

Rich Internet Applications

The development of Rich Internet Applications is now underway. Some people are calling this “Web 2.0”, but it is really the transition from a page based browsing experience to one that more closely resembles desktop applications. A variety of technologies can be used to deliver this experience... AJAX is currently one of the more popular sets of technologies, mainly because it can be easily adapted into existing web based applications. However, for those who are looking for something more robust, there appears to be 2 early front runners: Flex based applications that run in the Flash player from Adobe, and XAML based applications from Microsoft. In this article, I’ll introduce you to Adobe’s Flex product line, including Adobe Flex Builder and Adobe Flex Enterprise Services.

Intro to Adobe Flex

Flex applications are Rich Internet Applications that are built using tools from Adobe. These Flex applications run inside the Flash player, leveraging the existing install base of the Flash player. (Note: Applications built using Flex Builder 2.0 require Flash player 8.5, currently available in beta format only).

Flex is made up largely of two parts. The first is an IDE built on the open source Eclipse platform. That tool is called Flex Builder 2.0, and is available for free download now (in beta format) from the Adobe labs website at <http://labs.adobe.com>. Final pricing has yet to be announced, but Adobe has committed to making Flex Builder 2.0 available for less than \$1,000. As well, a free SDK will be made available for developers who want to build Flex applications. The SDK will include the Flex compiler, allowing developers who code by hand (without the IDE) the ability to create Flex applications for free. For those of us who need a visual IDE, we will still need to purchase the Flex Builder 2.0 tool.

The other part of Adobe Flex is Adobe Flex Enterprise Services. This tool is largely targeted at large enterprises, but will also be made available free to individual developers for limited use. The limitation on the free version of Flex Enterprise Services is that it will be limited in the number of concurrent connections it allows, and in the number of servers it can be installed on.

Integrating Flex and PHP

OK, now that we have a good idea of what Adobe Flex is, let’s see how we can build something. The first step is to download Flex Builder 2.0 from the Adobe labs site at <http://labs.adobe.com>. If you are familiar with Eclipse, and have it installed, you can install the plugin version of Flex Builder. If not, then you’ll want to install the standalone version.

Once installed, we’ll want to create a new Flex application. In this tutorial, we’re going to keep the application simple. We will create an application that will read names and email addresses from a database and display them to the user. Our simple application will also allow users to add usernames and email addresses to the database.

To start, create the database that we need. I’ve called mine “sample”, but you can call yours whatever you like. Next, create the table that will hold the user data. Here is the SQL structure for our table:

```
CREATE TABLE `users` (  
  `userid` int(10) unsigned NOT NULL auto_increment,
```

```

`username` varchar(255) collate latin1_general_ci NOT NULL,
`emailaddress` varchar(255) collate latin1_general_ci NOT NULL,
PRIMARY KEY (`userid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci
AUTO_INCREMENT=3 ;

```

OK, now that we've got our table, let's create the PHP script that will add users and export the XML that the Flex application will consume. Its relatively simple, only 25 lines of code or so (note that the quote_smart function is a best practice to help verify user input, according to the PHP.Net website at <http://www.php.net/manual/en/function.mysql-real-escape-string.php>)

```

<?php
Define( "DATABASE_SERVER", "localhost" );
Define( "DATABASE_USERNAME", "username" );
Define( "DATABASE_PASSWORD", "password" );
Define( "DATABASE_NAME", "sample" );

//connect to the database
$mysql = mysql_connect(DATABASE_SERVER, DATABASE_USERNAME,
DATABASE_PASSWORD);

mysql_select_db( DATABASE_NAME );

// Quote variable to make safe
function quote_smart(T$value)
{
    // Stripslashes
    if (get_magic_quotes_gpc()) {
        $value = stripslashes($value);
    }
    // Quote if not integer
    if (!is_numeric($value)) {
        $value = "'" . mysql_real_escape_string($value) . "'";
    }
    return $value;
}

if( $_POST["emailaddress"] AND $_POST["username"])
{
    //add the user
    $Query = sprintf("INSERT INTO users VALUES (', '%s', '%s')",
quote_smart($_POST["username"], quote_smart($_POST["emailaddress"]));

    $Result = mysql_query( $Query );
}

//return a list of all the users
$Query = "SELECT * from users";
$Result = mysql_query( $Query );

```

```

$return = "<users>";

while ( $User = mysql_fetch_object( $Result ) )
{
    $Return .= "<user><userid>".$User->userid."</userid><username>".$User-
>username."</username><emailaddress>".$User-
>emailaddress."</emailaddress></user>";
}
$return .= "</users>";
mysql_free_result( $Result );
print ($Return)
?>

```

The PHP should be fairly self explanatory. The \$_POST variable is populated with values from our Flex application, with two fields required: emailaddress and username. If both of those are filled out, then it adds the user to the database. After that, we return a list of users in XML format. Note: You cannot pass PHP variables to Flex applications directly, they must be encoded in XML first. By abstracting the user interface from the data retrieval, it allows you to easily change how the data is displayed. For example, you could use this same PHP script to pass data to a mobile phone version of the same application. All you would need for that is to write the front end of the application, the backend PHP script would remain the same.

Up until now, everything should seem fairly familiar. We've got a PHP script and a MySQL database. Now its time to start building the interface to our application.

Flex applications are built using a combination of ActionScript (AS) 3.0 and MXML. ActionScript is based on ECMA Script (similar to JavaScript), so it should be familiar to web developers. MXML is an XML based layout engine for Flex applications. Essentially, you layout the UI using XML, and script the UI using ActionScript. The MXML for our interface is, again, very simple (only 26 lines!).

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.macromedia.com/2005/mxml" xmlns="*"
layout="absolute" creationComplete="userRequest.send(">
    <mx:HTTPService id="userRequest" url="http://localhost/flex/php/request.php"
useProxy="false" method="POST">
        <mx:request xmlns="">

            <username>{username.text}</username><emailaddress>{emailaddress.text}</e
mailaddress>

        </mx:request>
    </mx:HTTPService>
    <mx:Form x="22" y="10" width="493">
        <mx:HBox>

```

```

        <mx:Label text="Username"/>
        <mx:TextInput id="username"/>
    </mx:HBox>
    <mx:HBox>
        <mx:Label text="Email Address"/>
        <mx:TextInput id="emailaddress"/>
    </mx:HBox>
    <mx:Button label="Submit" click="userRequest.send()"/>
</mx:Form>
<mx:DataGrid id="dgUserRequest" x="22" y="128"
dataProvider="{userRequest.result.users.user}">
    <mx:columns>
        <mx:DataGridColumn headerText="User ID"
columnName="userid"/>
        <mx:DataGridColumn headerText="User Name"
columnName="username"/>
    </mx:columns>
</mx>DataGrid>
    <mx:TextInput x="22" y="292" id="selectedemailaddress"
text="{dgUserRequest.selectedItem.emailaddress}"/>
</mx:Application>

```

Let's examine each line in detail:

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.macromedia.com/2005/mxml" xmlns="*"
layout="absolute" creationComplete="userRequest.send()">

```

These are the first two lines of each Flex application. The first line declares that this is an XML document. The second declares that this is an Application, provides the namespace for MX components, declares the layout to be “absolute” (meaning you can position items to the exact x and y coordinate. Other options are horizontal layouts or vertical layouts), and finally “creationComplete=userRequest.send()” says that on completion of loading the UI, call the function send() on the MXML element with the id userRequest.

```

<mx:HTTPService id="userRequest" url="http://localhost/flex/php/request.php"
useProxy="false" method="POST">
    <mx:request xmlns="">

        <username>{username.text}</username><emailaddress>{emailaddress.text}</e
mailaddress>
    </mx:request>
</mx:HTTPService>

```

This is where we setup the HTTPService, to send and receive data from the PHP script we created. We set the id to userRequest, and provide a URL to our PHP script. We set the method of submit to POST (we could also use GET, but then we'd have to change our variables in the PHP script). The request itself contains two variables, username and emailaddress. The value for username is set to the text attribute of the element with id

“username” (username.text) and the value for the PHP variable `_POST[“emailaddress”]` is set to the text attribute of the element with id “emailaddress” (emailaddress.text). The { and } bind the variables to the value of the UI elements.

Just to be clear, if we changed `<username>` to `<user_name>`, we would have to change our PHP variable to `_POST[“user_name”]`. If we change `{username.text}` to `{user_name.text}`, we would have to modify our MXML: the element with the ID “username” would need to have its ID changed to “user_name”.

Next, we build a simple form:

```
<mx:Form x="22" y="10" width="493">
  <mx:HBox>
    <mx:Label text="Username"/>
    <mx:TextInput id="username"/>
  </mx:HBox>
  <mx:HBox>
    <mx:Label text="Email Address"/>
    <mx:TextInput id="emailaddress"/>
  </mx:HBox>
  <mx:Button label="Submit" click="userRequest.send()"/>
</mx:Form>
```

Notice that we can layout the exact x and y coordinates of the form, and set its exact width. Then, two HBoxes surround a label and textinput, allowing them to flow from left to right, one above the other. Finally, our Submit button appears at the end of our form. When the button is clicked, it calls the `send()` function of the element with ID “userRequest” (in this case, it is our `HTTPService` element).

OK, so we’ve got the area that we submit new entries to the database, but where to we display them? That’s next:

```
<mx:DataGrid id="dgUserRequest" x="22" y="128"
dataProvider="{userRequest.result.users.user}">
  <mx:columns>
    <mx:DataGridColumn headerText="User ID"
columnName="userid"/>
    <mx:DataGridColumn headerText="User Name"
columnName="username"/>
  </mx:columns>
</mx:DataGrid>
<mx:TextInput x="22" y="292" id="selectedemailaddress"
text="{dgUserRequest.selectedItem.emailaddress}"/>
</mx:Application>
```

In this case, we have a `DataGrid`, which populates itself with the XML that we get from the `userRequest HTTPService`. We return an XML document, and in this case we bind the `DataGrid` to the user elements in the XML document that gets returned.

The returning XML looks something like this:

```
<users>
  <user>
    <userid>1</userid>
    <username>Joe Schmoe</username>
    <emailaddress>joe@schmoe.com</emailaddress>
  </user>
  <user>
    <userid>2</userid>
    <username>Betty Schmoe</username>
    <emailaddress>betty@schmoe.com</emailaddress>
  </user>
</users>
```

Notice that we bind to the actual elements that get returned, not to the wrapper element around them.

The DataGrid displays the userid and usernames of people in the database. I decided not to show the emailaddress in the datagrid, but you could add another column with that information in it. Notice that the columnName needs to map directly to the XML elements. The DataGrid element will take care of allowing our users to sort and highlight the rows as they are selected – we don't need to do anything for that!

Finally, we have a TextInput, which shows the emailaddress of the selected user (dgUserRequest.selectedItem.emailaddress), and then an XML tag that closes the application.

So, there you go. A simple Flash based application that submits and retrieves data from a MySQL database, using PHP as a backend. I urge you to download Flex Builder 2.0 and build more complicated applications using PHP, MySQL and Adobe Flex. Check out my blog at <http://blogs.adobe.com/mikepotter/> for more information on Adobe Flex. And please provide suggestions for future articles and what other samples you'd like to see using this set of technologies.

Mike Potter
Adobe Systems Inc.
Web Developer Evangelist