

Integrating BlazeDS and ColdFusion 9

a hands-on BYOL (bring your own laptop) lab @ Adobe MAX 2009
October 4 - 7, Los Angeles, California



Instructor:

Aaron S. West

a.west@me.com

<http://www.trajiklyhip.com/blog>

Teaching Assistants:

Yancy Wharton

Rakshith N

These lab instructions and all code were written by Aaron S. West. Last revised: September 25, 2009

Introduction	3
Lab Description	3
Instructor Assumptions	3
Software Prerequisites	3
Ensuring Your Computer is Ready	5
Important Note About the Walkthrough Exercises	6
Walkthrough 1 - Creating a New Flash Builder Project	6
Walkthrough 2 - Configuring BlazeDS	7
Walkthrough 3 - Creating the ColdFusion Code in ColdFusion Builder	11
Walkthrough 4 - Creating an HTML form the application will use to send messages to BlazeDS	13
Walkthrough 5 - Adding HTML Script Includes Needed to Send Messages	14
Walkthrough 6 - Creating the jQuery Form Submission Handler	15
Walkthrough 7 - Creating a ColdFusion CFC That Functions as an Event Gateway to BlazeDS	20
Walkthrough 8 - Defining the Event Gateway Instance in the ColdFusion Administrator	22
Walkthrough 9 - Creating the onIncomingMessage Function to Handle Message Receipt	23
Walkthrough 10 - Creating a Message Consumer Flex Application Using Flash Builder	25
(Optional) Walkthrough 11 - Adding the Flex Application to the HTML Page	29

Introduction

This 90 minute hands-on BYOL (bring your own laptop) lab was written especially for Adobe MAX 2009. The purpose of the lab is to expose students to BlazeDS and ColdFusion 9 by building an application using ColdFusion Builder and Flash Builder 4. Welcome to the lab, we hope you enjoy it!

Lab Description

Learn how BlazeDS and ColdFusion 9 (code-named “Centaur”) can be used to create a real-world, Flex-based messaging service. During this lab you will create a Flex-based Flash application and an HTML page, from start to finish, that communicates with ColdFusion and BlazeDS to consume a one-way, real-time messaging service.

Instructor Assumptions

The following assumptions have been made of all lab attendees by the instructor.

- you are using a Windows or Mac computer
- you have ALL the software prerequisites (listed below) installed and functioning properly (VERY important)
- you are an intermediate (or above) developer with a good understanding of basic CFML, ActionScript 3, and MXML

Software Prerequisites - IMPORTANT, please read!

This lab requires you have certain software already installed and functioning properly on your Windows or Mac computer. Please review the following prerequisites *before* you show up for the lab. We will not have time to ensure your computer are working properly.

1. Adobe ColdFusion 9

We will use ColdFusion 9 and the integrated version of BlazeDS as our application server during this lab. At the time of this writing ColdFusion 9 was in public beta and available for download at the following URL. I recommend attendees install ColdFusion using the multiserver option at install time. This will allow you to install multiple instances of ColdFusion should you ever want to do so. During the installation you will be asked to choose an external web server or the built-in web server. You may choose either but I will be working with the built-in web server for simplicity purposes.

<http://labs.adobe.com/technologies/coldfusion9/>

2. Eclipse Classic Development Environment (version 3.4 or 3.5)

You will need to download and install the Classic edition of the Eclipse development environment if you don't have it installed already. Version 3.4 (Ganymede) or 3.5 (Galileo) will work fine. We will use Eclipse as the foundation of all our development installing other software into Eclipse as plugins. *NOTE: If you are on a Mac you must download the Carbon version of Eclipse Classic. The Cocoa version is not supported by Flash Builder 4.*

Eclipse Classic 3.5 (Windows or Mac Carbon 32-bit):

<http://www.eclipse.org/downloads/packages/>

Eclipse Classic 3.4 (Windows or Mac Carbon 32-bit):

<http://www.eclipse.org/downloads/packages/release/ganymede/sr2>

3. Adobe ColdFusion Builder

ColdFusion Builder will be used to create and edit CFML and JavaScript. At the time of this writing ColdFusion Builder was in public beta and available for download at:

<http://labs.adobe.com/technologies/coldfusionbuilder/>

4. Adobe Flash Builder 4

Flash Builder 4 will be used to create and edit MXML and ActionScript 3 code. At the time of this writing Flash Builder 4 was in public beta and available for download at:

<http://labs.adobe.com/technologies/flashbuilder4/>

5. Adobe Flash Player 10 (Debugger version)

Adobe Flash Player 10 is required for the Flex application to run properly. I highly recommend you download the debugger version of Flash Player 10 from the following Web page: <http://www.adobe.com/support/flashplayer/downloads.html>

After you have installed the latest version of Flash Player 10, close all your browsers and then restart them. Visit the following Web page to ensure you are running FP10: http://kb2.adobe.com/cps/155/tn_15507.html You should see something similar to the following:

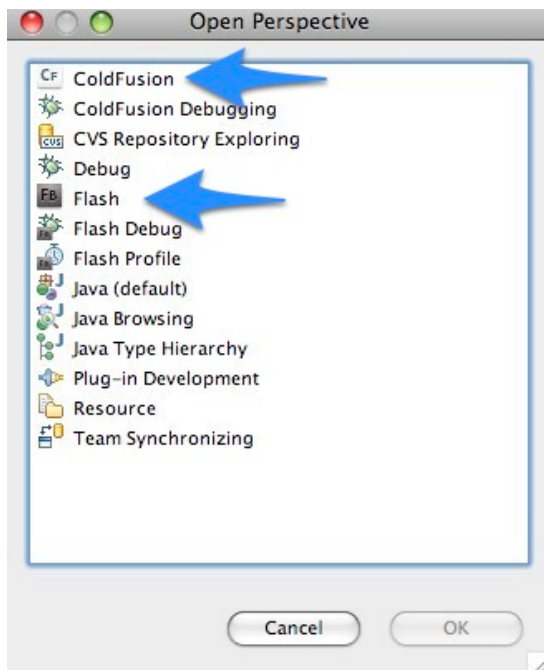


6. *Lab / exercise files available on the DVD. You should've received the DVD when you checked into the conference.*

Ensuring Your Computer is Ready

Verify ColdFusion 9 is installed and working properly. You should be able to access the ColdFusion Administrator as well as write a simple ColdFusion template to make sure - for instance - a cfoutput works. Please make a note of your webroot location as we will be working in this location extensively.

Verify Eclipse, ColdFusion Builder, and Flash Builder are installed and working properly. You should be able to launch Eclipse as well as open the Flash and ColdFusion perspectives within Eclipse.



Opening a perspective in Eclipse (Window - Open Perspective - Other)

Important Note About the Walkthrough Exercises

The walkthrough exercises that make up the rest of the content of this lab are meant to be followed sequentially. During the lab we will start with Walkthrough 1 and progress to the end. Given our 90 minute time limit we may not finish all the lab content however, the instructions have been written so you can complete all walkthroughs on your own at a later date.

If you get stuck on any one walkthrough don't worry too much about it. You can get back on track with the start of the next walkthrough by simply copying the contents of that walkthrough folder into the root folder of your lab project. For example: if you become stuck on walkthrough 3 and the lab is continuing to walkthrough 4 you can get on track by performing the following steps:

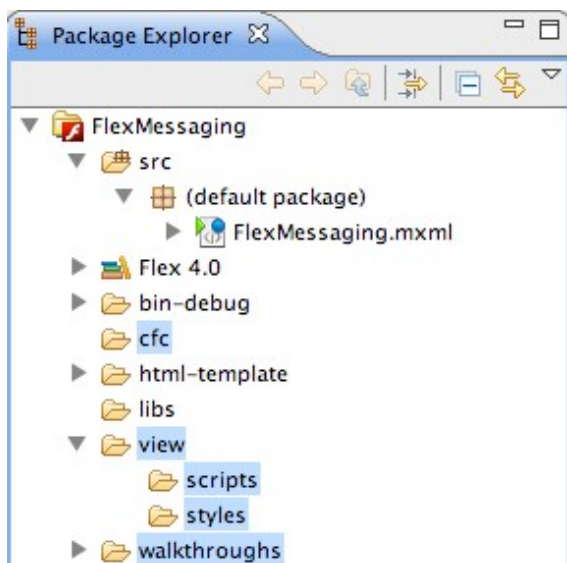
1. Delete the *cfc* folder in your project and replace it with the *cfc* folder in the *walkthroughs/wt-4/* folder
2. Delete the *view* folder in your project and replace it with the *view* folder in the *walkthroughs/wt-4/* folder
3. Delete the *FlexMessaging.mxml* file in your projects src folder and replace it with the file of the same named stored in *walkthroughs/wt-4/src/* folder

Walkthrough 1 - Creating a New Flash Builder Project

In this walkthrough we will create a new Flex project in Flash Builder and import the lab assets.

1. Launch the Eclipse IDE if you have not done so already
2. Close any open projects
3. Switch to the Flash Builder perspective: *Window - Open Perspective - Other*, then choose "Flash."
4. Create a new Flex project
 1. File - New - Flex Project
 2. Project name: FlexMessaging
 3. Project location: <your_web_root>/FlexMessaging
 4. Application type: Web (runs in Adobe Flash Player)
 5. Flex SDK version: Flex 4.0 (this should be selected by default)
 6. Application server type: None/Other
 7. Press Finish
5. Create additional project folders and files
 1. Right-click (command-click on Mac) your root project folder and choose New - Folder. Name the folder "cfc"

2. Right-click your root project folder and create a new folder called “view”
3. Right-click the view folder and choose New - Folder. Name the folder “scripts”
4. Right-click the view folder once again and choose New - Folder. Name the folder “styles”
6. Copy the lab assets into your project
 1. Locate the “walkthroughs” folder on the MAX content DVD. Or, if you have already have the lab assets on your computer, locate that directory.
 2. Copy the “walkthroughs” directory from Windows Explorer or Mac’s Finder and paste the directory into the root of your FlexMessaging project.
7. Close the FlexMessaging.xml file if it opened. We won’t be needing this file until later.



Your FlexMessaging project should look like this after this walkthrough exercise. Folders in blue are the ones you should’ve added to your project.

Walkthrough 2 - Configuring BlazeDS

BlazeDS Community Edition 3.2.0.3978 comes bundled with ColdFusion 9. This is different from ColdFusion 8 where LiveCycle Data Services Express was included. To use BlazeDS with ColdFusion 8 you had to remove the LCDS files and replace them with the BlazeDS counterparts by manually adding and removing directories and files. Since ColdFusion 9 includes BlazeDS none of this work is necessary. However, we will perform some minor configuration of the BlazeDS server in order to lay the foundation for our messaging-based application.

1. Locate the WEB-INF directory of your ColdFusion 9 webroot. Regardless of your operating system and version of ColdFusion installed (server configuration, multi-

server configuration) this directory will be located at <your_coldfusion_webroot>\WEB-INF. Actual example directories include but are not limited to:

Server Configuration on Windows

C:\ColdFusion9\wwwroot\WEB-INF\

Multi-server Configuration on Windows

C:\JRun4\servers\<instance_name>\cfusion-ear\cfusion-war\WEB-INF\

Server Configuration on Mac

/Applications/ColdFusion9/wwwroot/WEB-INF/

Multi-server Configuration on Mac

/Applications/JRun4/servers/<instance_name>/cfusion-ear/cfusion-war/WEB-INF/

2. BlazeDS configuration files will be located in <your_coldfusion_webroot\WEB-INF\flex\>
 1. messaging-config.xml (configuration file for the Messaging portion of BlazeDS)
 2. proxy-config.xml (configuration file for BlazeDS proxy settings)
 3. remoting-config.xml (configuration file for the Remoting portion of BlazeDS)
 4. services-config.xml (the main BlazeDS configuration file)
3. (Instructor demo of services-config.xml)
4. Open your services-config.xml and add the following long-polling channel definition. For your convenience the code for the channel has been provided in the lab assets.
 1. In Eclipse, make sure you are still in the Flash perspective
 2. Expand the /walkthroughs/wt-2/ folder in your FlexMessaging project
 3. Open the *services-config-channel.xml* file
 4. Copy the entire contents of the file and paste them into your *services-config.xml* file just after the existing *cf-polling-amf* channel
5. Scroll to the bottom of services-config.xml
6. Towards the bottom of the file will be a <logging> section
7. You should see the following line of code in this section

```
<target class="flex.messaging.log.ConsoleTarget" level="Error">
```

8. Change the log level by replacing the word “Error” with the word “All.” This will allow you to see all errors and messages coming across the BlazeDS server in your console (Windows) or Terminal (Mac) window.

```
<target class="flex.messaging.log.ConsoleTarget" level="All">
```

9. Save services-config.xml


```

<channel-definition id="cf-longpolling-amf"
class="mx.messaging.channels.AMFChannel">
  <endpoint uri="http://{server.name}:{server.port}/{context.root}/
flex2gateway/cfamflongpolling"
class="coldfusion.flash.messaging.CFAMFEndPoint"/>
  <properties>
    <polling-enabled>true</polling-enabled>
    <polling-interval-seconds>3</polling-interval-seconds>
    <wait-interval-millis>60000</wait-interval-millis>
    <client-wait-interval-millis>1</client-wait-interval-millis>
    <max-waiting-poll-requests>200</max-waiting-poll-requests>
    <serialization>
      <enable-small-messages>>false</enable-small-messages>
    </serialization>
    <coldfusion>
      <!-- define the resolution rules and access level of the cfc
being invoked -->
      <access>
        <!-- Use the ColdFusion mappings to find CFCs-->
        <use-mappings>true</use-mappings>
        <!-- allow "public and remote" or just "remote"
methods to be invoked -->
        <method-access-level>remote</method-access-level>
      </access>

      <!-- Whether the Value Object CFC has getters and setters.
Set the value of use-accessors to true if there are getters and setters in
the Value Object CFC. -->
      <use-accessors>true</use-accessors>
      <!--Set the value of use-structs to true if you don't
require any translation of ActionScript to CFCs. The assembler can still
return structures to Flex, even if the value is false. The default value is
false.-->
      <use-structs>>false</use-structs>

      <property-case>
        <!-- cfc property names -->
        <force-cfc-lowercase>>false</force-cfc-lowercase>
        <!-- Query column names -->
        <force-query-lowercase>>false</force-query-lowercase>
        <!-- struct keys -->
        <force-struct-lowercase>>false</force-struct-lowercase>
      </property-case>
    </coldfusion>
  </properties>

```

```
</channel-definition>
```

10. (Instructor demo of messaging-config.xml)
11. Open your messaging-config.xml and add the following *cfflexmessaging* destination. For your convenience the code for the destination has been provided in the lab assets.
 1. Open the *messaging-config-destination.xml* file
 2. Copy the entire contents of the file and paste them into your *messaging-config.xml* file just after the last line of code (</service>)
 3. Save messaging-config.xml

```
<destination id="cfflexmessaging">
  <properties>
    <gatewayid>FlexMessaging</gatewayid>
  </properties>
  <channels>
    <channel ref="cf-longpolling-amf"/>
    <channel ref="cf-polling-amf" />
  </channels>
</destination>
```

12. In order for the changes you have made to services-config.xml and messaging-config.xml to take affect you must restart the ColdFusion server
 1. On Windows perform the following steps
 1. Note: You would normally start/stop ColdFusion using the *Services* tool in *Control Panel - Administrative Tools - Services*. For the purposes of this lab we need to start ColdFusion from MS-DOS.
 2. Launch an MS-DOS prompt by pressing *Start - Run* and typing "cmd" in the resulting box.
 3. Press Ok
 4. Change to the "bin" directory of your ColdFusion installation.
 1. Server configuration: cd C:\ColdFusion9\bin <press enter>
 2. Multi-server configuration: cd C:\JRun4\bin\ <press enter>
 5. Start ColdFusion
 1. Server configuration: cfstart <press enter>
 2. Multi-server configuration: jrun start <instance_name> & <press enter>
 2. On Mac perform the following steps
 1. Launch Terminal from /Applications/Utilities/. Double-click Terminal.app
 2. Change to the "bin" directory of your ColdFusion installation.
 1. Server configuration: cd /Applications/ColdFusion9/bin <press enter>
 2. Multi-server configuration: cd /Applications/JRun4/bin <press enter>
 3. Start ColdFusion
 1. Server configuration:

1. `sudo coldfusion start` <press enter>
2. enter your OS X password <press enter>
2. Multi-server configuration:
 1. `sudo ./jrun start <instance_name> &` <press enter>
 2. enter your OS X password <press enter>
13. When ColdFusion starts up text that wasn't showing before will now be present due to the log level change in services-config.xml file. It will look something like this:

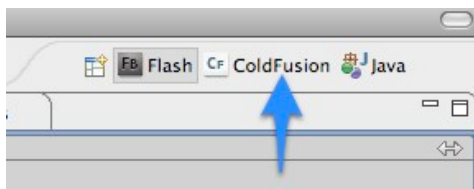
```
09/20 19:08:32 user MessageBrokerServlet: init
[BlazeDS]BlazeDS - Community Edition: 3.2.0.3978
[BlazeDS]Endpoint java-polling-amf created with security: None
at URL: http://{server.name}:{server.port}/{context.root}/flex2gateway/amfpolling
[BlazeDS]Endpoint cf-longpolling-amf created with security: None
at URL: http://{server.name}:{server.port}/{context.root}/flex2gateway/cfamflongpolling
[BlazeDS]Endpoint cf-polling-amf created with security: None
at URL: http://{server.name}:{server.port}/{context.root}/flex2gateway/cfamfpolling
```

14. If you see any error messages or your ColdFusion server does not start properly, please raise your hand and a teaching assistant will help you
15. Keep the console (Windows) or Terminal (Mac) open. We will refer back to this window in a later walkthrough exercise.

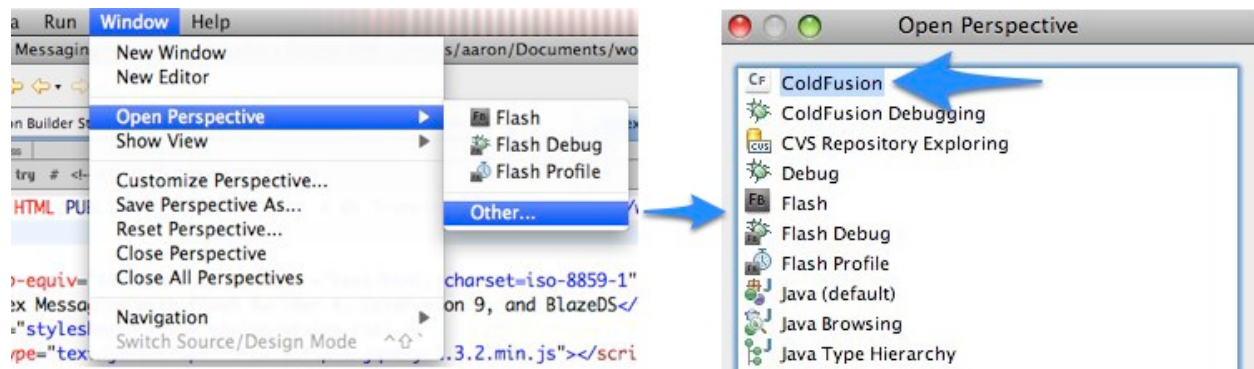
Walkthrough 3 - Creating the ColdFusion Code in ColdFusion Builder

In this walkthrough you will use ColdFusion Builder to create the initial CFML code of your messaging application. This will consist of a really basic Application.cfc component and the beginnings of the index.cfm template.

1. Switch to the ColdFusion perspective by pressing the ColdFusion button in the upper right-hand corner of Eclipse or by accessing *Window - Open Perspective - Other - ColdFusion*



Switching to the ColdFusion perspective using the perspective buttons



Switching to the ColdFusion perspective using the menus

2. Create an Application.cfc component
 1. Right-click the view folder in your project and select *New - ColdFusion Component*.
 2. Component Name: Application (you do not need to type ".cfc" after the name)
 3. Press the *Finish* button
 4. Close the Application.cfc component. We won't be using it, but we need it to ensure another Application.cfc file isn't executed.
3. Create the index ColdFusion template
 1. Right-click the view folder in your project and select *New - ColdFusion Page*
 2. Name: index (you do not need to type ".cfm" after the name)
4. Add the following code to the index.cfm file. I suggest copying the code from the *walkthroughs/wt-4/view/index.cfm* template into your project index.cfm template.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Flex Messaging with Flash Builder 4, ColdFusion 9, and
BlazeDS</title>
<body>

</body>
</html>
```

5. Save the index.cfm file
6. Test your work by accessing the index.cfm template in a Web browser
 1. The URL you need to load should be *<your_web_root>/FlexMessaging/view/index.cfm*

2. You should see a blank page with no errors. The title of the page in the browser should be *Flex Messaging with Flash Builder 4, ColdFusion 9, and BlazeDS*

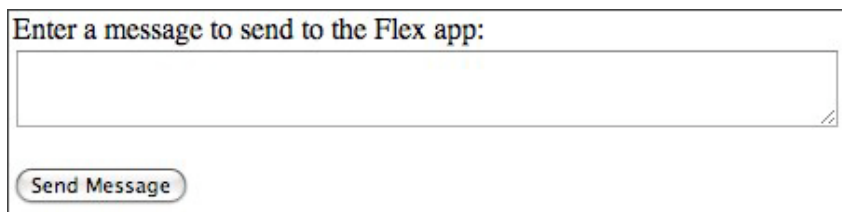
Walkthrough 4 - Creating an HTML form the application will use to send messages to BlazeDS

Your messaging application needs a way to communicate with BlazeDS. While some server-side code will be instrumental in this process we need some place for the message to originate. In this walkthrough you'll create a basic HTML form users will type messages into.

1. You should have index.cfm opened from the previous walkthrough. If not, open index.cfm which is located in the view folder.
2. Type the following code just after the opening <body> tag

```
<!-- Show a form for sending new messages to BlazeDS. -->  
Enter a message to send to the Flex app:<br/>  
<form action="index.cfm" method="post" id="frmMessage">  
    <textarea id="message" rows="3" cols="60"></textarea><br/>  
><br/>  
    <input type="submit" value="Send Message">  
</form>
```

3. Save the index.cfm file
4. Test your work by accessing index.cfm in a browser: <your_web_root>/FlexMessaging/view/index.cfm
 1. You should see something similar to the following



Enter a message to send to the Flex app:

Send Message

Walkthrough 5 - Adding HTML Script Includes Needed to Send Messages

Your messaging application now has a way for users to type messages that will be sent to BlazeDS. But the form doesn't have an action page. Instead of creating a traditional action page (separate CFML template) we will use the jQuery Ajax library to intercept the form submission and send the users typed message to ColdFusion/BlazeDS.

1. Create the styles.css stylesheet
 1. Right-click the empty styles folder located inside the view folder and select *New - File*
 2. File name: styles.css
 3. Press Finish
2. A new file named styles.css should be created inside the */view/styles/* folder and the empty file should be opened in ColdFusion Builder.
3. Add the following basic stylesheet code to styles.css

```
body {  
    font-family: verdana;  
    font-size: 12pt;  
    color: #000000;  
}
```

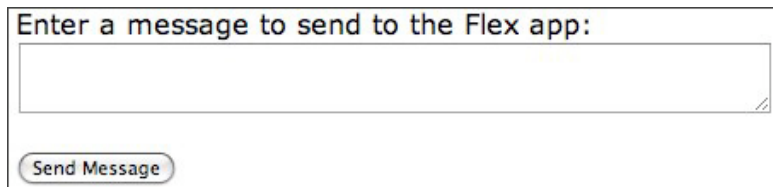
4. Save the styles.css file then close the file
5. Open index.cfm if it isn't open already
6. Before we add our form submission handlers let's change some basic styles of our application. To do this type the following stylesheet include code just after the <title> tag and before the <body> tag.

```
<link rel="stylesheet" href="styles/styles.css" />
```

7. Now you need to add the jQuery include to index.cfm
 1. Type the following code after the <link> tag and before the <body> tag. This line of code will include a minified version of the jQuery library in our index page.

```
<script type="text/javascript" src="scripts/jquery-1.3.2.min.js"></script>
```

8. In order for the previous `<script>` include to work you'll need to add the jQuery JavaScript file to your *scripts* folder. To do this, copy the *jquery-1.3.2.min.js* file from the completed walkthroughs folder (*/walkthroughs/wt-6/view/scripts/*) into your */view/scripts/* folder.
9. Save the *index.cfm* file
10. Test your work by accessing *index.cfm* in a browser: `<your_web_root>/FlexMessaging/view/index.cfm`
 1. The text above the HTML form should change slightly since the stylesheet we added included style changes for the `<body>` tag.



Walkthrough 6 - Creating the jQuery Form Submission Handler

In this walkthrough you will create a JavaScript file that uses jQuery to process the form submission in *index.cfm*. The jQuery code will intercept the form, ensure a message was typed into the textarea, and send the message string to a ColdFusion CFC in a Web Service call.

1. Open *index.cfm* if it isn't open already
2. Add a new JavaScript include after the existing jQuery `<script>` tag and before the `<body>` tag

```
<script type="text/javascript" src="scripts/FlexMessaging.js"></script>
```

3. Save the *index.cfm* file
4. In order for the previous JavaScript include to work you need to create the *FlexMessaging.js* file
 1. Right-click the */view/scripts/* folder and select *New - File*
 2. File name: *FlexMessaging.js*
 3. Press Finish
5. A new file named *FlexMessaging.js* should be created inside the */view/scripts/* folder and the empty file should be opened in ColdFusion Builder.
6. Add the following jQuery code to *FlexMessaging.js*. This code ensures all the programming we add to *FlexMessaging.js* will not execute until the browsers Document Object Model (DOM) is finished loading.

```
$(document).ready(function() {  
  
});
```

7. Create a submit handler function for the HTML form with the ID *frmMessage*.
 1. This code uses jQuery notation to “bind” a submit event handler to an HTML form with ID *frmMessage*.
 2. In other words, any time a users presses the Send Message button of the form this function will execute.

```
// Bind the submit button of the message form.  
$('#frmMessage').submit(function() {  
  
});
```

8. Add the following code to the body of the submit event handler.
 1. You’ll first ensure the user types something into the HTML textarea. The if statement uses jQuery notation to ensure the value of the HTML element with the name *message* has a value greater than one. This element name corresponds to the ID of the <textarea> in the index.cfm file.
 2. Finally, as long as the user enters a message into the textarea you’ll call a JavaScript function named *submitMessage()*.

```
// Ensure the user has typed something into the message box.  
if ($('#message').val().length < 1) {  
    alert("Please enter a message in the provided box.");  
} else {  
    submitMessage();  
}  
return false;
```

9. Create the *submitMessage()* JavaScript function
 1. Type the following code into the FlexMessaging.js file after the existing submit event handler function

```
// Send the message to the CFC via Ajax.  
function submitMessage($obj) {  
  
}
```


10. Type the following code into the body of the submitMessage() function. This code creates the beginnings of an Ajax call your application will make to a ColdFusion Component (we will create the component in later walkthroughs).

```
$.ajax({
    url: '../cfc/FlexMessaging.cfc?wsdl&method=createMessage',
    cache: false,
    type: 'GET',
    returnFormat: "JSON",
    data: {message: $('#message', $obj).val()},
})
```

11. Your submitMessage function isn't complete as it's missing two important pieces, a success handler and an error handler.

1. Add a success handler to the Ajax call by typing the following just after the line that says "data: {message:..."

```
success: function(data) {
    // If the call to the server happens without an issue this
    // success function will execute.
    clearForm();
    var obj = eval(data);
    var returnData = obj[0].SENTAT + " -- " + obj[0].MESSAGE +
    "<br/>";
    $('#messageList').append(returnData);
},
```

2. Add an error handler to the Ajax call by typing the following just after the code you added in the previous step.

```
error: function() {
    // If the call to the server generates an error, this
    // failure function will execute.
    alert('There was an error communicating with the Web
    Service.');
```

12. The full submitMessage() function should now have the following code. If you want to copy the complete function body from working files feel free to do so. A finalized working copy of FlexMessaging.js with the submitMessage() function is in the /walkthroughs/wt-7/view/scripts/ folder.

```
// Send the message to the CFC via Ajax.
function submitMessage($obj) {
    $.ajax({
        url: '../cfc/FlexMessaging.cfc?
wsdl&method=createMessage',
        cache: false,
        type: 'GET',
        returnFormat: "JSON",
        data: {message: $('#message', $obj).val()},
        success: function(data) {
            // If the call to the server happens without an
            issue this success function will execute.
            clearForm();
            var obj = eval(data);
            var returnData = obj[0].SENTAT + " -- " +
obj[0].MESSAGE + "<br/>";
            $('#messageList').append(returnData);
        },
        error: function() {
            // If the call to the server generates an error,
            this failure function will execute.
            alert('There was an error communicating with the
Web Service.');
```

13. With the submitMessage() function complete we're almost done editing FlexMessaging.js. Notice how the success event in the Ajax call runs a function called clearForm(). We need to create this function which will clear the message typed into the textarea if and only if the Ajax call to ColdFusion is a success.

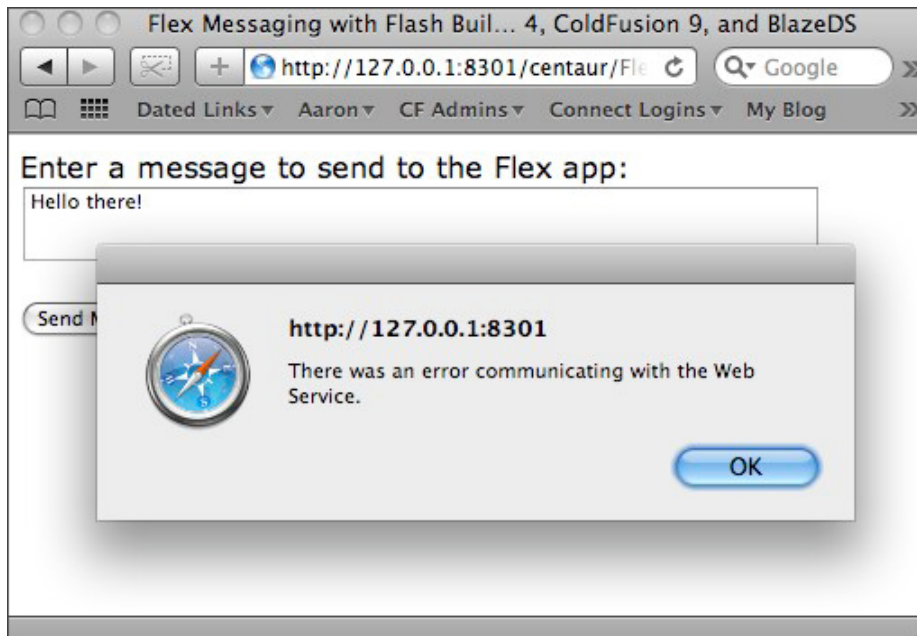
1. Create the clearForm() function in FlexMessaging.js by typing the following code after the submitMessage() function

```
// Clear the form after a message has been successfully received
by the server.
function clearForm() {
    $('#message').val('');
}
```

14. The FlexMessaging.js file is now complete. Feel free to save it.
15. Before you test your work notice the last line in the success handler. Basic jQuery code is used to target an HTML entity with the ID messageList and append to that entity the date/time the message was sent and the message string itself.
 1. Switch to your index.cfm template and type the following code just after the closing </form> tag and before the closing </body> tag

```
<p />
<!-- This is where sent messages will show. --->
<div id="messageList"></div>
```

16. Save the index.cfm file if you haven't done so already
17. Save the FlexMessaging.js file if you haven't done so already
18. Test your work in this walkthrough by accessing index.cfm in a browser:
 <your_web_root>/FlexMessaging/view/index.cfm
 1. Type a message into the textarea and press the Send Message button
 2. An error should display indicating the Ajax call to the ColdFusion Component failed.
 3. This makes sense since we haven't yet created the CFC
 4. Also note the clearForm() JavaScript function didn't execute because of the Ajax failure
 5. If you don't see the alert error message pop up, check your work. You might want to copy the working version of FlexMessaging.js from */walkthroughs/wt-7/view/scripts/* into your project's */view/scripts/* folder.



Walkthrough 7 - Creating a ColdFusion CFC That Functions as an Event Gateway to BlazeDS

Our messaging application will communicate to BlazeDS using a ColdFusion Event Gateway (CFC). In this walkthrough you will create the ColdFusion Event Gateway CFC, writing the code needed to send messages via BlazeDS.

1. The *cfc* folder in the root of your FlexMessaging project should be empty.
2. Right-click the *cfc* folder and select *New - ColdFusion Component*
 1. Component Name: FlexMessaging (you do not need to type “.cfc” after the name)
 2. Hint: Uses ColdFusion 9 and BlazeDS for messaging
 3. Output: false
 4. Press Finish
3. A new file named FlexMessaging.cfc should be created inside the */cfc/* folder and the file should be opened in ColdFusion Builder.
4. Create a new function called *createMessage*. This function will be called from the Ajax GET request in the FlexMessaging.js file.
 1. Create the createMessage function by typing the following code

```
<cffunction name="createMessage" access="remote"
returnFormat="JSON">

</cffunction>
```

2. Add the *message* argument to the function

```
<cfargument name="message" required="true" type="String">
```

3. Create the *messageBody* Array that will hold the message sent to BlazeDS. Create the *blazeMessage* structure. This will hold the *messageBody* Array as well as other critical information the BlazeDS server needs.

```
<cfset var messageBody = ArrayNew(1)>  
<cfset var blazeMessage = StructNew(>
```

4. Add properties to the *messageBody* Array

```
<cfset messageBody[1] = StructNew(>  
<cfset messageBody[1].sentAt = DateFormat(Now(), "mm-dd-yyyy") &  
" " & TimeFormat(Now(), "hh:mm tt")>  
<cfset messageBody[1].message = arguments.message>
```

5. Store the *messageBody* Array as a property of the *blazeMessage* structure

```
<cfset blazeMessage.body = messageBody[1]>
```

6. Define the BlazeDS Destination your application will connect with

```
<cfset blazeMessage.Destination = "cfflexmessaging">
```

7. Send the users message to the event gateway using *SendGatewayMessage()*

```
<cfset SendGatewayMessage("FlexMessaging", blazeMessage)>
```

8. Return the *messageBody* Array so the HTML portion of your application can be updated with a new message. The return value will automatically be formatted as JSON XML using the *returnFormat* attribute of the *cffunction* tag. This was a feature added to ColdFusion 8.

```
<cfreturn messageBody>
```

9. Save your FlexMessaging.cfc component

Walkthrough 8 - Defining the Event Gateway Instance in the ColdFusion Administrator

The createMessage function in the FlexMessaging CFC uses SendGatewayMessage() to send messages to an event gateway called FlexMessaging. We need to create this gateway instance.

1. Go to your Web browser and open the ColdFusion Administrator. The URL for your ColdFusion Administrator is typically:

[http://127.0.0.1:\[port_number\]/\[coldfusion_instance_name\]/CFIDE/administrator/index.cfm](http://127.0.0.1:[port_number]/[coldfusion_instance_name]/CFIDE/administrator/index.cfm)

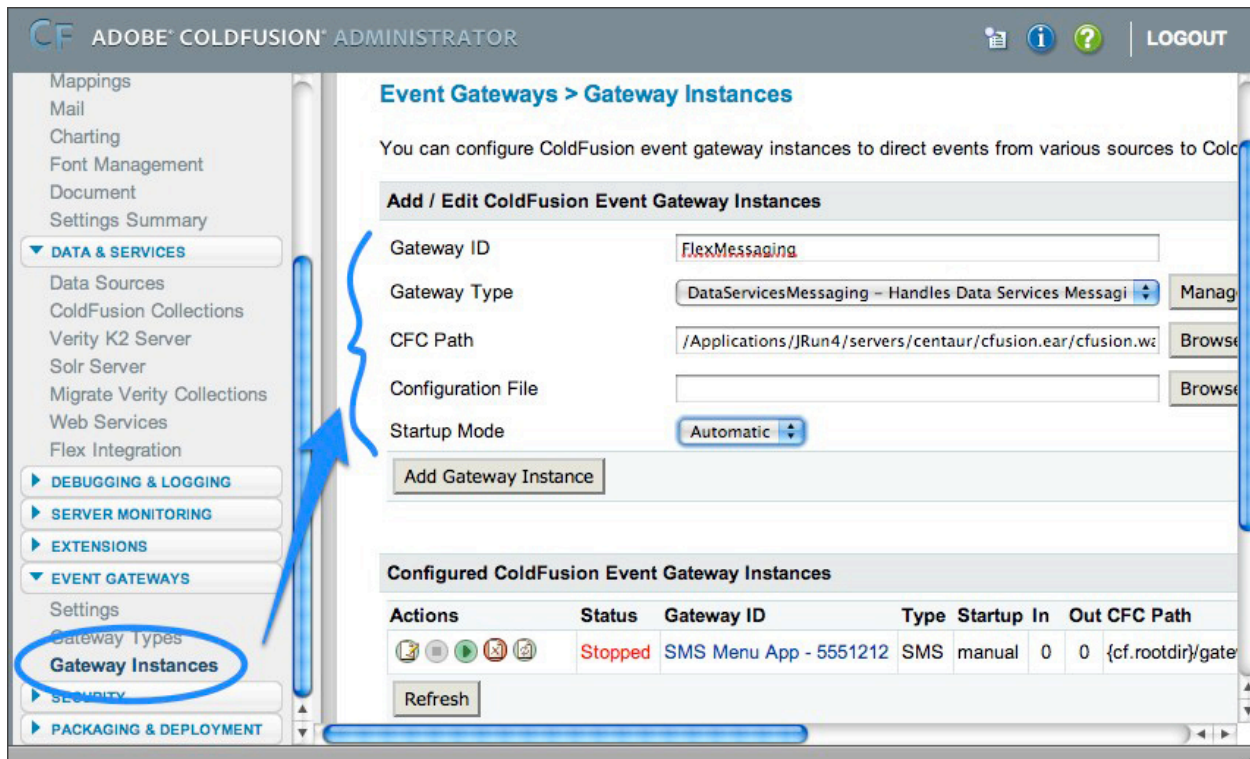
Instructor's ColdFusion Administrator (using the built-in Web server on port 8301 and an instance name of centaur):

<http://127.0.0.1:8301/centaur/CFIDE/administrator/index.cfm>

If you have "hooked" ColdFusion to an external Web server using the Web Server Connector your URL might look something like:

<http://127.0.0.1/CFIDE/administrator/index.cfm>

2. Log-in to your ColdFusion Administrator
3. Choose Event Gateways in the left hand navigation
4. Select Gateway Instances
5. Create a new ColdFusion Event Gateway Instance
 1. Gateway ID: FlexMessaging (make sure there are no spaces in this name)
 2. Gateway Type: DataServicesMessaging
 3. CFC Path: [choose the path to the FlexMessaging.cfc we created in this walkthrough]
 4. Configuration File: leave this blank
 5. Startup Mode: Automatic (this should be selected by default)
 6. Press the Add Gateway Instance button



7. The FlexMessaging gateway you created will be created and will have a Status of *Stopped*
8. Start the event gateway by pressing the green button with a right facing arrow. It looks like a "play" button.

Walkthrough 9 - Creating the onIncomingMessage Function to Handle Message Receipt

In this walkthrough you will continue working with your FlexMessaging.cfc component. You will add the final function needed in this component, a function that has a special event gateway specific name designed to be used with event gateway instances.

1. Return to your FlexMessaging.cfc component
2. Add a new function called *onIncomingMessage*

```
<cffunction name="onIncomingMessage" access="public"
returntype="struct">

</cffunction>
```

3. Add the *messageArg* argument to the function

```
<cfargument name="messageArg" type="struct" required="true">
```

4. Create a new structure with three properties
 1. msg.body will contain the date/time the message was sent and the message itself
 2. msg.destination will be FlexMessaging based on the SendGatewayMessage() function all in the createMessage function
 3. msg.headers will contain the BlazeDS headers needed for message direction

```
<cfset var msg = StructNew(>
<cfset msg.body = arguments.messageArg.data.body>
<cfset msg.destination = arguments.messageArg.data.destination>
<cfset msg.headers['gatewayid'] =
arguments.messageArg.data.headers.gatewayid>
```

5. Add a cfreturn to the function so the newly formatted msg structure will be passed on to BlazeDS

```
<cfreturn msg>
```

6. Save your FlexMessaging.cfc file
7. Test your work by accessing index.cfm in a browser: *<your_web_root>/FlexMessaging/view/index.cfm*
 1. Type a message into the textarea and press the Send Message button
 2. Your message should be sent from jQuery to the createMessage function of the FlexMessaging.cfc component
 3. The createMessage function will do two things: Send a gateway message with SendGatewayMessage and return the message structure via the <cfreturn> tag
 4. The SendGatewayMessage() will result in the message going to BlazeDS and displaying in your console or Terminal window you should still have open
 5. The jQuery success function should also execute which will result in the date/time of the message and the message string you typed displaying below the textarea in the HTML page
 6. Notice how the console/Terminal window indicates there are no subscribed clients (no applications are subscribed to the *cfflexmessaging* destination)

Enter a message to send to the Flex app:

Send Message

09-20-2009 07:10 PM -- It always works the first time!

```
[BlazeDS][CFEventGatewayAdapter] Received message from ColdFusion Event Gateway:
Flex Message (flex.messaging.messages.AsyncMessage)
  clientId = FlexMessaging
  correlationId = null
  destination = cfflexmessaging
  messageId = 835A7B35-F93B-B85C-97C0-27C0485EBF9F
  timestamp = 1253491831481
  timeToLive = 0
  body = {MESSAGE=It always works the first time!, SENTAT=09-20-2009 07:10 PM}
[BlazeDS]Sending message: Flex Message (flex.messaging.messages.AsyncMessage)
  clientId = FlexMessaging
  correlationId = null
  destination = cfflexmessaging
  messageId = 835A7B35-F93B-B85C-97C0-27C0485EBF9F
  timestamp = 1253491831481
  timeToLive = 0
  body = {MESSAGE=It always works the first time!, SENTAT=09-20-2009 07:10 PM}
  to subscribed clientIds: []
aaron-west@macbook-pro:bin aaron$
```

Walkthrough 10 - Creating a Message Consumer Flex Application Using Flash Builder

In the first 9 walkthroughs you configured BlazeDS, and then wrote the code that allows messages to be sent in an HTML page of a Web browser to ColdFusion/BlazeDS, and ultimately back to the Web page. For this walkthrough we will return to Flash Builder and create the MXML / ActionScript 3 code needed to consume messages sent by BlazeDS.

1. Return to Eclipse and switch to the Flash perspective
2. Open the FlexMessaging.mxml file located in the src directory of your project. The file should look like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://
ns.adobe.com/flex/halo" minWidth="1024" minHeight="768">

</s:Application>
```

3. Edit the *minWidth* and *minHeight* values
 1. *minWidth* = 300
 2. *minHeight* = 400
4. Add three new attributes, *creationComplete*, *fontFamily*, and *color* to the Application tag such that the code now looks like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="library://
ns.adobe.com/flex/halo" minWidth="300" minHeight="400"
creationComplete="initApp()" fontFamily="Verdana"
color="#000000">

</s:Application>
```

5. In between the opening and closing Application tag create a new Script block

```
<fx:Script>
    <![CDATA[

    ]]>
</fx:Script>
```

6. Add the import statements needed to create a connection to the BlazeDS server

```
import mx.messaging.channels.AMFChannel;
import mx.messaging.ChannelSet;
import mx.messaging.messages.IMessage;
```

7. Add the import statement for *ArrayCollection*. The messages received by the Flex application will be stored in an *ArrayCollection*.

```
import mx.collections.ArrayCollection;
```

8. Create the ArrayCollection that will hold all messages sent from BlazeDS. Since the ArrayCollection will be bound to interface elements through databinding you need to add the [Bindable] metadata instruction.

```
[Bindable]
private var messageCollection:ArrayCollection = new
ArrayCollection();
```

9. Flex applications communicate to BlazeDS through channels and ChannelSets. Create an instance of the ChannelSet class as well as a new AMFChannel. The channel endpoint URI you use should be in the form:
[http://127.0.0.1:\[port_number\]/\[coldfusion_instance_name\]/flex2gateway/\[channel_name\]](http://127.0.0.1:[port_number]/[coldfusion_instance_name]/flex2gateway/[channel_name])

```
private var cs:ChannelSet = new ChannelSet();
private var amfChannel:AMFChannel = new AMFChannel("cf-
longpolling-amf", "http://127.0.0.1:8301/centaur/flex2gateway/
cfamflongpolling");
```

10. The Application tag of the application (line 1) defines a creationComplete event that will execute a function called initApp(). Create this function by typing the following code.

```
// Initialize the Flex application after everything is loaded up
and ready.
private function initApp():void {
    messageCollection.addItem("...Now listening for
messages...");
    amfChannel.enableSmallMessages = false;
    amfChannel.pollingEnabled = true;
    amfChannel.pollingInterval = 3000;
    cs.addChannel(amfChannel);
    consumer.channelSet = cs;
    consumer.subscribe();
}
```

11. In order for your Flex application to talk to BlazeDS and consume messages you need to create an instance of the Consumer class. Type the following code after the

closing `</fx:Script>` tag which creates a Consumer called “consumer” associated with the *cfflexmessaging* destination (defined in *messaging-config.xml*).

```
<fx:Declarations>
    <mx:Consumer id="consumer" destination="cfflexmessaging"
message="messageHandler(event.message)"/>
</fx:Declarations>
```

12. Your Flex application needs a place to display all the messages coming from BlazeDS so create a simple List component that gets its data from the *messageCollection* Array we created earlier.

```
<s:List id="messageList" dataProvider="{messageCollection}"
width="400" height="300" x="0" y="0"/>
```

13. Finally, go back to your `<fx:Script>` block and add a new function that will serve as the event handler for messages. The `<mx:Consumer/>` code you wrote was associated with this event handler which will receive all incoming BlazeDS messages. Type the following code to create the *messageHandler()* function.

```
// Event handler for messages received. Push the new message
onto the ArrayCollection.
private function messageHandler(message:IMessage):void {
    messageCollection.addItem(message.body.SENTAT + " -- " +
message.body.MESSAGE);
}
```

14. Save your *FlexMessaging.mxml* file
15. Run the Flex application in a browser by pressing the green button with a white right-facing triangle. The button looks like a play button.
16. If the Run As dialog box opens, choose Web Application and press Ok
17. Type a message into the textarea and check for the message in your Flex application.
 1. If everything works, congratulations!
 2. If for some reason the message doesn't show up in the Flex app or you get errors, try restarting your browser and then run the application again.
 3. If it still doesn't work raise your hand and a teaching assistant will help you.

(Optional) Walkthrough 11 - Adding the Flex Application to the HTML Page

In this optional walkthrough you will add additional files to your project and edit index.cfm so the Flex portion of the applications runs beneath the HTML textarea. This will allow you to test and demo your application in one browser window.

1. Expand the html-template folder in your FlexMessaging project
2. Copy the generated swfobject.js file in the html-template folder and paste it into /view/scripts/
3. Open index.cfm if it isn't open already
4. Add the following script include after the include for FlexMessaging.js

```
<script type="text/javascript" src="scripts/swfobject.js"></script>
```

5. Add the following script block just after the swfobject.js include and before the opening <body> tag. To speed things up I suggest copying these lines from the /walkthroughs/final_app/view/index.cfm file.

```
<script type="text/javascript">
    <!-- For version detection, set to min. required Flash
    Player version, or 0 (or 0.0.0), for no version detection. -->
    var swfVersionStr = "10.0.0";
    <!-- To use express install, set to
    playerProductInstall.swf, otherwise the empty string. -->
    var xiSwfUrlStr = "playerProductInstall.swf";
    var flashvars = {};
    var params = {};
    params.quality = "high";
    params.bgcolor = "#ffffff";
    params.allowscriptaccess = "sameDomain";
    params.allowfullscreen = "true";
    var attributes = {};
    attributes.id = "FlexMessaging";
    attributes.name = "FlexMessaging";
    attributes.align = "middle";
    swfobject.embedSWF(
        "../bin-debug/FlexMessaging.swf", "flashContent",
        "100%", "100%",
```

```

        swfVersionStr, xiSwfUrlStr,
        flashvars, params, attributes);
    <!-- JavaScript enabled so display the flashContent div in
case it is not replaced with a swf object. -->
    swfobject.createCSS("#flashContent", "display:block;text-
align:left;");
</script>

```

6. Add the following code just after the <div id="messageList"></div> line and before the closing </body> tag. To speed things up I suggest copying these lines from the /walkthroughs/final_app/view/index.cfm file.

```

<hr />
<p />
<!-- This is the Flash content holder. --->
<div id="flashContent">
    <p>
        To view this page ensure that Adobe Flash Player version
        10.0.0 or greater is installed.
    </p>
    <a href="http://www.adobe.com/go/getflashplayer">
    
    </a>
</div>

```

7. Save index.cfm
8. Return to your browser and refresh your index.cfm file. You should now see your FlexMessaging application running beneath the HTML textarea.
9. Type a message into the textarea and watch for the message to appear in your Flex application.
10. If everything works, congratulations!