by Tom Muck

Upgrading the Web

If you are a Web programmer who works with ColdFusion, JSP, or ASP.NET and has said to himself "Flash isn't for me", then this article is for you.

I've been involved with Web programming for a few years and have always worked with HTML in conjunction with a server-side technology such as ColdFusion, ASP, or PHP. Dreamweaver MX is an application development environment that makes it easy for the average Web page designer to break into the world of application development. Using Dreamweaver MX, you can create HTML pages that co-exist with server-side functionality on the page in the guise of server-side tags or scriptlets. These hybrid pages (called *templates*) are the basis of much of the application development that takes place on the Web today. A typical user experience is like this:

The user comes to your site. He enters something into a search field and presses submit. After hitting the submit button, the user waits for a response from the server. The user's browser loads in the page and then the user is able to view the search results. If the results span over more than one page, there might be a link to the next set of results. If the user clicks the link, he sends the search back to the server, which conducts the search again and this time sends back the second page of results. Again, a new page is loaded into the browser.

This is client/server communication at its most primitive, yet it is the accepted standard on the Web. You can liken this to a phone conversation. Imagine that you are conversing with a friend on the phone. You ask a question and hang up the phone. Your friend calls you back and answers the question, then hangs up. You call him back and add something to the conversation and hang up again. This is similar in concept to how the client/server communication with a Web browser operates. Two plastic cups with a taught string between them gives you a more advanced communication method than that.

I have a confession to make. I have never been a big fan of HTML. The language is archaic and limited. It has not really advanced in the last 10 years. Computers, in that same time period, have increased in speed 500 times. Application development for desktop applications has gotten to the point where you can build a program to do whatever your imagination dreams up. We use HTML to build Web applications because that is the universally accepted Web format. With the introduction of Flash MX, the Web has been upgraded.

What makes Flash MX unique – and infinitely superior to previous versions of Flash – is the introduction of UI Components in conjunction with the new Flash Remoting technology. UI Components are fully functional interface elements that are completely configurable and skinable. In previous versions of Flash, this sort of functionality had to be created for each new application. The addition of components allows the developer to concentrate on the functionality of the Web application rather than worry about the mundane aspects of how the buttons and form elements work. This is similar in concept to how Delphi and Visual Basic work in the arena of application development.

The Flash Remoting technology is the equivalent of true client/server communication on the Web. Gone is the click/wait/reload approach of HTML. With Flash MX, you can build your Web application as a unit. The communication with the server is done by Flash behind the scenes. When you build a Web site with Flash MX using Flash Remoting, the user experience is similar to what you would expect from a desktop application. I'm not advocating replacing all HTML pages with Flash movies, but I am suggesting that any client/server communication should be performed with Flash rather than using the methods that you may have become accustomed to using HTML in conjunction with scriptlets.

Flash Remoting is a technology that resides on the application server. It is included with ColdFusion MX and JRun 4, and offered as an add-on for WebSphere (and eventually other J2EE servers) and also ASP.NET. The WebSphere and ASP.NET components were in beta at the time of this writing, but have been well documented on the Macromedia Web site.

Older versions of Flash were able to communicate with a server in a limited fashion with name/value pairs or with XML. Flash Remoting offers many advantages over these methods. Because the server component can communicate directly with the Web page through the Flash movie, more complex objects can be transferred, such as structures and recordsets. Also, because the server contains functionality that can be accessed directly, the Flash code can be more concise. Recordsets can be loaded into Flash movies using a new Recordset object of Flash, making it easy to sort and page through results directly from the client browser. Also, DataGlue is a new set of functions that allows you to bind Flash user interface elements to the data. Flash Remoting also offers debugging of client and server-side ActionScript code using the NetConnect debugger. With all of these new features, Web programming has finally come of age.

You need the following for Flash Remoting:

Figure 1: Flash UI Components

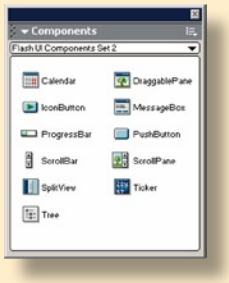

Figure 2: The second set of UI Components for Flash MX

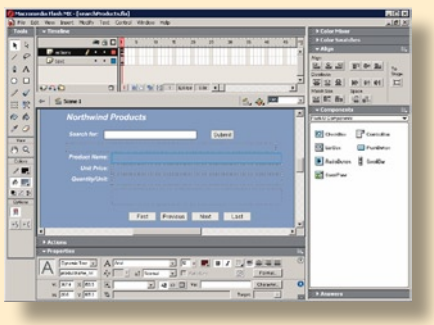

Figure 3: The completed Flash interface

Figure 4: The Property Inspector can be used to set Click Handlers for buttons

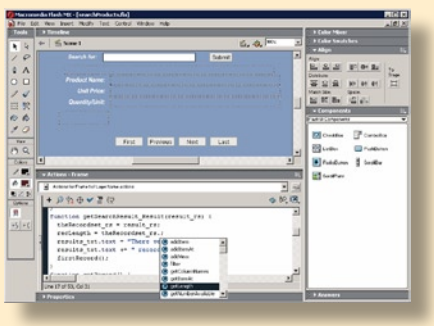

Figure 5: Code hints pop up when you use the naming convention of Flash (such as _rs for a recordset)

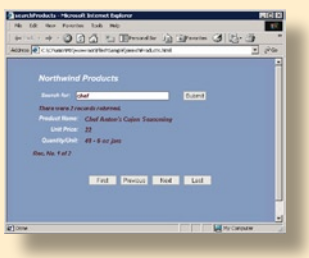

Figure 6: Searching the Northwind database from the Flash movie

- A tool to build your Web pages (Dreamweaver MX is ideal).
- Flash MX
- Flash Remoting components (free download located at http://www.macromedia.com/software/flash/flashremoting/)
- A suitable application server (ColdFusion MX, JRun 4, IIS with ASP.NET, WebSphere)
- Optionally, Flash UI Components Set 2 (shown in Figure 2) and Flash Charting Components (two sets of components offering such things as calendar controls and graphs). These are available from the Macromedia Exchange (www.macromedia.com/exchange)

With these items in place, you are ready to build your first client/server application using Flash MX. We'll build a simple search/results application using the Northwind* database that is supplied with MS Access and MS SQL Server. There are three things we'll have to do: create an interface (in Flash - not in HTML); create a recordset (using ColdFusion, ASP.NET, or JSP); display the results of the recordset in the Flash interface. I am not a designer, so the interface is very simple and the Flash movie will never leave the first frame. You've all seen Flash graphics before - you don't need me to tell you that Flash is capable of outstanding graphical displays. We aren't interested in graphics at this point - only the client/server functionality of Flash Remoting.

The interface, shown in Figure 3, contains the following items:

- Input Text element named *search_txt*.
- Submit button (a Push Button from the UI Components) named *submit_btn*
- Dynamic text element named *results_txt*
- More dynamic text elements named *ProductName_txt*, *UnitPrice_txt*, and *QuantityPerUnit_txt*. These items should have corresponding labels as well.
- Buttons to allow paging through the recordset using labels that read First, Previous, Next, and Last. These buttons should also be Push Button UI Components.

*If you don't have the Northwind database, any database will work for the demo by substituting the appropriate field names.

With the interface out of the way, we can build the server-side functionality. I'll do it in ColdFusion, since that is what I know best. ColdFusion MX has a new feature called a ColdFusion Component (CFC) that really works well with Flash Remoting. We'll create a simple CFC named searchProducts that will query the database for all products where the product name is like the value that a user will type into the search field (search_txt from the Flash movie).

A CFC is created by using a <cfcomponent> tag in a special ColdFusion file that has a .cfc file extension. Within this <cfcomponent> tag is a <cffunction> tag. The <cffunction> tag, for all practical purposes, works in a similar fashion to a function in any other computer language - you call it by name, supply the necessary parameters, and handle the result. Our CFC looks like this:

```
<cfcomponent displayName="searchProducts">
   <cffunction name="getSearchResult" access="remote"
    returnType="query">
      <cfargument name="search_txt" type="string"
       default="%">
      <cfquery name="rsGetProducts" datasource="Northwind">
       SELECT ProductName, UnitPrice, QuantityPerUnit
       FROM Products
       WHERE ProductName LIKE '%#search_txt#%'
      </cfquery>
      <cfreturn rsGetProducts>
   </cffunction>
</cfcomponent>
```

Let's take a look at this component line by line. The ColdFusion Component (which will be named searchProducts.cfc) contains only one function (also called a *method*): getSearchResult:

```
<cffunction name="getSearchResult"
access="remote"
 returnType="query">
```

The function specifies a return type of *query*. This is exactly what we are going to return to the Flash movie - the results of a query. The function has within it a <cfargument> tag. This specifies the argument that will be supplied to the CFC. In this case, the argument is the search_txt field from the Flash interface. We'll also give it a default value of "%" so that the query to the database will return all results if no value is supplied (the % is the wildcard character in SQL):

```
<cfargument name="search_txt" type="string"
default="%">
```

Next is the actual query to the database, which utilizes the ColdFusion tag <cfquery>:

```
<cfquery name="rsGetProducts"
datasource="Northwind">
 SELECT ProductName, UnitPrice, QuantityPerUnit
 FROM Products
 WHERE ProductName LIKE '%#search_txt#%'
</cfquery>
```

Finally, the <cfreturn> tag returns a result to the caller. In this case the caller is our Flash movie, and the result is the entire query that was just executed. We return the query by name here:

```
<cfreturn rsGetProducts>
```

Now we move back to the Flash movie to add the script for Flash Remoting. To utilize Flash Remoting services, you have to follow these general steps in your Flash movie:

1. Include the Flash Remoting classes located in NetServices.as in your movie. If you are going to utilize DataGlue, include the DataGlue.as classes as well.
2. Set the URL where your Flash Remoting gateway lives.
3. Make a connection to gateway.
4. Create the service object.
5. Define any event handlers you might need.
6. Call any service functions that you are utilizing.

This is going to require some hand-coded ActionScript in the Flash movie. If you are unfamiliar with ActionScript, you'll find that it is almost identical to JavaScript and very easy to learn. There are several very good books out there for learning ActionScript, including *Macromedia Flash MX ActionScripting: Advanced Training from the Source*, by Derek Franklin and Jobe Makar which offers a good tutorial approach by example.

It's a good idea to put ActionScript in its own layer in your movie. We've added a layer to the movie timeline called *actions* and have added this script in the Actions panel of Flash:

```
#include "NetServices.as"
if (connected == null) {
  connected = true;
  NetServices.setDefaultGatewayUrl("http://
127.0.0.1/flashservices/gateway");
  //my_conn uses the _conn naming convention to
enable Flash coding hints
  var my_conn = NetServices.createGatewayConnect
ion();
  var myService = my_conn.getService("flashsample.
searchProducts",this);
  var theRecordset_rs = null;
  var recNum = 0;
  var recLength = 0;
}
function onSubmit () {
  myService.getSearchResult(search);
}
function getSearchResult_Result(result_rs) {
  theRecordset_rs = result_rs;
  recLength = theRecordset_rs.getLength();
  results_txt.text = "There were " + recLength;
  results_txt.text += " records returned.";
  firstRecord();
}
function getRecord() {
  if(recLength == 0) {
    ProductName_txt.text = "";
    UnitPrice_txt.text = "";
    QuantityPerUnit_txt.text = "";
    recordnumber_txt.text = "No Records";
  }else{
    ProductName_txt.text =
theRecordset_rs.items[recNum].ProductName;
    UnitPrice_txt.text =
theRecordset_rs.items[recNum].UnitPrice;
    QuantityPerUnit_txt.text =
theRecordset_rs.items[recNum].QuantityPerUnit;
    recordnumber_txt.text =
"Rec. No. " + (recNum + 1) + " of " + recLength;
  }
}
function firstRecord() {
  recNum = 0;
  getRecord();
}
function previousRecord() {
  recNum--;
  if(recNum < 0) recNum = 0;
  getRecord();
}
function nextRecord() {
  recNum++;
  if(recNum >= recLength) recNum = recLength - 1;
  getRecord();
}
function lastRecord() {
  recNum = recLength - 1;
  getRecord();
}
```

That's a lot of script to digest at one time, so we'll go through it a bit at a time. The first thing that you need to do is to include the NetServices.as script. This allows you to create Flash Remoting functionality in your Flash movie:

```
#include "NetServices.as"
```

The file is located in the \Configuration\Include directory in your Flash MX program directory, so you don't need to include a path. Flash will know where to look for it. If you are debugging, you can include the NetDebug.as file as well.

Next, you have to create a connection to the Flash Remoting services. Since this only has to be done once, we check

for a global variable flag that we create, named *connected*. If it doesn't exist, we know that this is the first time our script has been executed. The first time through, we also set this variable so that next time we won't execute this section of code:

```
if (connected == null) {
  connected = true;
```

Now, the connection. First you have to set the path to the gateway:

NetServices.setDefaultGatewayUrl("http://127.0.0.1/flashservices/gateway");

This will be a path to the Web server (http://127.0.0.1 is our local machine) and a reference to the Flash Remoting gateway, which is not a physical path. The CF MX server knows that this is a reference to the gateway.

Next, set the connection:

```
  var my_conn = NetServices.createGatewayConnect
ion();
```

Now, create a service object, which is essentially a reference to the CFC that you created. In this line, we have a folder named *flashsample* at the root of our Web site with a file named searchProducts.cfc:

```
 var myService = my_conn.getService("flashsample.s
earchProducts",this);
```

Next, we initialize some global variables that we'll need for the recordset paging. In actual practice we would probably not use global variables, but for this simple example we are more concerned with the functionality of the client/server interaction:

```
  var theRecordset_rs = null;
  var recNum = 0;
  var recLength = 0;
}
```

Next, the submit button click handler function, named onSubmit. The submit button will call the service named getSearchResult (our function from the CFC) using the contents of the textfield named search_txt:

```
function onSubmit () {
  myService.getSearchResult(search_txt.text);
}
```

The submit button doesn't know about this function yet, but there is an easy way to add callback functions to the new UI Components: the Property Inspector. Figure 4 shows the onSubmit button being given a Click Handler. After the user clicks submit, the contents of the text field are sent to the CFC and the results are returned to the Flash movie.

Next, we'll handle the results from the database search. This is done with a function that is named using a naming convention of Flash Remoting. Results are always returned to a function named using the function name of the remote method along with a suffix of *_Result*. If you recall, the CFC will return an entire recordset to the caller, so this function will receive that result, which we've called *result_rs*. By giving the recordset a name with a _rs suffix, we can take advantage of Flash's built-in code hints and code completion when coding our ActionScript (see Figure 5):

```
function getSearchResult_Result(result_rs) {
  theRecordset_rs = result_rs;
```

The recordset object has a built-in method to allow you to find out how many records are in it: getLength(). We'll set one of the global variables that we've already defined to the length of the recordset and also display that in the results_txt text element in the Flash movie, along with some descriptive text:

```
  recLength = theRecordset_rs.getLength();
  results_txt.text = "There were " + recLength;
  results_txt.text += " records returned.";
```

Now we'll call a function called firstRecord() which will display the first record in the resultset. That function will be explained shortly:

```
  firstRecord();
}
```

Before we get to that, however, we need to create a general utility function that will display the current record. This function will be used by each of our recordset navigation buttons:

```
function getRecord() {
  if(recLength == 0) {
    ProductName_txt.text = "";
    UnitPrice_txt.text = "";
    QuantityPerUnit_txt.text = "";
    recordnumber_txt.text = "No Records";
  }else{
    ProductName_txt.text = theRecordset_rs.items[
recNum].ProductName;
    UnitPrice_txt.text = theRecordset_rs.items[re
cNum].UnitPrice;
    QuantityPerUnit_txt.text = theRecordset_rs.it
ems[recNum].QuantityPerUnit;
    recordnumber_txt.text = "Rec. No. " + (recNum
+ 1) + " of " + recLength;
  }
}
```

Here we are simply setting the text elements in our Flash movie to the corresponding recordset field, or setting them to blank if there are no records. The recordset row is referenced using the *items* array of the recordset. The items array is indexed by number. We can pinpoint which record (row) we want to look at by using the recNum global variable that we set up previously. Then we can reference each field by name (as in the expression *theRecordset_rs.items[recNum].ProductName*).

All that is left is to set the click handlers for each of our recordset navigation buttons: First, Previous, Next, and Last. We can set these the same way as we set the submit button: using the Property Inspector. Each function corresponds to a button. Each function sets the record number and then calls our utility function *getRecord()*:

```
function firstRecord() {
  recNum = 0;
  getRecord();
}
function previousRecord() {
  recNum--;
  if(recNum < 0) recNum = 0;
  getRecord();
}
function nextRecord() {
  recNum++;
  if(recNum >= recLength) recNum = recLength - 1;
```

```
  getRecord();
}
function lastRecord() {
  recNum = recLength - 1;
  getRecord();
}
```

You can test this movie from the Flash environment by clicking Control > Test Movie, or you can publish it to your site and browse the resulting HTML page with a browser. As you can see, the HTML page is acting as a home for the Flash movie, but we are delivering server-side functionality to the page. When you browse to the page, you can search the database once, or search it again and again – the browser doesn't need to reload the page. The communication with the server is done by Flash behind the scenes. Figure 6 shows the interface in use.

This was a very simple example, but it should show you the techniques that you can use with Flash Remoting to deliver database content to a browser seamlessly. We've only scratched the surface and haven't even touched on things like server-side ActionScript, DataGlue, and Web Services. Flash MX truly revolutionizes the Web with these technologies. Try it: you might like it.

For more information on Flash Remoting, go to www.macromedia.com/software/flash/flashremoting/ or go to the Designer & Developer center at http://www.macromedia.com/desdev/mx/flash/. For 30-day trial versions of Dreamweaver MX, ColdFusion MX, and Flash MX, go to www.macromedia.com/software/trial_download/.