Externalized LiveCycle/Blaze Data Services 2.6 Configuration

October 7, 2009
Eric Garza

Summary

This document provides the reader with a discussion on LiveCycle/Blaze Data Services configuration from an external file or http service. These techniques along with dynamic service configuration will allow service location information to be loaded at run-time avoiding recompilation of the application when service locations change.

Introduction

Normally the **services-config.xml** file is provided to the Flex compiler during compilation. If the services ip and port do not change, this is fine. What most people forget, is that this file is used at compile time, not run-time.

Using the data services APIs that allow you to dynamically configure LiveCycle/Blaze services at run-time and an external configuration file, it is possible to avoid specifying a **services-config.xml** file in the compiler when building your application and to change services information without recompiling your application.

Chapter 4 discussed how to specify service information from MXML or ActionScript. This has limitations, as you still have the services information hard coded, possibly requiring you to recompile the application when service information changes. Several techniques will be presented that can help you to load this services information externally.

Services configuration at compile tim

When you create a new BlazeDS or LiveCycle Data Services ES project in Flex Builder, you will typically select J2EE as the Application Server Type and then check '**Use Remote Object Access Service**' in the New Flex Project Wizard. This adds a compiler argument that specifies the location of your **services-config.xml** file. If you check the Flex Compiler properties of your Flex Builder project, you'll see something like this:

```
-services "c:\blazeds\tomcat\webapps\samples\WEB-INF\flex\services-config.xml"
```

When you then compile your application, the required values from **services-config.xml** are baked into the SWF. To abstract things a little bit, you can use tokens such as `{server.name}`, `{server.port}`, and `{context.root}` in **services-config.xml**. However, `{context.root}` is still resolved at compile time, while `{server.name}` and `{server.port}` are replaced at runtime using the server name and port number of the server that the SWF was loaded from (which is why you can't use these tokens for AIR applications).

Externalizing services configuration

Fortunately, the Flex SDK provides an API that allows you to configure your channels at runtime and entirely externalize your services configuration from your code (you definitely do not want to recompile your application when you move it to another server). **See Chapter 4 in this document, Dynamic Service Configuration, for more information**. To recap, it works like this:

```
var channelSet:ChannelSet = new ChannelSet();
var channel:AMFChannel = new AMFChannel("my-amf", "http://localhost:8400/lcds-    samples/
messagebroker/amf");
channelSet.addChannel(channel);
remoteObject.channelSet = channelSet;
```

Clearly, this is still not optimal because the endpoint URL is still hardcoded in the application. At least in this case it is obvious that this is happening. So, the last step in externalizing the services configuration is to pass that endpoint URL value at runtime. There are a number of ways you can pass values to a SWF at runtime, including flashVars and URL parameters.

You could load the services configuration from a property or xml file deployed with your Flex application. To centralize control of the configuration file and to use it for multiple applications, you could serve the configuration file from a remote Yellow Page service and use an `HTTPService` call at application startup to load service information in the Flex client. The configuration file served by the service includes (among other things) the information you need to programmatically create your channel set at runtime. Here is a basic Flex implementation that loads three channel sets from a remote services configuration file. A factory could be used to make this more reusable. This file will be loaded as soon as the application starts up:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="loadConfiguration()">
       <mx:Script>
            <![CDATA[
                 import mx.messaging.ChannelSet;
                 import mx.messaging.channels.AMFChannel;
                 import mx.messaging.channels.RTMPChannel;
                 import mx.messaging.channels.SecureRTMPChannel;
                 import mx.controls.Alert;
                 import mx.rpc.events.ResultEvent;
                 import mx.rpc.events.FaultEvent;
                 import mx.rpc.http.mxml.HTTPService;

                 private static var _rtmpChannel:RTMPChannel = null;
                 private static var _rtmpChannelSecure:SecureRTMPChannel = null;
                 private static var _amfChannel:AMFChannel = null;
                 private static var _amfPollingChannel:AMFChannel = null;

                 [Bindable]
                 public static var amfChannelSet:ChannelSet = null;
                 [Bindable]
                 public static var rtmpChannelSet:ChannelSet = null;
                 [Bindable]
                 public static var rtmpChannelSetWithFallBack:ChannelSet = null;

                 private function loadConfiguration():void
                 {
                      var httpSrv:HTTPService = new HTTPService();
                      httpSrv.url = "http://localhost:9191/helloworld/ChannelsConfiguration.xml";
                      httpSrv.showBusyCursor = true;
                      httpSrv.method = "GET";
                      httpSrv.resultFormat = "xml";
                      httpSrv.addEventListener(ResultEvent.RESULT,
                      loadConfFileResultHandler);
                      httpSrv.addEventListener(FaultEvent.FAULT,
                      loadConfFileFaultHandler);
                      httpSrv.send();
                 }

                 private function loadConfFileResultHandler(event:ResultEvent):void
```

```
                {
                    var resultXml:XML = XML(event.result);
                    parseConfigurationFile(resultXml);
                }

                private function loadConfFileFaultHandler(event:FaultEvent):void
                {
                    Alert.show(event.fault.faultString);
                }

                private function parseConfigurationFile(confXml:XML):void
                {
                    if(confXml != null)
                    {
                        //retrieving channel details from configuration file
                        var channels:XMLList = confXml.channels.channel;
                        for each(var channel:XML in channels)
                        {
                            if (channel.type == "rtmp")
                            {
                                _rtmpChannel = new RTMPChannel(channel.@id, channel.endpoint);
                            } else if (channel.type == "rtmps")
_rtmpChannelSecure = new SecureRTMPChannel(channel.@id, channel.endpoint);
                            } else if (channel.type == "amf-polling")
                            {
                                _amfPollingChannel = new AMFChannel(channel.@id, channel.endpoint);
                                _amfPollingChannel.pollingEnabled = true;
                                _amfPollingChannel.pollingInterval = channel.interval;
                            }
                            else if(channel.type == "amf")
                            {
                                _amfChannel = new AMFChannel(channel.@id, channel.endpoint);
                            }
                        }

                        //creating channel sets
                        rtmpChannelSet = new ChannelSet();
                        rtmpChannelSet.addChannel(_rtmpChannel);

                        rtmpChannelSetWithFallBack = new ChannelSet();
                        rtmpChannelSetWithFallBack.addChannel(_rtmpChannel);
                        rtmpChannelSetWithFallBack.addChannel(_amfPollingChannel);

                        amfChannelSet = new ChannelSet();
                        amfChannelSet.addChannel(_amfChannel);
                    }
                    else
                    {
                        Alert.show("Invalid configuration file");
                    }
                }

                private function getUserName():void
                {
                    rmObj.getUserName();
                }

                private function resultHandler(event:ResultEvent):void
                {
```

```
                    var value:String = "";
                    if(event.result != null && event.result is String)
                    {
                        value = event.result as String;
                    }
                    else
                    {
                        value = "No value returned";
                    }
                    Alert.show(value);
                }

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

            ]]>
        </mx:Script>
            <mx:RemoteObject id="rmObj" destination="MySessionHandler"
                channelSet="{amfChannelSet}"
             result="resultHandler(event)"
             fault="faultHandler(event)"
            showBusyCursor="true"/>
            <mx:Button label="Get name" enabled="{amfChannelSet != null}" click="getUserName()"/>

</mx:Application>
```

***Note**: With this type of runtime configuration in place, you can create plain Flex Builder projects (that is, you can select None as the Application Server Type).*

Here is the configuration file you can deploy to your web server:

```
<ChannelsConfig>
        <channels>
            <channel id="my-rtmp">
                <type>rtmp</type>
                <endpoint>rtmp://localhost:2038</endpoint>
            </channel>
            <channel id="my-rtmp-secure">
                <type>rtmps</type>
                <endpoint>rtmps://localhost:443</endpoint>
            </channel>
            <channel id="my-polling-amf">
                <type>amf-polling</type>
                <endpoint>http://localhost:9191/lcdssamples/messagebroker/amfpolling</endpoint>
                <interval>8000</interval>
            </channel>
            <channel id="my-amf">
                <type>amf</type>
                <endpoint>http://localhost:9191/lcdssamples/messagebroker/amf</endpoint>
            </channel>
        </channels>
```

```
</ChannelsConfig>
```

These services endpoints are publicly accessible, so there is no need to protect them. Though you can make secure connections to the services.

Note: There is a bug using SSL to connection to rtmps. A patch is available for 2.6.1. See http://www.mail-archive.com/flexcoders@yahoogroups.com/msg111429.html

References

This guide has been compiled from the following sources:

LiveCycle Blaze Data Services - http://opensource.adobe.com/wiki/display/blazeds/Developer+Documentation

LiveCycle Data Services 2.6 Development Guide - http://livedocs.adobe.com/livecycle/8.2/programLC/programmer/lcds

Externalizing service configuration for LiveCycle/Blaze DS - http://www.adobe.com/devnet/livecycle/articles/externalize_serviceconfig.html

Creating BlazeDS channels at runtime - http://sujitreddyg.wordpress.com/2008/07/03/creating-blazeds-channels-at-runtime/