Author: dailey.dai@openthinks.com

Qt Quick Event

Signal & Slot

QML has a signal and handler mechanism, where the *signal* is the event and the signal is responded to through a *signal handler*. When a signal is emitted, the corresponding signal handler is invoked.

Signal in QML

```
import QtQuick 2.2
import QtQuick.Controls 1.2

Rectangle{
    width:320;
    height:240;
    color:"gray";
    Button{
        text:"Quit";
        anchros.centerIn:parent;
        onClicked:{
            Qt.quit();
        }
    }
}
```

Signal handlers

`onClicked` is signal handler, and `clicked()` is a signal; usually looks like : on

```
onClicked:{
    //TODO
}
```

Property change signal handlers

A signal is automatically emitted when the value of a QML property changes.

```
import QtQuick 2.0

Text{
    text:"Some Text";
    onTextChanged:console.log(text);
}
```

```
<Property>
on<Property>Changed
```

Attached signal handlers

An attached signal handler is a signal handler that receives a signal from an *attaching type* rather than the object within which the handler is declared.

```
import QtQuick 2.0

Rectangle {
    width: 200; height: 200
    color: Qt.rgba(Qt.random(), Qt.random(), Qt.random(), 1)

    Component.onCompleted: {
        console.log("The rectangle's color is", color)
    }
}
```

Attached Properties and Attached Signal Handlers

```
<AttachingType>.<propertyName>
<AttachingType>.on<SignalName>
```

Connections

In some cases it may be desirable to access a signal outside of the object that emits it. For these purposes, the Qt Quick module provides the Connections type for connecting to signals of arbitrary objects. A Connections object can receive any signal from its specified target.

```
import QtQuick 2.0

Rectangle {
    id: rect
    width: 100; height: 100

    MouseArea {
        id: mouseArea
        anchors.fill: parent
    }

    Connections {
        target: mouseArea
        onClicked: {
            rect.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
        }
    }
}
```

```
Connections{
    target: area;
    on<Signal>: function or code block;
}
```

Custom signal

Signals can be added to custom QML types through the signal keyword.

The syntax for defining a new signal is:

`signal <name>[([<type> <parameter name>[, ...]])]`

```
 // SquareButton.qml
  Rectangle {
      id: root

      signal activated(real xPosition, real yPosition)

      property int side: 100
      width: side; height: side

      MouseArea {
          anchors.fill: parent
          onPressed: root.activated(mouse.x, mouse.y)
      }
  }
```

```
// myapplication.qml
  SquareButton {
      onActivated: console.log("Activated at " + xPosition + "," + yPosition)
  }
```

Signal connect to slot

Signal objects have a connect() method to a connect a signal either to a method or another signal.

```
Rectangle {
      id: relay
      signal messageReceived(string person, string notice)
      Component.onCompleted: {
          relay.messageReceived.connect(sendToPost)
          relay.messageReceived.connect(sendToTelegraph)
          relay.messageReceived.connect(sendToEmail)
          relay.messageReceived("Tom", "Happy Birthday")
      }
      function sendToPost(person, notice) {
          console.log("Sending to post: " + person + ", " + notice)
      }
      function sendToTelegraph(person, notice) {
```

```
            console.log("Sending to telegraph: " + person + ", " + notice)
        }
        function sendToEmail(person, notice) {
            console.log("Sending to email: " + person + ", " + notice)
        }
    }
```

There is a corresponding disconnect() method for removing connected signals:

```
Rectangle {
        id: relay
        //...

        function removeTelegraphSignal() {
            relay.messageReceived.disconnect(sendToTelegraph)
        }
    }
```

Signal to Signal Connect

By connecting signals to other signals, the connect() method can form different signal chains.

```
    Rectangle {
        id: forwarder
        width: 100; height: 100

        signal send()
        onSend: console.log("Send clicked")

        MouseArea {
            id: mousearea
            anchors.fill: parent
            onClicked: console.log("MouseArea clicked")
        }

        Component.onCompleted: {
            mousearea.clicked.connect(send)
        }
    }
```

```
  output:
        MouseArea clicked
        Send clicked
```