

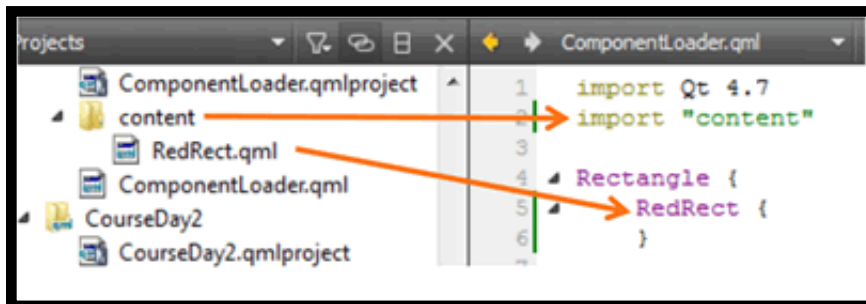
Gestión dinámica de componentes QML - I

Antes de empezar a interactuar con los distintos componentes de forma dinámica tenemos que aprender algunas cuestiones más:

import para todo

La clausula **import** se puede utilizar para referenciar a **ficheros QML** en otros directorios

- Importación de fichero simple
- o de directorio



- Un directorio importado puede ser referenciado mediante un alias (**qualifier**)

```
import Qt 4.7
import "content" as Content
Rectangle {
    Content.RedRect {
    }
```

La clausula **import** también se puede utilizar para importar **ficheros JavaScript**

- No se pueden importar directorios
- Tienes que ponerles obligatoriamente un alias (**qualifier**)

```
import "js/startup.js" as Startup
Rectangle {
    Component.onCompleted: Startup.loadItems(rootRect);
}
```

Ámbitos de las propiedades

- Las propiedades (**properties**) de los componentes son visibles en los hijos. Que lo sean

no quiere decir que sea una buena práctica

```

Main.qml
Rectangle {
    width: 200
    height: 200
    property string inheritedText: "x"
    RedRect { }
}

RedRect.qml
Rectangle {
    width: 25
    height: 25
    x: 25; y: 25
    color: "red"
    Text {
        anchors.fill: parent
        verticalAlignment: Text.AlignVCenter
        horizontalAlignment: Text.AlignHCenter
        text: inheritedText
    }
}
    
```

- Mejor es que en cada componente tenga un API propio (propiedades específicas).

```

Rectangle {
    width: 200
    height: 200
    property string inheritedText: "x"
    RedRect {
        text: inheritedText
    }
}

Rectangle {
    property alias text: text.text
    width: 25
    height: 25
    x: 25; y: 25
    color: "red"
    Text {
        id: text
        anchors.fill: parent
        verticalAlignment: Text.AlignVCenter
        horizontalAlignment: Text.AlignHCenter
        text: ""
    }
}
    
```

Ámbito de los Scripts

- Es indiferente que los scripts estén contenidos en el mismo fichero [try this site.qml](#) o sean ficheros externos .js

```

import Qt 4.7
import "script.js" as StartupScript

Rectangle {
    width: 200
    height: 200
    property string inheritedText: "x"
    RedRect { }
    Component.onCompleted: StartupScript.run();
}

function run()
{
    inheritedText = "xx";
}
    
```

- Siempre que se pueda separa el UI (.qml) y el comportamiento del mismo (.js).

Componentes declarados en línea (Inline components)

- Los componentes pueden declararse en línea

```
Component {
  id: helloComponent
  Text { text: "Loaded from: " + helloComponent.url }
}
```

- Se suele usar para declarar componentes pequeños o privados
- Se pueden instanciar mediante un cargador (**Loader**). Esto nos permite cargarlos e instanciarlos desde la web.

```
import QtQuick 1.0

Item {
  width: 200; height: 200

  Loader { id: pageLoader }

  MouseArea {
    anchors.fill: parent
    onClicked: pageLoader.source = "Page1.qml"
  }
}
```

Carga dinámica (Dynamic loading)

Además de los cargadores (**Loader**), los componentes se pueden leer dinámicamente mediante código script.

- **Qt.createComponent** (Parámetro: File o URL)

```
import QtQuick 1.0

Item {
  id: container
  width: 300; height: 300

  function loadButton() {
    var component = Qt.createComponent("Button.qml");
    if (component.status == Component.Ready) {
      var button = component.createObject(container);
      button.color = "red";
    }
  }

  Component.onCompleted: loadButton()
}
```

- **component.createObject** crea una instancia del componente (Parámetro: objeto padre)

```
component.createObject(parent, {"x": 100, "y": 100, "specialProperty": someObject});
```

- **Qt.createQmlObject** permite crear objetos QML desde una cadena

```
var newObject = Qt.createQmlObject('import QtQuick 1.0; Rectangle {color: "red"; width: 20; height: 20}',  
parentItem, "dynamicSnippet1");
```