Qt Quick Composing UIs

Training Course

Visit us at `http://qt.digia.com`

Produced by Digia Plc.

Material based on Qt 5.0, created on September 27, 2012

Digia Plc.

digia

Module: States and Transitions

- States
- State Conditions
- Transitions

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
- State Conditions
- Transitions

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

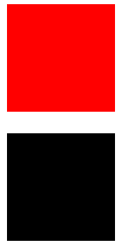
See QML States Documentation

States Example

```
import QtQuick 2.0

Rectangle {
  width: 150; height: 250

  Rectangle {
      id: stop_light
      x: 25; y: 15; width: 100; height: 100
  }
  Rectangle {
      id: go_light
      x: 25; y: 135; width: 100; height: 100
  }
  ...
```



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

Defining States

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

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

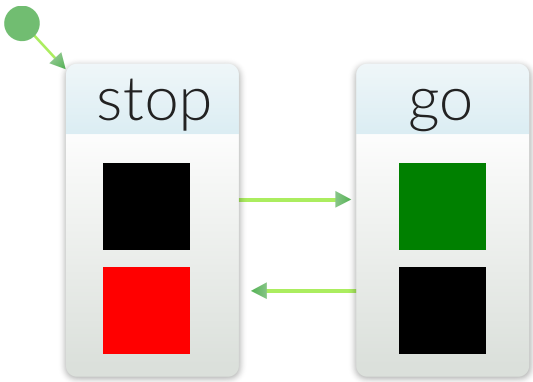
Demo qml-states-transitions/ex-states/states.qml

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

States change properties with the `PropertyChanges` element:

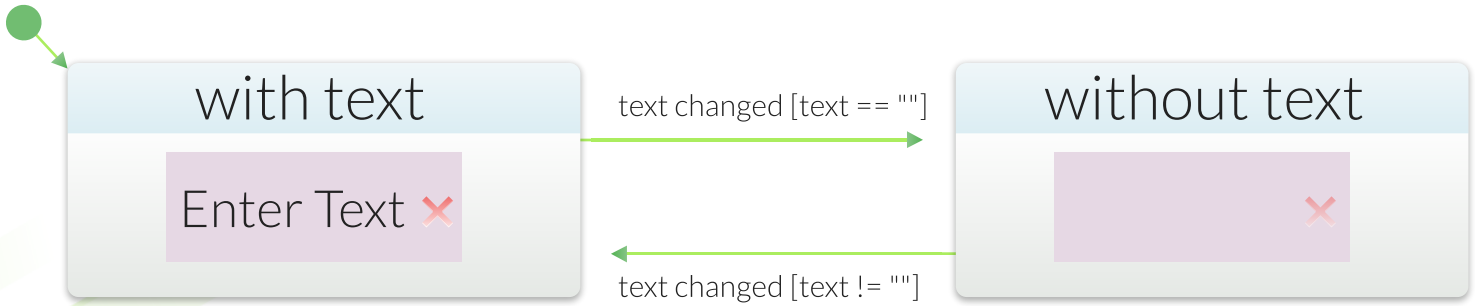
```
State {
  name: "go"
  PropertyChanges { target: stop_light; color: "black" }
  PropertyChanges { target: go_light; color: "green" }
}
```

- 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*

- States
- State Conditions
- Transitions

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

```
import QtQuick 2.0

Rectangle {
    width: 250; height: 50; color: "#ccffcc"

    TextInput { id: text_field
                text: "Enter text..." ... }

    Image {
        id: clear_button
        source: "../images/clear.svg"
        ...

        MouseArea { anchors.fill: parent
                onClicked: text_field.text = "" }
    }

    ...
```

- Define default property values and actions

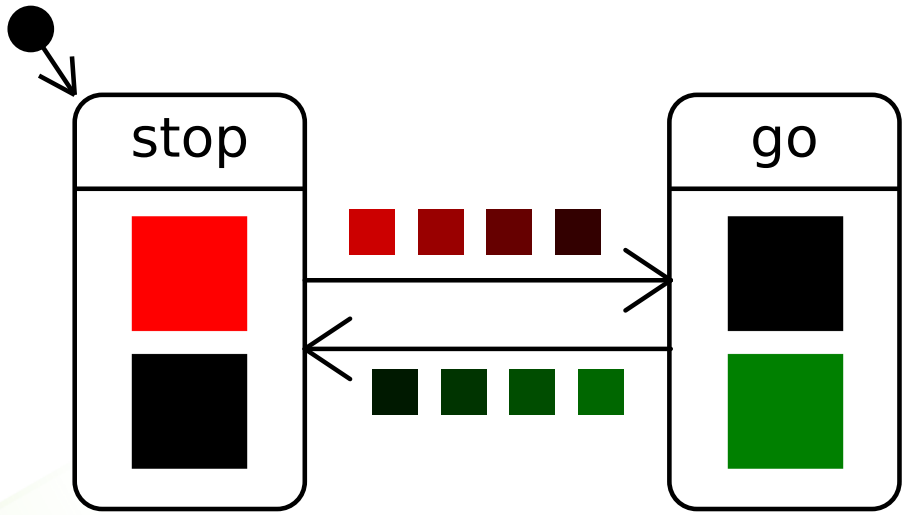
State Conditions Example

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

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

- States
- State Conditions
- 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

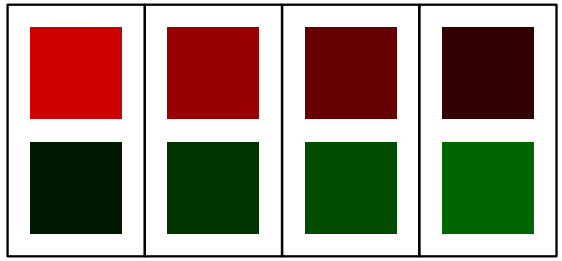
```
transitions: [
  Transition {
      from: "stop"; to: "go"
      PropertyAnimation {
          target: stop_light
          properties: "color"; duration: 1000
      }
  },
  Transition {
      from: "go"; to: "stop"
      PropertyAnimation {
          target: go_light
          properties: "color";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

  } ]
```

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

Wildcard Transitions

```
transitions: [
  Transition {
      from: "*"; to: "*"
      PropertyAnimation {
          target: stop_light
          properties: "color"; duration: 1000
      }
      PropertyAnimation {
          target: go_light
          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

- Useful when two transitions operate on the same properties

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

]
```

- 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

- Used to animate an element when its parent changes

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

- ParentAnimation applies only when changing the parent with ParentChange in a state change

Demo qml-animations/ex-animations/parent-animation.qml

Anchor Changes

- Used to animate an element when its anchors change

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

- AnchorAnimation applies only when changing the anchors with AnchorChanges in a state change

Demo qml-animations/ex-animations/anchors-animation.qml

Demo qml-animations/ex-animations/parent-anchors-animation.qml

- Avoid defining complex statecharts
  - not just one statechart 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

`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

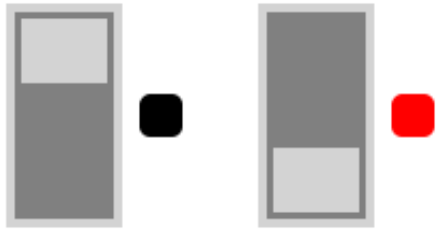
`Transition` items describe how items change between states:

- 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

Exercise – States and Transitions

- 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