Author: dailey.dai@openthinks.com

Components and Dynamic Objects

Component

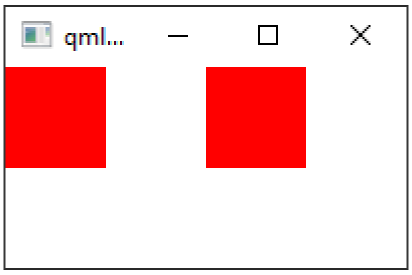
Components are reusable, encapsulated QML types with well-defined interfaces.

Component can be defined with two ways:

1. Embed QML document
2. Separate QML file

Embed Component

```
//component_sample.qml
import QtQuick 2.0
Item {
    width: 200; height: 100
    Component {
        id: redSquare
        Rectangle {
            color: "red"
            width: 50
            height: 50
        }
    }
    Loader { sourceComponent: redSquare }
    Loader { sourceComponent: redSquare; x: 100 }
}
```



Component only can include one top level `Item` , and only one property `id` , the component is not visible, it is just a new type, only can be see after instantiated.

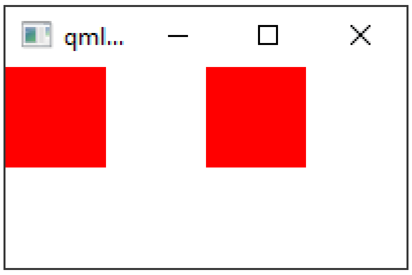
Separate QML in file

```
// RedSquare.qml
import QtQuick 2.0
Rectangle {
    id:redSquare;
    color: "red"
    width: 50
    height: 50
}
```

In `RedSquare.qml` we defined a component named `redSquare` , here two points need notice:

1. `Component` should not used in this definition.
2. the separate QML file name first character must be uppercase.

```
//component_sample.qml
import QtQuick 2.0
Item {
    width: 200; height: 100
    RedSquare{}
    RedSquare{
        x:100;
    }
}
```



It is easy to use the component which defined in separate QML file, but still two points need notice:

1. the new component type name must be same with the separate QML file name (without suffix)
2. the separate QML file should be placed in same folder with other referenced QML

Loader

Loader is used to dynamically load QML components.

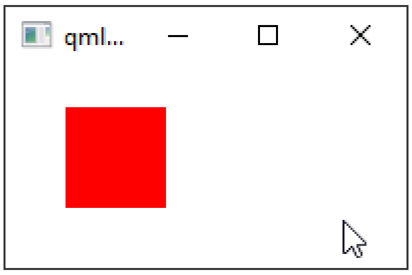
Loader can load a QML file (using the source property) or a Component object (using the sourceComponent property).

It is useful for delaying the creation of a component until it is required.

Load from component

```
//component_sample_loader.qml
import QtQuick 2.0
```

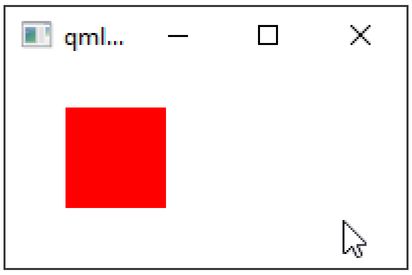
```
Item {
    width: 200; height: 100;
    Component {
        id: redSquare;
        Rectangle {
            color: "red";
            width: 50;
            height: 50;
        }
    }
    Loader {
        sourceComponent: redSquare ;
        onLoaded:{
            item.x= 20;
            item.y= 20;
        }
    }
}
```



Here, we can access loaded component by `Loader.item` , the property `item` for `Loader` is point to loaded component top level item.

Load from QML file

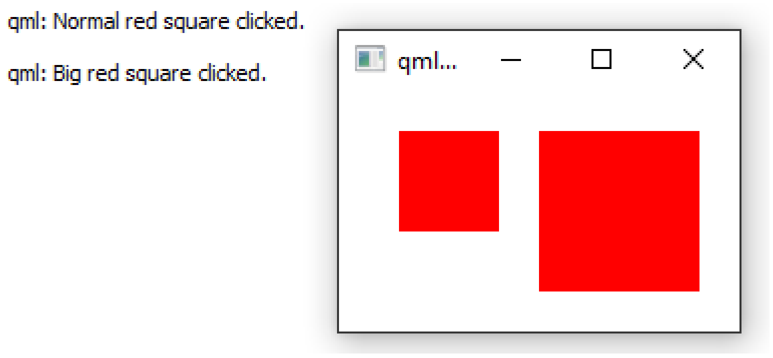
```
//component_sample_loader2.qml
import QtQuick 2.0
Item {
    width: 200; height: 100
    Loader{
        id:redloader;
        source: "RedSquare.qml";
        anchors.left: parent.left;
        anchors.leftMargin: 10;
        onLoaded: {
            item.x= 20;
            item.y= 20;
        }
    }
}
```



Event handler

```
// RedSquare.qml
import QtQuick 2.0
Rectangle {
    id:redSquare;
    color: "red";
    width: 50;
    height: 50;
    signal squarePressed();
    MouseArea{
        anchors.fill: parent;
        onClicked:squarePressed();
    }
}
```

```
//component_sample_loader3.qml
import QtQuick 2.0
Item {
    width: 200; height: 120;
    Loader{
        id:redloader;
        source: "RedSquare.qml";
        anchors.left: parent.left;
        anchors.leftMargin: 10;
        onLoaded: {
            item.x= 20;
            item.y= 20;
        }
    }
    Loader{
        id:redloaderBig;
        source: "RedSquare.qml";
        anchors.left: parent.left;
        anchors.leftMargin: 10;
        onLoaded: {
            item.x= 90;
            item.y= 20;
            item.width=80;
            item.height=80;
        }
    }
    Connections{
        target: redloader.item;
        onSquarePressed:{
            console.log("Normal red square clicked.");
        }
    }
    Connections{
        target: redloaderBig.item;
        onSquarePressed:{
            console.log("Big red square clicked.");
        }
    }
}
```

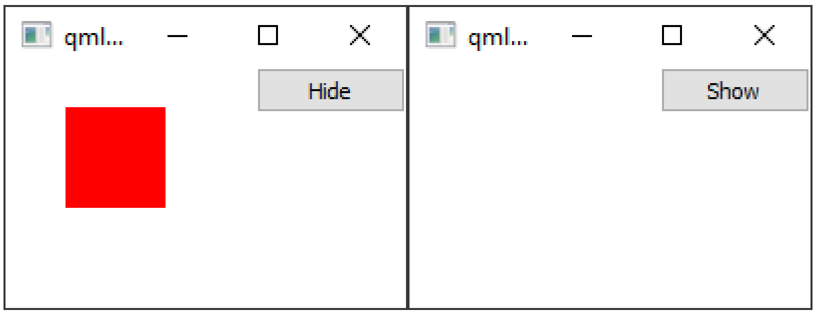


```
qml: Normal red square clicked.
qml: Big red square clicked.
```

Dynamic create & destroy component

```
import QtQuick 2.0
import QtQuick.Controls 1.2
Item {
    width: 200; height: 120;
    property bool isShow: true;
    Loader{
        id:redloader;
        source: "RedSquare.qml";
        anchors.left: parent.left;
        anchors.leftMargin: 10;
        onLoaded: {
            item.x= 20;
            item.y= 20;
        }
    }

    Button{
        id:ctrlButton;
        text:"Hide";
        anchors.right: parent.right;
        onClicked: {
            if(isShow){
                ctrlButton.text="Show";
                redloader.source="";
                //redloader.sourceComponent=undefined;
                isShow=false;
            }else{
                ctrlButton.text="Hide";
                redloader.source="RedSquare.qml";
                isShow=true;
            }
        }
    }
}
```

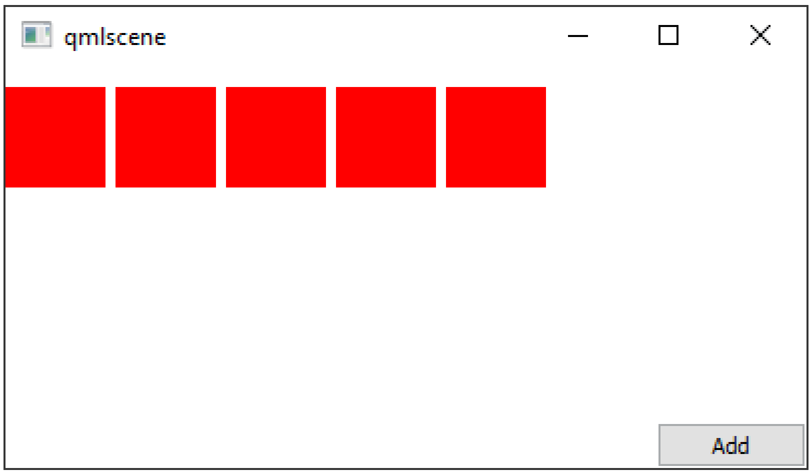


Dynamic object creation in ECMAScript

QML supports the dynamic creation of objects from within JavaScript. This is useful to delay instantiation of objects until necessary, thereby improving application startup time. It also allows visual objects to be dynamically created and added to the scene in reaction to user input or other events.

Qt.createComponent()

```
import QtQuick 2.0
import QtQuick.Controls 1.2
Item {
    id:rootItem;
    width: 400; height: 200;
    property var count: 0;
    property Component component: null;
    function createRedSquare(){
        if(component==null){
            component=Qt.createComponent("RedSquare.qml");
        }
        var redSquare;
        if(component.status==Component.Ready){
            redSquare = component.createObject(rootItem,{"x":count*55,"y":10})
        }
        count++;
    }
    Button{
        id:addButton;
        text:"Add";
        anchors.right: parent.right;
        anchors.bottom: parent.bottom;
        onClicked: {
            createRedSquare();
        }
    }
}
```



```
//url: QML file address or network addrss
//mode: Component.PreferSynchronous, Component.Asynchronous; default value is synchronous
//parent: parent component or object
object createComponent(url,mode,parent)
//parent: parent item
//options: key-value object which initialize object
object createObject(parent,options)
```

For embed component, just use `createObject()` directly .

Qt.createQmlObject()

If the QML is not defined until runtime, you can create a QML object from a string of QML using the `Qt.createQmlObject()` function, as in the following example:

```
var newObject = Qt.createQmlObject('import QtQuick 2.0; Rectangle {color: "red"; width: 20;
height: 20}',parentItem,"dynamicSnippet1");
```

The first argument is the string of QML to create. Just like in a new file, you will need to import any types you wish to use.

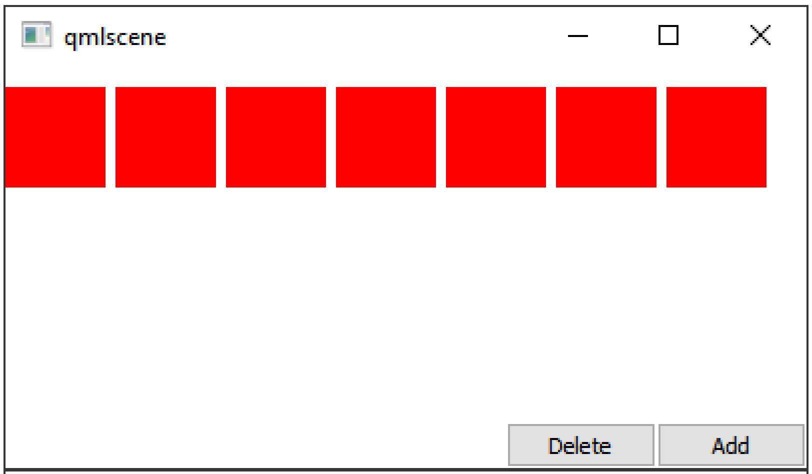
The second argument is the parent object for the new object, and the parent argument semantics which apply to components are similarly applicable for `createQmlObject()`.

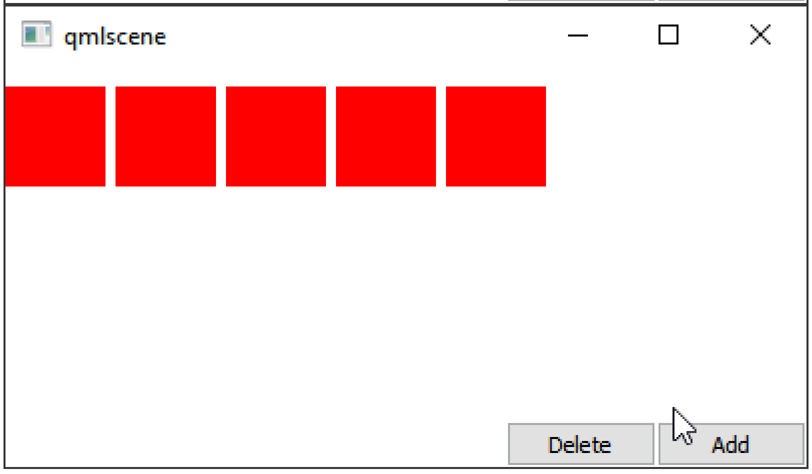
The third argument is the file path to associate with the new object; this is used for error reporting.

Destroy object by dynamic creation

1. The object created by `Loader`, should destroy them by set `source=""` or `sourceComponent=undefined`
2. The object created by `Qt.createComponent()` or `Qt.createQmlObject()` should destroy them by method `destroy()`

```
import QtQuick 2.2
import QtQuick.Controls 1.2
Item {
    id:rootItem;
    width: 400; height: 200;
    property var count: 0;
    property Component component: null;
    property var dynamicObjects: []; //new Array();
    function createRedSquare(){
        if(component==null){
            component=Qt.createComponent("RedSquare.qml");
        }
        var redSquare;
        if(component.status==Component.Ready){
            redSquare = component.createObject(rootItem,{"x":count*55,"y":10})
            dynamicObjects.push(redSquare);
        }
        count++;
    }
    function deleteRedSquare(){
        var redSquare = dynamicObjects.pop();
        if(redSquare!==undefined){
            redSquare.destroy();
            count--;
        }
    }
    Button{
        id:addButton;
        text:"Add";
        anchors.right: parent.right;
        anchors.bottom: parent.bottom;
        onClicked: {
            createRedSquare();
        }
    }
    Button{
        id:delButton;
        text:"Delete";
        anchors.right: addButton.left;
        anchors.bottom: parent.bottom;
        onClicked: {
            deleteRedSquare();
        }
    }
}
```


qmlscene
Delete
Add


qmlscene
Delete
Add