How to write a generic Flex / AIR UI that can connect to different services

October 5, 2009
John Mattos

Summary

This document provides the reader with a quick step by step guide to setting up a generic Flex application to consume an RPC service from the client.

Introduction

This document is intended to give you a quick start guide to create a Flex project that will consume the Remote Object application created. If you haven't done so already, run through the quickstart guide on setting up a simple LCDS service before doing this tutorial. For the purposes of brevity, best practices in terms of the architecture of the application will not be considered here. Furthermore, we will not introduce the concept of dynamic endpoints. This slightly more advanced topic is covered off in a separate development guide.

Defining the RemoteObject in Flex

Define a `<mx:RemoteObject>` that will consume the created Java service. With the destination defined in the `remoting-config.xml` file, we can call the Java class's methods from our Flex application using the `<mx:RemoteObject>` tag. The table below explains the properties:

Property	Description
id	Instance name of the object.
destination	Service ID destintation, matching the one defined in remoting-config.xml
result	The event handler for the result
fault	The event handler for the fault

```
<mx:RemoteObject
       id="roService" destination="roDestination"/>
```

Invoking the service

To invoke the service simply call the Java class' public methods in either system events or user events

Example 1 – User Event – the click of a button

```
<mx:Button label="get Data"
       click="roService.getData()" />
```

Example 2 – System Event – creationComplete of the application

```
<mx:Application
```

```
        xmlns:mx=http://www.adobe.com/2006/mxml
        layout="absolute"
        creationComplete="roService.getData();">
```

Accessing returned data in binding expressions

To directly access data returned in a binding expression, use the `lastResult` property of the method call object. In the following example, the result is bound to a `DataGrid` control.

```
<mx:DataGrid dataProvider="{roService.getData.lastResult}"/>
```

Handling results from the RemoteObject

The ResultObject's result event occurs on successful completion of the remote call. Object and event listeners must be declared to handle the event. Note that in these examples, `event` is an instance of `mx.rpc.events.ResultEvent`. This Object's `result` property refers to the returned data and must be cast to the right datatype since the result object can be of any type.

```
<mx:RemoteObject
id="roService" destination="roDestination"
            result="myResultHandler(event)"/>
```

MXML example showing definition of result handler

```
import mx.rpc.events.ResultEvent;
import mx.rpc.remoting.RemoteObject;

private var myRemoteObject:RemoteObject = new RemoteObject("roService");

private function init():void{
       myRemoteObject.addEventListener(ResultEvent.RESULT, resultHandler);
}
```

ActionScript example

A typical result handler is shown below:

```
[Bindable]
private var data:ArrayCollection;
private function resultHandler(event:ResultEvent):void {
       data=event.result as ArrayCollection;
}
```

Note that in this example we are casting the result to ArrayCollection. This would be the cast when the Java return type is `List`

Handling faults from the RemoteObject

A fault event occurs when an error is triggered (on the client or server). `fault` is an instance of `mx.rpc.events.FaultEvent`, which has the following properties:

Property	Description
faultCode	Simple code describing the fault
faultDetail	Extra details (if any) on the fault
faultString	The text description of the fault
rootCause	The cause of the fault

As with the result event, handlers must be created to handle the fault:

```
<mx:RemoteObject
id="roService" destination="roDestination"
           fault="myfaultHandler(event)"/>
```

MXML example showing declaration of fault handler

```
private function init():void{
      myRemoteObject.addEventListener(ResultEvent.FAULT, faultHandler);
}
```

ActionScript example

A typical fault handler is shown below:

```
[Bindable]
import mx.rpc.events.FaultEvent;
import mx.controls.Alert;

private function faultHandler(event:FaultEvent):void {
      Alert.show (event.fault.faultString,event.fault.faultCode);
}
```

Debugging techniques

Flex builder debugger

Flex builder has a built in debugger which may be used to examine the resulting data from a remote call. To activate the debugger to examine the result object, place a breakpoint on the closing brace of the result event handler, and debug the application.

Navigate in the variables view to **event>result...** to see the returned data

Charles

Charles is an HTTP proxy / HTTP monitor that enables a developer to view HTTP traffic between their machine and the internet. If you have Charles version 2.4 or higher you can inspect AMF3 requests and responses between Flex and Java `RemoteObjects`. The messages are displayed in a tree format representing the structure of the message. Using Charles is invaluable to the developer working with LCDS as it enables them to have low-level inspection of the messages passing back and forth between client and server.