties not modified by states

Defining States

The Qt Company

```
states: [
    State {
        name: "stop"
        PropertyChanges { target: stopLight; color: "red" }
        PropertyChanges { target: goLight; color: "black" }
    },
    State {
        name: "go"
        PropertyChanges { target: stopLight; color: "black" }
        PropertyChanges { target: goLight; color: "green" }
    }
]
```

- Define states with names: `"stop"` and `"go"`
- Set up properties for each state with `PropertyChanges`
  - Defining differences from the default values

Demo: qml-states-transitions/ex-states/states.qml

Setting the State

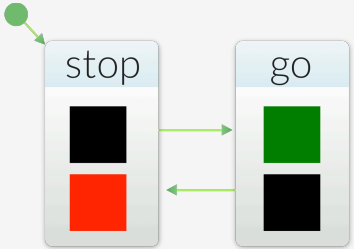
The Qt Company

- Define an initial state:

```
state: "stop"
```

- Use a `MouseArea` to switch between states:

```
MouseArea {
    anchors.fill: parent
    onClicked: parent.state == "stop" ?
               parent.state = "go" : parent.state = "stop"
}
```



- Reacts to a click on the user interface
  - Toggles the parent's `state` property between `"stop"` and `"go"` states

Changing Properties

The Qt Company

- States change properties with the `PropertyChanges` element:

```
State {
    name: "go"
    PropertyChanges { target: stopLight; color: "black" }
    PropertyChanges { target: goLight; color: "green" }
}
```

- Acts on a target element named using the target property
  - The target refers to an id
- Applies the other property definitions to the target element
  - One PropertyChanges element can redefine multiple properties
- Property definitions are evaluated when the state is entered
- PropertyChanges describes new property values for an item
  - New values are assigned to items when the state is entered
  - *Properties left unspecified are assigned their default values*

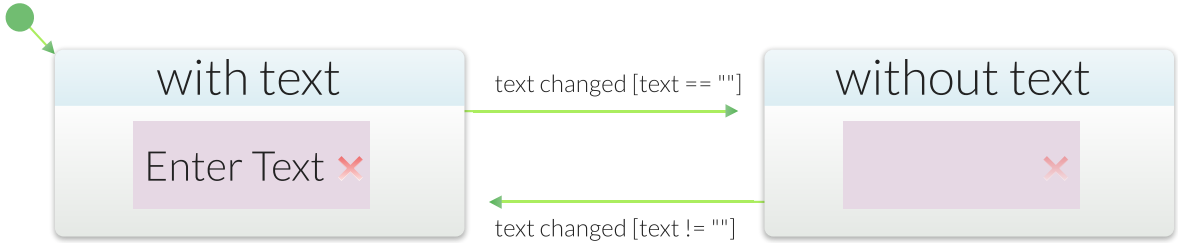
State Conditions

State Conditions

The Qt Company

Another way to use states:

- Let the `State` decide when to be active
  - Using conditions to determine if a state is active
- Define the `when` property
  - Using an expression that evaluates to `true` or `false`
- Only one state in a `states` list should be active
  - Ensure `when` is `true` for only one state



Demo: qml-states-transitions/ex-states/states-when.qml

State Conditions Example

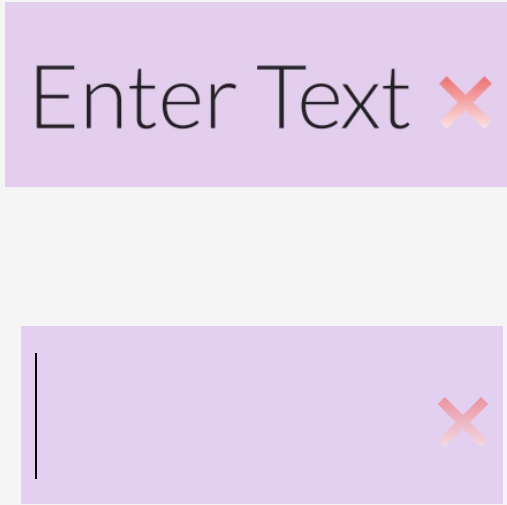
```
TextInput { id: textField
            text: "Enter text..."
            … }
Image { id: clearButton
        source: "../images/clear.svg”
        …
        MouseArea { anchors.fill: parent
                    onClicked: textField.text = "" }
}
```

- Define default property values and actions

State Conditions Example

The Qt Company

```
states: [
    State {
        name: "with text"
        when: textField.text != ""
        PropertyChanges {
            target: clearButton; opacity: 1.0
        }
    },
    State {
        name: "without text"
        when: textField.text == ""
        PropertyChanges {
            target: clearButton; opacity: 0.25 }
        PropertyChanges {
            target: textField; focus: true }
    }
]
```

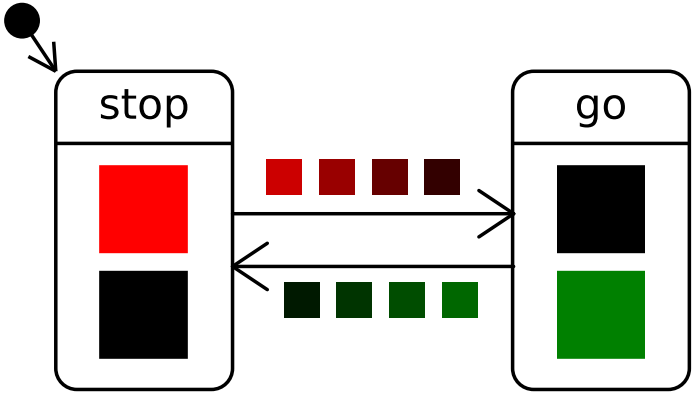


- A clear button that fades out when there is no text
- Do not need to define `state`

Transitions

Transitions

- Define how items change when switching states
- Applied to two or more states
- Usually describe how items are animated



- Let's add transitions to a previous example...

Demo: qml-states-transitions/ex-transitions/transitions.qml

Transitions Example

The Qt Company

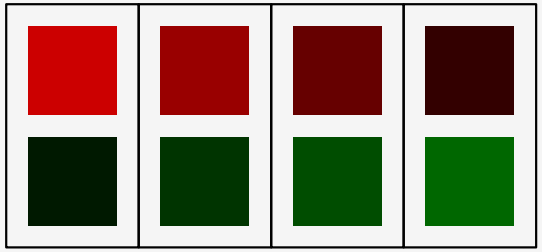
```
transitions: [
    Transition {
        from: "stop"; to: "go"
        PropertyAnimation {
            target: stopLight
            properties: "color"; duration: 1000
        }
    },
    Transition {
        from: "go";
        to: "stop"
        PropertyAnimation {
            target: goLight
            properties: "color"; duration: 1000
        }
    }]
```

- The `transitions` property defines a list of transitions
- Transitions between `"stop"` and `"go"` states

Wildcard Transitions

The Qt Company

```
transitions: [
    Transition {
        from: "*"; to: "*"
        PropertyAnimation {
            target: stopLight
            properties: "color"; duration: 1000 }
        PropertyAnimation {
            target: goLight
            properties: "color";
            duration: 1000 }
} ]
```



- Use "*" to represent any state
- Now the same transition is used whenever the state changes
- Both lights fade at the same time

Demo: qml-states-transitions/ex-transitions/transitions-multi.qml

Reversible Transitions

The Qt Company

```
transitions: [
    Transition {
        from: "with text"; to: "without text"
        reversible: true
        PropertyAnimation {
            target: clearButton
            properties: "opacity";
            duration: 1000
        }
} ]
```

- Useful when two transitions operate on the same properties
- Transition applies from "with text" to "without text"
  - And back again from "without text" to "with text"
- No need to define two separate transitions

Demo: qml-states-transitions/ex-transitions/transitions-reversible.qml

Parent Changes

```
states: State { name: "reanchored"
                ParentChange {
                    target: myRect
                    parent: yellowRect
                    x: 60; y: 20 }
              }
transitions: Transition { ParentAnimation {
                              NumberAnimation {
                                  properties: "x,y"
                                  duration: 1000 }
                          }
}
```

- Used to animate an element when its parent changes
- Element `ParentAnimation` applies only when changing the parent with `ParentChange` in a state change

Demo: qml-states-transitions/ex-animations/parent-animation.qml

Anchor Changes

The Qt Company

```
states: State { name: "reanchored"
                AnchorChanges {
                    target: myRect
                    anchors.left: parent.left
                    anchors.right : parent.right }
              }
transitions: Transition { AnchorAnimation {
                                duration : 1000 }
}
```

- Used to animate an element when its anchors change
- Element `AnchorAnimation` applies only when changing the anchors with `AnchorChanges` in a state change

Demo: qml-states-transitions/ex-animations/anchors-animation.qml

Using States and Transitions

The Qt Company

- Avoid defining complex state charts
  - Not just one state chart to manage the entire UI
  - Usually defined individually for each component
  - Link together components with internal states
- Setting state with script code
  - Easy to do, but might be difficult to manage
- Setting state with state conditions
  - More declarative style
  - Can be difficult to specify conditions
- Using animations in transitions
  - Do not specify `from` and `to` properties
  - Use `PropertyChanges` elements in state definitions

Summary – States

The Qt Company

`State` items manage properties of other items:

- Items define states using the `states` property
  - Must define a unique `name` for each state
- Useful to assign `id` properties to items
  - Use `PropertyChanges` to modify items
- The `state` property contains the current state
  - Set this using JavaScript code, or
  - Define a `when` condition for each state

Summary – Transitions

The Qt Company

`Transition` items describe how items change between states:

- Items define transitions using the `transitions` property

- Transitions refer to the states they are between
  - Using the `from` and `to` properties
  - Using a wildcard value, `"*"`, to mean any state

- Transitions can be reversible
  - Used when the `from` and `to` properties are reversed

Questions – States and Transitions

The Qt Company

- How do you define a set of states for an item?
- What defines the current state?
- Do you need to define a name for all states?
- Do state names need to be globally unique?
- Remember the thumbnail explorer page**?** Which states and transitions would you use for it?

Lab – Light Switch

- Using the partial solutions as hints, create a user interface similar to the one shown above.
- Adapt the reversible transition code from earlier and add it to the example.

Lab: qml-states-transitions/lab-switch