Getting started

If you want to build a medium- to large-scale enterprise application, use Adobe Flex Data Services 2. That will simplify the data interaction by adding high-performance data transfer, message-based publish and subscribe and more. AMFPHP only provides a small subset of what is possible with Flex Data Services. If you're working for a large company, then you'll want to look at Flex Data Services for your data interaction layer.

The key to accomplishing this task is a project called AMFPHP. The project was originally started by Wolfgang Hamann. The team has grown and now has approximately five to six developers. Thanks to their hard work, the Flex community now has a front end to PHP back-end applications based on Flex.

If you haven't done so already, download and install Flex 2 and AMFPHP v. 1.1 (for details see the readme files). Also, now is a good time to read my article, Integrating Flex 2 and PHP, if you haven't done so already.

The sample that you create in this tutorial demonstrates how to use Flex to display records from a database. You will not insert or update any records, you will only display the data to the user. Use a similar database to the first example, or if you have a database and table created from that example, then use that. Otherwise, create the following MySQL 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 ;
```

Populate that database with some rows of data. The field names are self explanatory. Remember, since this example only displays data from the database, if you don't populate it, then you'll have nothing to show. Use PHPMyAdmin if you need something to populate the table.

Below is the new PHP code. Place this code in a file called **sample.php** in the services folder of your AMFPHP installation:

```
<?php
// Create new service for PHP Remoting as Class
class sample
{
    function sample ()
    {
        // Define the methodTable for this class in the constructor
        $this->methodTable = array(
            "getUsers" => array(
                "description" => "Return a list of users",
                "access" => "remote"
            )
        );
    }

    function getUsers () {
        $mysql = mysql_connect(localhost, "username", "password");

        mysql_select_db( "sample" );

        //return a list of all the users
        $Query = "SELECT * from users";
        $Result = mysql_query( $Query );
        while ($row = mysql_fetch_object($Result)) {
                $ArrayOfUsers[] = $row;
            }
            return( $ArrayOfUsers );
        }
}
?>
```

If you're familiar with AMFPHP, then this is likely familiar to you.

The class name needs to match the filename. In our case, the filename is sample.php, so the class is called sample. When the class loads, it runs the function `sample()` (classes always initialize themselves by calling functions that match the class name), which defines the methods available to AMFPHP. In this case, there's only one method: `getUsers`, which as the description says, returns a list of users. In the function itself, you can see that it's relatively simple: connect to a MySQL database, get all the users, and return them in an array of objects.

The application front end

Now, let's check out the front end of the application, sample.mxml, which displays in Flash. Amazingly, it's only about 50 lines of code in total.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
creationComplete="initApplication()">
    <mx:DataGrid dataProvider="{dataProvider}">
        <mx:columns>
            <mx:DataGridColumn headerText="Userid" dataField="userid"/>
            <mx:DataGridColumn headerText="User Name" dataField="username"/>
            <mx:DataGridColumn headerText="User Name" dataField="emailaddress"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Script>
        <![CDATA[
            [Bindable]
            public var dataProvider:Array;

            import flash.net.Responder;

            public var gateway : RemotingConnection;

            public function initApplication()
            {
                gateway = new RemotingConnection(
"http://localhost/flex/php/gateway.php" );
                gateway.call( "sample.getUsers", new Responder(onResult,
onFault));
            }

            public function onResult( result : Array ) : void
            {
            dataProvider = result;
            }


            public function onFault( fault : String ) : void
            {
                trace( fault );
            }
        ]]>
    </mx:Script>
</mx:Application>
```

The first line, `<?xml version="1.0" encoding="utf-8"?>`, defines the file as an XML document. This line appears at the top of all Flex applications.

The second line specifies that this is an application, and to run the function `initApplication()` after Flash has loaded (`creationComplete`):

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  xmlns="*"
creationComplete="initApplication()">
```

The next section is a DataGrid component that displays the list of users:

```
<mx:DataGrid dataProvider="{PHPData}">
<mx:columns>
        <mx:DataGridColumn headerText="Userid" dataField="userid"/>
        <mx:DataGridColumn headerText="User Name" dataField="username"/>
        <mx:DataGridColumn headerText="User Name" dataField="emailaddress"/>
    </mx:columns>
</mx:DataGrid>
```

Unlike the DataGrid component in my article on integrating Flex 2 and PHP, this control displays all three columns from the database. Note that the `columnName` element must match the field name in the MySQL table.

The following section is a set of ActionScript instructions:

```
<mx:Script>
    <![CDATA[
        [Bindable]
            public var PHPData:Array;

        import flash.net.Responder;

        public var gateway : RemotingConnection;

        public function initApplication()
        {
            gateway = new RemotingConnection(
"http://localhost/flex/php/gateway.php" );
            gateway.call( "sample.getUsers", new Responder(onResult, onFault));
        }

        public function onResult( result : Array ) : void
        {
            PHPData = result;
        }


        public function onFault( fault : String ) : void
        {
            trace( fault );
        }
        ]]>
</mx:Script>
```

Here's what this set does. The following lines declare an array, PHPData, that can be bound (`[Bindable]`) to MXML objects:

```
[Bindable]
public var PHPData:Array;
```

In this case, the PHPData array is being bound to the `<mx:DataGrid>` as its dataProvider. This means that the DataGrid component gets its data from this variable; in this case, it is an array of

PHP objects.

The next code snippet is required for AMFPHP to work in Flex:

```
import flash.net.Responder;

public var gateway : RemotingConnection;

public function initApplication() : void
{
    gateway = new RemotingConnection( "http://localhost/flex/php/gateway.php" );
    gateway.call( "sample.getUsers", new Responder(onResult, onFault));
}

public function onResult( result : Array ) : void
{
    PHPData = result;
}


public function onFault( fault : String ) : void
{
    trace( fault );
}
```

First, the code imports the flash.net.Responder package, which you need for Remoting. Actually, Flex includes this package automatically when you declare that you need a new Responser later on. I've included it here just to demonstrate its use.

Next, the code sets a variable, `gateway`, to be a `RemotingConnection` datatype. You already know the `RemotingConnection` datatype—you should have a file in the root of the Flex application called RemotingConnection.as, which has the following code inside it:

```
package
{
    import flash.net.NetConnection;
    import flash.net.ObjectEncoding;

    public class RemotingConnection extends NetConnection
    {
    public function RemotingConnection( sURL:String )
        {
            objectEncoding = ObjectEncoding.AMF0;
            if (sURL) connect( sURL );
        }

        public function AppendToGatewayUrl( s : String ) : void
        {
            //
        }
    }
}
```

Note that the class name is the same as the filename. In this case, you are adding one method to the NetConnection package, `AppendToGatewayUrl`.

Next, you have a function, `initApplication`:

```
public function initApplication()
{
    gateway = new RemotingConnection( "http://localhost/flex/php/gateway.php" );
    gateway.call( "sample.getUsers", new Responder(onResult, onFault));
}
```

Your application runs the `initApplication` function when the Flash application loads. You set the variable gateway to be a new connection to the PHP script gateway.php that comes with AMFPHP. Be sure to set the proper path to that file here. Then, in the next line you call the function `getUsers` on the sample class (sample.getUsers). On response, you run one of two functions: `onResult` if things go well, or `onFault` if you receive an error.

The next lines simply set the variable PHPData to the variable result, which was passed back to ActionScript through AMFPHP:

```
public function onResult( result : Array ) : void

{
    PHPData = result;
}
```

This is the array of PHP objects returned when you ran the MySQL query ($ArrayOfUsers). AMFPHP has translated that PHP array of objects into an ActionScript array of objects for you. Pretty cool, eh?

Now look at the final lines:

```
public function onFault( fault : String ) : void
{
    trace( fault );
}
```

In this case, if an error occurs, you handle it now, likely by displaying a message to the user. In this case, you simply trace the fault variable, which is useful if you're running the Flex application in debug mode.

Things are still a bit complicated, so let's be clear about the location for the files. There are three files that you've created and a bunch of files that you received when downloading and deploying AMFPHP.

- Place the Flex project files, sample.mxml and RemotingConnection.as, in the same directory.
- Place the PHP files—such as the AMFPHP directories of actions, adapters, app, browser and so forth—in the services directory.
- Put the PHP files from AMFPHP into your root web directory.
- Put the sample.php into the services directory of the AMFPHP project.

So, you've coded a very simple application that shows how to connect a PHP back end to the front end, built in Flex. Contact me and let me know what you come up with!