

Getting Started with Flex 2

An introduction



By Jeffrey Houser

I'm going to postpone the second part of my RSS aggregator article to tie this column into this Flex-themed issue. Have no fears, though, it will be back in full force in the next issue.

Flex, as I'm sure most people know, is a

way for programmers (you, me, and us) to create Flash movies.

The focus of Flex is not on animation and drawing little fancy pictures; it's on creating advanced interfaces, which are used to create Rich Internet Applications (RIA). It is a "Flash for programmers"-oriented product. Macromedia had long been pushing the concept and benefits of Rich Internet Applications, so it's great to see Adobe taking up the charge and finally making them accessible to all.

Flex 2 was released at Adobe's CFUNITED keynote (a few days ago to me), so this issue seems appropriately timed. I thought I'd take the space in this beginner's column for an overview of Flex 2 and talk about why you want to care.

Putting the Pieces Together

Everyone who has seen Flex from the beginning viewed it as an amazing and revolutionary product. If you think of ColdFusion as a way to build HTML pages on the fly, Flex was a way to build Flash movies on the fly. Unfortunately, most people found the price tag to be a serious setback to Flex usage. Adobe has addressed those concerns head on with the release of Flex 2.

These are the components that make up the Flex 2 suite of products:

- **ActionScript 3:** The language of Flash has always been ActionScript. With the release of the Flash 9 Player comes a new version of ActionScript. ActionScript has always been used to provide advanced functionality in a Flash movie.
- **MXML:** MXML stands for Maximum Experience Markup Language and is a form of XML. You can use MXML to create Flash movies with Flex. Most things in ActionScript have a parallel in MXML, and vice versa.
- **Flash 9 Player:** Flex applications will run only on Flash Player 9. Flash Player 9 adds support for ActionScript 3,

and offers many performance enhancements over Flash 8.

- **Flex SDK:** The Flex SDK is everything you need to build Flex applications. It contains a command-line compiler along with all the built-in Flex components (which includes a bunch of user interface elements). You can write MXML code in any editor of your choice, use the SDK to compile it to a swf file, and then deploy it to a Web server of your choosing. There have been rumors that Adobe hopes the release of the SDK will allow for the creation of third-party tools for generating Flex applications. I haven't heard of any tools being built yet, but this definitely bodes well for the long-term release of the community. Did I mention that the Flex SDK is free?
- **Flex Builder 2:** Flex Builder 2 is an Eclipse-based editor for building Flex applications. Although you can use the Flex SDK to build them for free, Flex Builder offers many advantages, including tag insight and a step-through debugger. Flex Builder is available as a standalone product or as an Eclipse plug-in. I like to use the plug-in version so that my Flex applications can easily reside next to CFclipse applications.
- **Flex Charting Components:** The Flex charting components are available as a standalone package or as an add-on to Flex Builder. They make it easy for you to generate charts in Flex, as well as allow for drill down and roll over functionality.
- **Flex Data Services:** Flex data services allow you to push data

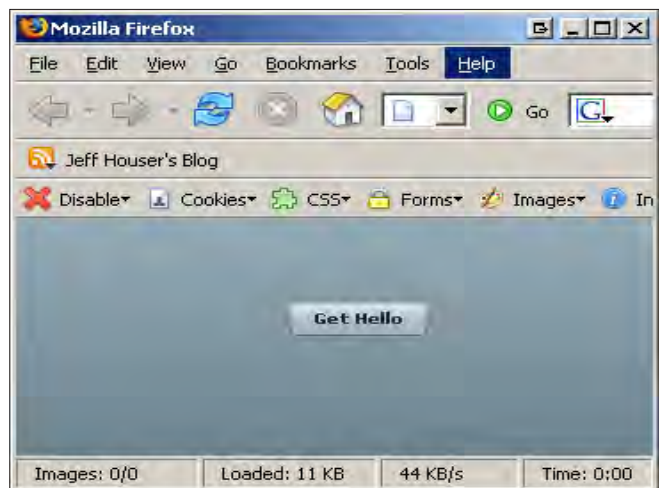


Figure 1: Hello World Pre-click

to the browser. It integrates with ColdFusion very nicely through the use of an event gateway. This is a feature that was unavailable in Flex 1.5, and can be very powerful in some applications.

Those are the important pieces of Flex. You can download the Flex components from the Adobe Website at <http://www.adobe.com/cfusion/tdrc/index.cfm?product=flex>. You can update

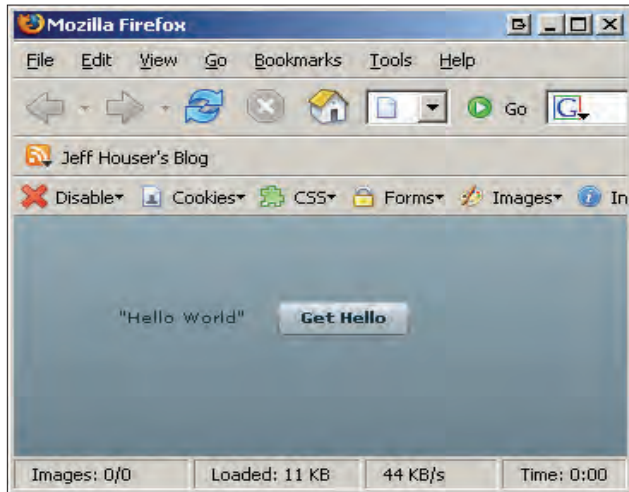


Figure 2: Hello World Post-click

your Flash player at http://www.adobe.com/shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash. For the remainder of this article, I'm going to give you an introduction in how Flex and ColdFusion work together. When working with Flex 2 and ColdFusion, you'll want to install the ColdFusion 7.02 updater. You can download that from http://www.adobe.com/support/coldfusion/downloads_updates.html.

Accessing a CFC from Flex

Flex can call CFCs directly using Flash Remoting; an update to ColdFusion's Flash Remoting components is located in the 7.02 updater. This update helps ColdFusion talk to something you built in Flex. To demonstrate, I'll start with a simple `helloWorld.cfc`:

```
<cfcomponent>
<cffunction name="GetHello" output="false" access="remote"
returntype="string">
    <cfreturn "Hello World">
    </cffunction>
</cfcomponent>
```

If you are not familiar with CFCs, there are a plethora of resources for learning about them. You might start with one of my previous columns on them at <http://coldfusion.sys-con.com/read/47203.htm>. This component has no instance variables and only contains a single "GetHello" method. The method returns

Other companies in this magazine spent a lot of time on pretty ads. As you can see, we did not. We spent our time hiring the best people and training them to deliver outstanding support for your website. We spent our time building a state of the art datacenter and staffing it with people who care about your website like it's their own. Compassion, respect, credibility, ownership, reliability, "never say no," and exceed expectations are words that describe our service philosophy. From the first time you interact with us, you'll see what a difference it really makes. And you'll also forgive us for not having a pretty ad.



WEB HOSTING • MANAGED DEDICATED SERVERS • COLOCATION • VPS • ECOMMERCE • BLOGGING • EMAIL

the string “Hello World.” This is simple stuff, and you all know it, right? Great, let’s look at some Flex code!

First, you’ll need to define a Flex application, like this:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
</mx:Application>
```

All the MXML code that you write will go in the mx:Application block. In CFML, all tags start with CF. In MXML, all tags start with “mx:”. Code blocks work the same way in either language. The mx:Application works, conceptually, the same way that a cfloop does. Now we can add a label and a button to our code:

```
<mx:Label id="Result" x="59" y="58"/>
<mx:Button x="154" y="56" label=" Get Hello"/>
```

Remember that this code goes in the mx:Application block. I used Flex Builder 2 to easily place the label and button, but if you are using the SDK without Flex Builder, you can specify the location of the elements using the x and y coordinates as shown in the code. This code will show you an empty label with a button next to it. The label, at present, doesn’t contain any text. The button displays the text “GetHello” but doesn’t actually do anything yet. The label is given an ID “result”. This is so we can reference it later to assign it a value. I did not give the button an ID because we won’t need to access it programmatically.

Next you need to tell Flex how to find your CFC. To do that, I used the RemoteObject tag and placed this code in my MXML file:

```
<mx:RemoteObject id="helloWorld" destination="ColdFusion" source="htdocs.
experiments.flex.helloworld">
<mx:method name="GetHello" result="GetHello_handler(event)" />
</mx:RemoteObject>
```

The code, once again, goes inside the application block. The RemoteObject tag has an ID, which allows us to access it in code, a destination that is a special distinguisher, and the source. The source is the Web location of your CFC. When accessing the CFC remotely, it must be Web accessible (unless you change settings to allow you to access CFCs via a ColdFusion mappings, but such a configuration is beyond the scope of this article). Inside the RemoteObject block there is one method that we respond to: “GetHello”. The mx:method tag accepts two arguments: a name and the result. The name is the name of the method on the CFC. The result is the name of a local method that will be called when the Flash Player gets the results from calling that event.

The next step in our code base is to write the GetHello_Handler, which is written in ActionScript. You can put ActionScript in a MXML page using the mx:Script tag:

```
<mx:Script>
<![CDATA[
import mx.rpc.events.ResultEvent;
import mx.utils.ObjectUtil;

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

```
Result.text = ObjectUtil.toString(event.result);
}
]]>
</mx:Script>
```

After the script tag comes the CDATA. This tells the XML parser to ignore the text in the script tag. ActionScript is not a valid XML dialect. (I’m not saying that’s bad, though.) Then I import the two objects that are used in the function. Importing objects in this manner is not common in ColdFusion development, but if you’ve worked with older versions of ActionScript or Java, you’ve probably seen it. It just says, “I need this object, so make it available to me.”


The function should look similar to a CFScript function; I specified the function as private. Then comes the function keyword, followed by the name of the function. Next comes the list of arguments. This function only has a single argument, the ResultEvent. Then comes a colon followed by the return type. This function doesn’t return anything. The one line of code takes the result from our function call, translates it to a string, and assigns it to the text of our Result label. It sounds more complicated than it actually is.

Type in the code (or copy and paste from the Web version), compile it, and execute it. You should see something similar to Figure 1. Click the button. Unfortunately nothing happened as the button was not told what to do yet. A click event needs to be added to the button. The new button code will look like this:

```
<mx:Button x="154" y="56" label="Get Hello" click="helloWorld.GetHello()"/>
```

The click event refers to the helloWorld remote object and says, “Execute the GetHello” method on that object. You should recognize this syntax for calling the method from your use of CFCs inside ColdFusion. When the button is clicked, the Flash player goes to the remote object and calls the method. When the method result is returned, the Flash player looks for the “mx:method” tag and executes the result function. Recompile the code and try it out. Click the button and you should see the Hello World text display next to the button.

Conclusion

Adobe has done a fantastic job of making ColdFusion the best back-end tool for developing Flex applications. Included with Flex Builder are some Eclipse extensions that work well with ColdFusion, including RDS support and some code generators. I’m just scratching the surface of what can be done with Flex and how you can combine it with ColdFusion. I’d love to see what you are going to do with this technology, so be sure to let me know! See you in a month. 

About the Author

Jeff Houser has been working with computers for over 20 years. He owns a DotComIt, a web consulting company, manages the CT Macromedia User Group, and routinely speaks and writes about development issues. You can find out what he’s up to by checking his Blog at www.jeffryhouser.com.

jeff@instantcoldfusion.com