Adobe Flex - AMFPHP Mini Cookbook

Here is a little thing that I made up for people using Flex/Flash and amfphp a little cheat sheet to use as a reference.

This has been completely re-done since it has been so popular. I have commented all of the code very well so that does most of the explaining of what code does what. I also added a few more calls such as sending email, and sending images.

For use without any framework and without a services-config.xml file in the compiler.

These examples are snippets from my Flash Remoting presentation and Snippr Air application. There will be a PDF version of this for easy printing.

So Please print and share with everyone who would find any use in this.

AMFPHP Recipes

Recipe 1.1 - AMFPHP Installation

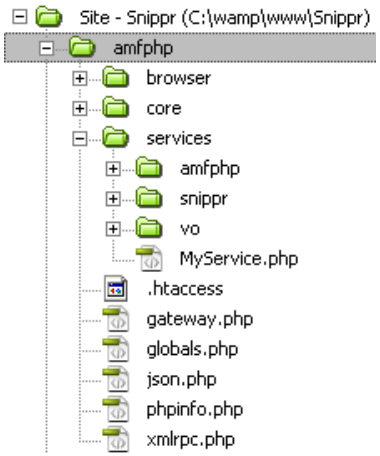
Problem:

You want to start using amfphp, but you do not know where to start.

Solution:

Download amfphp1.9 from http://amfphp.org, unzip the folder, then drop it into your web root.

Server folder structure:



Recipe 1.2 - Connecting to MySQL

Problem:

I have amfphp installed, now how they heck to I connect to my database?

Solution:

Create a simple class file for connecting to MySQL, and include it in any of our classes that need database interaction.

Code:

For connecting to a database make your connection inside the constructor or you could include it in a separate file.

Here is both ways of doing this.

External File	Inside Constructor

```php
<?php

class Database
{

   public function Database(){}

   public function connect()
   {
      define("db_host", "localhost");
      define("db_user", "username");
      define("db_pass", "password");
      define("db_name", "snippr");

      $dbc = mysql_connect( db_host, db_user, db_pass );
            mysql_select_db( db_name );

      return $dbc;
   }

}

?>
```

Inside the separate file you return a variable that you will use as a link to connect to the database, very handy for connecting to one database in multiple files.

```php
<?php

class SnipprService
{
   //Specify our table name
   private $table = "snippr";

   public function SnipprService()
   {
      mysql_connect( "localhost", "username", "password" );
      mysql_select_db( "snippr" ) ;
   }
?>
```

If you were to only have one service class for making calls, then it is very reasonable to include your connection inside your constructor.

Recipe 1.3 - Using AMFPHP with PHP Classes

Problem:

You want to connect Flex to PHP code via AMFPHP, you need to write your PHP code as a PHP class. The remote methods that your ActionScript code calls are the methods of your custom classes.

Solution:

Creating a class both in PHP and ActionScript.

A PHP class uses the following syntax:

```php
<?php

class MyClass
{

  var local1 = 1;
  var local2 = 2;

  public function MyClass( )
  {
    // do something
  }

```

```
14:    public function someMethod( $arg )
15:    {
16:      // do something else
17:    }
18:
19:  ?>
```

An ActionScript class uses this following syntax:

```
1:  package cookbook.amfphp
2:  {
3:     class MyClass
4:     {
5:        public var someVariable:Number;
6:        public var anotherVariable:Number;
7:
8:        public function MyClass()
9:        {
10:          // do something
11:       }
12:
13:       public function someFunction( arg:Object )
14:       {
15:          // do something else
16:       }
17:    }
18: }
```

Recipe 1.4 - Class Mapping

Problem:

You want to specify the data that is being returned from AMFPHP, but do not know how to go about doing so.

Solution:

Class mapping allows you to use a remote PHP object to be mapped directly to your ActionScript object, without having to build generic code to iterate through all of the results coming back. This is very useful when coding in Flex Builder, for one you have code hinting that will help you if you don't remember the exact name of some of the objects and such, and for two it enforces the type that you have defined in your value object in ActionScript is what you are using it for. If that makes any sense.

If you specify that snippet_id inside your value object is a ByteArray.

ex. public var snippet_id:ByteArray;

Then when you go to create a new snippet to send to AMFPHP you cannot set snippet_id to a

String or Number.

ex. var snippet:SnippetVO = new SnippetVO();
snippet.snippet_id = txt_name.text;

Code:

SnippetVO.php

Notice the similarities in the two

```php
<?php

class SnippetVO
{
   var $_explicitType="com.jonniespratley.snippr.vo.SnippetVO";

   var $snippet_id;
   var $snippet_title;
   var $snippet_code;
   var $snippet_type;
   var $snippet_created;
   var $snippet_user;

   public function SnippetVO(){}

   //Now map the database tables to the variables
   public function mapObject( $snip )
   {
      $this->snippet_id       = $snip[snippet_id];
      $this->snippet_title    = $snip[snippet_title];
      $this->snippet_code     = $snip[snippet_code];
      $this->snippet_type     = $snip[snippet_type];
      $this->snippet_created    = $snip[snippet_created];
      $this->snippet_user     = $snip[snippet_user];
   }
}
?>
```

SnippetVO.as

```actionscript
package com.jonniespratley.snippr.vo
{
   [RemoteClass(alias="vo.SnippetVO")]

   /**
    * VOs are used to create a layer of business objects that can be
    * transferred between tiers, instead of using records, results sets, and datasets.
    */
   [Bindable]
   public class SnippetVO
   {
      public var snippet_id:int;
      public var snippet_title:String;
      public var snippet_code:String;
      public var snippet_type:String;
      public var snippet_created:String;
      public var snippet_user:String;

      /**
       * Helper function for building the data.
       * @param source
       *
       */
      public function SnippetVO( source:Object = null )
      {
         if ( source != null )
         {
            for ( var item:String in source )
            {
               try
               {
                  this[item] = source[item];
               }
               catch ( error:Error )
               {
                  throw new Error( "SnippetVO: " + error );
               }
            }
         }
      }
   }
}
```

On this value object we have a mapObject function that maps our specified variables to the names of the table columns, so we will know exactly what is coming back and we can call those fields by the there variable. Our explicitType is the location of our client side value object. Because when Flex sends the vo its "id" is on the client, so we are letting php know that this explicitType is what we are excepting.	For the ActionScript value object we have the same variables that we have inside of our php object, but in our constructor we have a function that loops through every item so later on when we pass this array to our array collection in our model, we will use a special little function for parsing up that data. Our remote alias is set the the location of the server side value object.

Recipe 1.5 - Building CRUD for AMFPHP

Problem:

You want to create, read, update and delete records in your MySQL database from AMFPHP.

Solution:

Creating a CRUD service class for AMFPHP.

Code:

All of the code is commented, and this should help you solve your problem and have full CRUD of your database.

```
 1: <?php
 2:
 3: //If you use your database info inside an external file
 4: //require_once 'Database.php';
 5: class SnipprService
 6: {
 7:    //Database variable
 8:    //var $dbc;
 9:
10:    //Specify our table name
11:    private $table = "snippets";
12:
13:    //Connect to your database, this is for use with credential inside the constructor,
14:    public function SnipprService()
15:    {
16:       mysql_connect("localhost", "spratley_guest", "guest");
17:       mysql_select_db("spratley_snippr") ;
18:
19:       //This is how you connect using an external file with your database info
20:       //$this->dbc = Database::connect();
21:    }
22:
23:    /* Maps the recordset that is pulled from mysql to the value object */
24:    private function mapRecordSet( $recordset )
25:    {
26:       require_once( "../vo/SnippetVO.php" );
27:       $list = array();
28:
29:       while( $data = mysql_fetch_array( $recordset ) )
30:       {
```

```
31:           $vo = new SnippetVO();
32:           $vo->mapObject( $data );
33:           array_push( $list, $vo );
34:        }
35:        return $list;
36:     }
37:
38:     /* Returns all records in the database */
39:     public function getSnippets()
40:     {
41:        //We must specify our vo, because we need to map correctly
42:        require_once( "../vo/SnippetVO.php" );
43:
44:        $sql = mysql_query( "SELECT * FROM ". $this->table. "" );
45:
46:        $result = array();
47:
48:        while( $snip = mysql_fetch_array( $sql ) )
49:        {
50:           //Create a new snippet vo
51:           $snippet = new SnippetVO();
52:           $snippet->snippet_id          = $snip[snippet_id];
53:           $snippet->snippet_title       = $snip[snippet_title];
54:           $snippet->snippet_code          = $snip[snippet_code];
55:           $snippet->snippet_type          = $snip[snippet_type];
56:           $snippet->snippet_created       = $snip[snippet_created];
57:           $snippet->snippet_user          = $snip[snippet_user];
58:           //Result is a snippet
59:           $result[] = $snippet;
60:        }
61:        //Print out the result
62:        return $result;
63:     }
64:
65:     //This is used for returning the created or updated snippet for flex
66:     public function getOne( $id )
67:     {
68:        $rs = mysql_query( "SELECT * FROM ".$this->table." WHERE snippet_id = ".$id );
69:        //Map the recordset to our vo
70:        $list = $this->mapRecordSet( $rs );
71:        //Return our vo
72:        return $list[ 0 ];
73:     }
74:
75:     /**
76:     * This is the function that saves a item to the database if the snippet_id == 0.
77:     * But if the snippet_id doesn't equal 0 then it will update that snippet where
78:     * both id's are matching.
79:     *
80:     * @param @snippet Snippet object for mapping and insterting into database
81:     */
82:     public function saveSnippet( $snippet )
83:     {
84:        require_once( "../vo/SnippetVO.php" );
85:        //Check to see if the snippet has an id of 0
86:        if ( $snippet[snippet_id] == 0 )
87:        {
88:        $query = "INSERT INTO ".$this->table."
89:                           ( snippet_title,
```

```
 90:                          snippet_code,
 91:                          snippet_type,
 92:                          snippet_created,
 93:                          snippet_user )
 94:                          VALUES (
 95:                          '"
.mysql_real_escape_string($snippet[snippet_title])."',
 96:                          '"
.mysql_real_escape_string($snippet[snippet_code])."',
 97:                          '"
.mysql_real_escape_string($snippet[snippet_type])."',
 98:                          '"
.mysql_real_escape_string($snippet[snippet_created])."',
 99:                          '"
.mysql_real_escape_string($snippet[snippet_user])."')";
100:        if( !mysql_query( $query ) )
101:        {
102:           return false;
103:        }
104:           return $this->getOne( mysql_insert_id() );
105:
106:           } else {
107:                 //Update the snippet
108:                 $id = $snippet[snippet_id];
109:
110:                 $query = "UPDATE ".$this->table." SET
111:                 snippet_title = '"
.mysql_real_escape_string($snippet[snippet_title])."',
112:                 snippet_code = '"
.mysql_real_escape_string($snippet[snippet_code])."',
113:                 snippet_type = '"
.mysql_real_escape_string($snippet[snippet_type])."',
114:                 snippet_created = '"
.mysql_real_escape_string($snippet[snippet_created])."',
115:                 snippet_user = '"
.mysql_real_escape_string($snippet[snippet_user])."'
116:
117:                 WHERE snippet_id ="
. $id;
118:
119:                 if( !mysql_query( $query ) )
120:                 {
121:                    return false;
122:                 }
123:                 //Return the created snippet
124:                 return $this->getOne( $id );
125:              }
126:     }
127:
128:     /**
129:      * This function removes a snippet from the database based on the id passed
130:      * @param id id of the snippet to be removed
131:      */
132:     public function removeSnippet($id)
133:     {
134:        $sql = mysql_query( "DELETE FROM ".$this->table." WHERE snippet_id = ".$id );
135:
136:        if( !$sql )
137:        {
```

```
138:        //   trigger_error("Unable to delete Snippets", E_USER_ERROR);
139:           return "There was an error removing this snippet";
140:        }
141:        else return $id;
142:     }
143:
144:     /* For searching the database */
145:     public function search( $arr )
146:     {
147:        $keywords = $arr[0];
148:        $offset  = $arr[1];
149:        $this->offset = ($arr[1] > 0) ? $arr[1] : $this->offset;
150:
151:        $get_count = mysql_query("SELECT snippet_id FROM ".$this->table."
152:                              WHERE ( snippets.snippet_title OR snippets.snippet_code )
153:                              LIKE ( '$keywords' )"
);
154:
155:        $count = mysql_num_rows( $get_count );
156:        $total_snippets = array(
157:                             array(
158:                                'totalProducts'=>$count,
159:                                'offset' => $this->offset,
160:                                'pageSize' => $this->pagesize
161:                                )
162:                             );
163:
164:        $search = "SELECT * FROM products
165:                     WHERE ( snippets.snippet_title OR snippets.snippet_code )
166:                     LIKE ( '$keywords' ) ORDER BY snippet_id ASC
167:                     LIMIT  $this->offset, $this->pagesize"
;
168:
169:
170:        $snippets = mysql_query( $search );
171:
172:        if ( $snippets )
173:        {
174:           while( $row = mysql_fetch_object( $snippets ) )
175:        {
176:           $results[] = $row;
177:        }
178:           $return  = array_merge( $results, $total_snippets );
179:        } else {
180:           $return = array('DEBUG_OUTPUT', mysql_error());
181:        }
182:        return $return;
183:     }
184:
185:  }
186:  ?>
```

Recipe 1.6 - Making Calls

Problem:

You want to actually start using AMFPHP since you have the database set up, a full CRUD service and all necessary value objects.

Solution:

Before we can make any calls to AMFPHP we have to let Flex know where our gateway.php is, connecting to AMFPHP is probably one of the easiest things to do. It only takes 3 steps to connect to our server side remoting.

1. A NetConnection variable
ex. private var service:NetConnection = new NetConnection();

2. A String variable
ex. private var gateway:String = "http://localhost/amfphp/gateway.php";

3. Now connect
ex. service.connect(gateway);

Code:

And we are now connected to AMFPHP, no service-config.xml, no <mx:RemoteObject/> tags, just as above and you will be connected.

To make actual calls to a method that you have created you use syntax like the following:

```
1:  public function getSomeData():void
2:  {
3:     service.call ( "Path.ServiceName.MethodName", new Responder
( someResultHandler, someFaultHandler ), Arguments ) );
4:  }
```

Recipe 1.7 - Creating a Service Proxy

Problem:

You have all service methods on your server working as expected, but you have no access to them from Flex.

Solution:

Create a Service Proxy in ActionScript containing all necessary functions for connecting, calling methods and handling the result for your views

Code:

Use the following example for a reference when creating your own service proxy class for the methods that you created.

```
1:  package com.jonniespratley.snippr.services
2:  {
3:     import com.jonniespratley.snippr.model.ModelLocator;
4:     import com.jonniespratley.snippr.vo.SnippetVO;
```

```
  5:
  6:     import flash.net.NetConnection;
  7:     import flash.net.Responder;
  8:
  9:     import mx.collections.ArrayCollection;
 10:     import mx.controls.Alert;
 11:     import mx.rpc.events.ResultEvent;
 12:
 13:     /**
 14:      * This file is for use without! using the services-config.xml file
 15:      * @author Jonnie Spratley
 16:      * @website http://jonniespratley.com
 17:      *
 18:      */
 19:     public class SnipprService
 20:     {
 21:        /** NetConnection variable for creating our amfphp connection */
 22:        private static var _service:NetConnection;
 23:
 24:        /** Location of our gateway for amfphp */
 25:        private var gateway:String = "http://WEBSITE/amfphp/gateway.php";
 26:
 27:        /** Our model so we can update it when we receive our data */
 28:        private var model:ModelLocator = ModelLocator.getInstance();
 29:
 30:        /**
 31:         * Here we are creating a new connection to our amfphp service, when this
is instantiated,
 32:         * it connects to our service.
 33:         *
 34:         */
 35:        public function SnipprService()
 36:        {
 37:           _service = new NetConnection();
 38:           _service.connect( gateway );
 39:        }
 40:
 41:        /* ************************************************************************
 42:        *         All Service Calls to AMFPHP (updated)
 43:        *
 44:        *    This is where all of our service calls are taken, when our
 45:        *   outside componets calls these functions all required arguemtns
 46:        *   must be passed to properly send/update/delete data.
 47:        *
 48:        *   If arguments are not present Flex wont compile. In all of our
 49:        *   calls we attach assigned result handlers for the specific calls
 50:        *   that we are making. They all use the same fault handler.
 51:        ************************************************************************/
 52:
 53:
 54:        /**
 55:         * Here we are calling the getSnippets on our server (amfphp) and setting the
result and fault handlers
 56:         *
 57:         */
 58:        public function getSnippets():void
 59:        {
 60:           _service.call( "snippr.SnipprService.getSnippets", new
Responder( snippetResultHandler, snipprFaultHandler ) );
```

```
  61:            trace( "Gettings Snippets" );
  62:         }
  63:
  64:         /**
  65:          * We take one argument here, and that is a snippet, because our server (amfphp)
is expecting a snippetVO
  66:          *
  67:          * @param snippet snippetVO object
  68:          *
  69:          */
  70:         public function saveSnippet( snippet:SnippetVO ):void
  71:         {
  72:            _service.call( "snippr.SnipprService.saveSnippet", new
Responder( snippetSavedHandler, snipprFaultHandler ), snippet );
  73:            trace( "Saving Snippet" );
  74:         }
  75:
  76:         /**
  77:          * We take one argument here, and that is the id of the snippet we are wanting
to remove
  78:          *
  79:          * @param snippet_id the id to be removed
  80:          *
  81:          */
  82:         public function removeSnippet( snippet_id:uint ):void
  83:         {
  84:            _service.call( "snippr.SnipprService.removeSnippet", new
Responder( snippetRemoveHandler, snipprFaultHandler ), snippet_id );
  85:            trace( "Removing Snippet" );
  86:         }
  87:
  88:
  89:         /* ************************************************************
  90:         *         Result and Fault Handlers
  91:         *
  92:         *    This is where all of our result and fault handling is
  93:         *   going to take place, we updating the model on the results
  94:         *    that we get back. Or simply displaying to the user what
  95:         *    comes back to Flex.
  96:         ************************************************************/
  97:
  98:         /**
  99:          * We are handling the result coming back as an array of snippets,
 100:          * then we add our snippets to our model
 101:          *
 102:          * @param data the array of snippets
 103:          *
 104:          */
 105:         private function snippetResultHandler( data:Array ):void
 106:         {
 107:            model.snippetCollection = initVO( data );
 108:         }
 109:
 110:         /**
 111:          * We have this helper to help parse our
 112:          * result that is coming back, looping through
 113:          * all of the objects then creating an array collection
 114:          * from it.
 115:          *
```

```
116:          * @param resultArray array of objects
117:          * @return ArrayCollection of snippets
118:          *
119:          */
120:         private function initVO( resultArray:Array ):ArrayCollection
121:         {
122:            var tempArray:ArrayCollection = new ArrayCollection();
123:
124:            for ( var s:String in resultArray )
125:            {
126:               tempArray.addItem( new SnippetVO( resultArray[s] ) );
127:
128:            }
129:            return tempArray;
130:         }
131:
132:         /**
133:          * Here we are handling the result and adding it to the value of serviceResponse in
our model
134:          *
135:          * @param data the result from amfphp
136:          */
137:         private function snippetSavedHandler( data:Object ):void
138:         {
139:            ModelLocator.getInstance().serviceResponse = data.toString();
140:         }
141:
142:         /**
143:          * Here we are handling the result that is being returned, which will be the id of
the removed snippet,
144:          * removing it from our model, at the snippet index
145:          *
146:          * @param data we are just refreshing/calling for the snippets again.
147:          */
148:         private function snippetRemoveHandler( data:Object ):void
149:         {
150:            getSnippets();
151:         }
152:
153:         /**
154:          * Here we are alerting the user that there was an error connection to our server
155:          *
156:          * @param fault the fault object from the call
157:          */
158:         private function snipprFaultHandler( fault:Object ):void
159:         {
160:            Alert.show( "There was an error connecting to the server.", "Snippr Service Error" );
161:         }
162:     }
163: }
```

Now if you wanted to use any of these calls in your view use the following.

SnippetForm.mxml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <!--SnippetFormProxyService-->
```

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%" height="100%"
   creationComplete="init()"
   xmlns:components="com.jonniespratley.snippr.view.components.*">

   <mx:Script>
      <![CDATA[
         import mx.validators.Validator;
         import mx.controls.Alert;
         import mx.rpc.events.FaultEvent;
         import mx.rpc.events.ResultEvent;

         import com.jonniespratley.snippr.vo.SnippetVO;
         import com.jonniespratley.snippr.model.ModelLocator;
         import com.jonniespratley.snippr.services.SnipprService;

         /* Out Model so we can bind to the selectedSnippet */
         [Bindable] private var model:ModelLocator = ModelLocator.getInstance();

         /* Our validation array to hold the values of our validators */
         [Bindable] private var validators:Array = new Array();

         /* Our custom remote proxy service for connection to amfphp */
         private var service:SnipprService;

         /*
         On init we create a new instance of our service proxy
         We alway set our validators to our validator array
         */
         private function init():void
         {
            service = new SnipprService();

            validators = [ titleV, authorV, codeV, typeV ];
         }

         /*
         When the save button is clicked instead of sending the data right away
         we first check it to see if it is indeed valid. If our validation array
         is empty, then we can go ahead and send our value object to amfphp, other
         wise we need to alert the user that there are some errors in the form
         */
         private function checkForm():void
         {
            var vals:Array = new Array();
               vals = Validator.validateAll( validators );

            //If no errors
            if ( vals.length == 0 )
            {
               saveSnippet();
               //cleanForms();
            } else {
               Alert.show( "Please correct invalid form entries", "Validation Error" );
            }
         }

         /* Clears all form inputs, and resets the selected index of the snippet list */
         private function cleanForms():void
         {
```

```
62:              //Set the model.selectedSnippet to null, so we dont have any fields used up
63:              //model.selectedSnippet = null;
64:              txt_title.text = "";
65:              txt_author.text = "";
66:              txt_code.text = "";
67:              txt_type.text = "";
68:           }
69:
70:           /*
71:           The saveSnippet function that gets called when there is no errors in our form
72:           This is one function that is going to handle both creating a new snippet, and
73:           updating an existing one. Our server side php script says that if the
snippetVO[snippet_id]
74:           is equal to 0, then go ahead and insert it as a new snippet. But if the
snippetVO[snippet_id]
75:           is not set to 0, then update that snippet where the recieved id is equal to the
id we are updating.
76:           */
77:           private function saveSnippet():void
78:           {
79:              /* If the selectedSnippet is empty create a new snippet */
80:              if ( model.selectedSnippet == null )
81:              {
82:              var createS:SnippetVO = new SnippetVO();
83:                 createS.snippet_id = 0;
84:                 createS.snippet_title = txt_title.text;
85:                 createS.snippet_code = txt_code.text;
86:                 createS.snippet_user = txt_author.text;
87:                 createS.snippet_type = txt_type.text;
88:
89:              /* Service proxy */
90:              service.saveSnippet( createS );
91:
92:              } else {
93:                 /* Set the snippet id to the value of the selected snippet_id */
94:                 var updateS:SnippetVO = new SnippetVO();
95:                    updateS.snippet_id = model.selectedSnippet.snippet_id;
96:                    updateS.snippet_title = txt_title.text;
97:                    updateS.snippet_code = txt_code.text;
98:                    updateS.snippet_user = txt_author.text;
99:                    updateS.snippet_type = txt_type.text;
100:
101:             service.saveSnippet( updateS );
102:
103:             }
104:             /* Do nothing */
105:             return;
106:          }
107:
108:
109:          /* ****************** MXML Result and Fault handlers ******************** */
110:          private function onResult( event:ResultEvent ):void
111:          {
112:             Alert.show(event.result.toString(), "Success" );
113:          }
114:
115:          private function onFault( event:FaultEvent ):void
116:          {
117:             Alert.show( event.fault.faultString );
```

```
118:           }
119:
120:
121:
122:        ]]>
123:     </mx:Script>
124:
125:     <!--
126:     MXML Remote Object This is the way you could make the same calls to amfphp, the
only difference is
127:     you need to use a service-config.xml file or just specify the endpoint inside
the RemoteObject tag.
128:     -->
129:     <!--Remote Object-->
130:     <mx:RemoteObject id="snipprSvc"   source="snippr.SnipprService"
131:        destination="amfphp"
132:        showBusyCursor="true"
133:        fault="onFault( event )">
134:           <!--Methods that are on our server-->
135:           <mx:method name="saveSnippet" result="onResult( event )"/>
136:           <mx:method name="getSnippets" result="onResult( event )"/>
137:     </mx:RemoteObject>
138:
139:     <mx:ApplicationControlBar width="100%" styleName="formBar">
140:        <mx:HBox width="100%" verticalAlign="middle">
141:           <mx:Label text="Author:" fontWeight="bold"/>
142:           <mx:TextInput id="txt_author"
143:              text="{ model.selectedSnippet.snippet_user }"
144:              width="100%"/>
145:        </mx:HBox>
146:     </mx:ApplicationControlBar>
147:
148:     <mx:ApplicationControlBar width="100%" styleName="formBar">
149:        <mx:HBox width="100%" verticalAlign="middle">
150:
151:              <mx:Label text="Title:" fontWeight="bold"/>
152:              <mx:TextInput id="txt_title"
153:                 text="{ model.selectedSnippet.snippet_title }"
154:                  width="100%"/>
155:
156:              <mx:Label text="Type:" fontWeight="bold"/>
157:              <mx:TextInput id="txt_type"
158:                 text="{ model.selectedSnippet.snippet_type }"
159:                 width="100%"/>
160:
161:           <mx:Button id="btn_clear"
162:              click="cleanForms()"
163:               label="Clear"/>
164:           <mx:Button id="btn_save"
165:              click="checkForm()"
166:               label="Save"/>
167:        </mx:HBox>
168:     </mx:ApplicationControlBar>
169:           <mx:VBox width="100%" height="100%" label="Edit">
170:              <mx:TextArea id="txt_code"
171:                 text="{ model.selectedSnippet.snippet_code }"
172:                 width="100%"
173:                 height="100%"
174:                  styleName="codeView"/>
```

```
175:           </mx:VBox>
176:
177:     <!-- Validators -->
178:     <mx:StringValidator id="titleV"
179:        source="{ txt_title }"
180:        minLength="1"
181:        maxLength="200"
182:        required="true"
183:        property="text"/>
184:     <mx:StringValidator id="authorV"
185:        source="{ txt_author }"
186:        minLength="1"
187:        maxLength="200"
188:        required="true"
189:        property="text"/>
190:     <mx:StringValidator id="codeV"
191:        source="{ txt_code }"
192:        minLength="5"
193:        required="true"
194:        property="text"/>
195:     <mx:StringValidator id="typeV"
196:        source="{ txt_type }"
197:        minLength="1"
198:        maxLength="200"
199:        required="true"
200:        property="text"/>
201:  </mx:VBox>
```

SnippetList.mxml

```
  1:  <?xml version="1.0" encoding="utf-8"?>
  2:  <!--SnippetList-->
  3:  <mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="200"
height="100%" creationComplete="init()">
  4:
  5:     <mx:Script>
  6:        <![CDATA[
  7:           import mx.controls.Alert;
  8:           import mx.events.CloseEvent;
  9:           import com.jonniespratley.snippr.services.SnipprService;
 10:           import com.jonniespratley.snippr.vo.SnippetVO;
 11:           import com.jonniespratley.snippr.events.SnippetGetEvent;
 12:           import com.jonniespratley.snippr.model.ModelLocator;
 13:
 14:           /* Make a instance of our model for our data display */
 15:           [Bindable] private var model:ModelLocator = ModelLocator.getInstance();
 16:
 17:           /* Make a variable to check weither there is a selected snippet or not */
 18:           [Bindable] private var isSelected:Boolean = false;
 19:
 20:           /* Make variable of our service */
 21:           private var service:SnipprService;
 22:
 23:           private function init():void
 24:           {
 25:              service = new SnipprService();
 26:              getSnippets();
```

```
27:           }
28:
29:           /* Send a call to get the snippets */
30:           private function getSnippets():void
31:           {
32:              service.getSnippets();
33:           }
34:
35:           /*
36:            * Alert the user and see what the response is
37:            * if the response is yes, then delete, else return
38:            */
39:           private function removeSnippet():void
40:           {
41:              Alert.show( "Are you sure?", "Remove Snippet", 3,
null, removeSnippetAlertHandler );
42:           }
43:
44:           /*
45:            * Remove snippet handler when the alert box is a yes
46:            */
47:           private function removeSnippetAlertHandler( event:CloseEvent ):void
48:           {
49:              if ( event.detail == Alert.YES )
50:              {
51:                 service.removeSnippet( lt_snippets.selectedItem.snippet_id );
52:              }
53:           }
54:
55:           /* Make sure we handle the selected snippet and bind it to our model */
56:           private function selectHandler( event:Event ):void
57:           {
58:              isSelected = true;
59:              model.selectedSnippet = event.target.selectedItem as SnippetVO;
60:           }
61:
62:        ]]>
63:     </mx:Script>
64:
65:     <!--Helper for Data Binding-->
66:     <mx:Binding destination="lt_snippets.selectedItem" source="model.selectedSnippet"/>
67:
68:        <!--List of Snippets-->
69:        <mx:List id="lt_snippets"
70:           dataProvider="{ model.snippetCollection }"
71:           change="selectHandler( event )"
72:           labelField="snippet_title"
73:           width="100%"
74:           height="100%"/>
75:
76:        <!--Remove Button-->
77:        <mx:Button label="Remove"
78:           click="removeSnippet()"
79:           enabled="{ isSelected }"
80:           width="100%"/>
81:  </mx:VBox>
```

Recipe 1.8 - Sending Emails with AMFPHP

Problem:

You want to be able to stay in touch with your users and to be able to do this, sending emails would be a big plus.

Solution:

Create a new method in your service class that allows AMFPHP to recieve and send out email. Then add the same functions to Flex where the magic happens.
We use a file called eMail.php the information about that file is provided inside the source code at the end if you want to look.

Code:

All the following code to your **SnipprService.php** file:

```
1:  //eMail Class
2:  require_once "eMail.php";
3:
4:  /**
5:  * -> We take one argument here that is emailVO,
6:  * Flex is sending over a value object that has all
7:  * of the details needed to successfully send an email
8:  * to the specified address in the vo.
9:  *
10: * @param emailVO The email object from Flex.
11: */
12: public function sendEmail( $emailVO )
13: {
14:    //Create a new email Object
15:    $email = new eMail( "Flex Mail Form", $emailVO[email_from] );
16:    {
17:       //Create Subject Line
18:       $email->subject( $emailVO[email_subject] );
19:
20:       //To ( email address )
21:       $email->to( $emailVO[email_to] );
22:
23:       //From ( email address )
24:       $email->bcc( $emailVO[email_from]);
25:
26:       //HTML Message
27:       $email->html( $emailVO[email_message] );
28:
29:       //Send e-mail
30:       $email->send();
31:    }
32:    return "Message Sent";
33: }
```

And Now for the **SnipprSerivce.as** File:

```
 1:  /**
 2:   * We take one argument here, and that is a email value object.
 3:   * We are passing this object to amfphp where our email will be sent
 4:   *
 5:   * @param email
 6:   *
 7:   */
 8:  public function sendEmail( email:EmailVO ):void
 9:  {
10:     _service.call( "snippr.MediaService.sendEmail", new Responder
( emailResultHandler, snippetSavedHandler ), email );
11:     trace( "Sending Email" );
12:  }
13:
14:  /**
15:   * We are taking the data object as the result,
16:   * tracing it and we could display an alert to
17:   * the user showing him/her whatever message
18:   * we want.
19:   *
20:   * @param data a message showing us the status
21:   *
22:   */
23:  private function emailResultHandler( data:Object ):void
24:  {
25:     var result:ResultEvent = data as ResultEvent;
26:     trace( data );
27:  }
```

Can't forget the **EmailVO.as**

```
 1:  package com.jonniespratley.snippr.vo
 2:  {
 3:     [Bindable]
 4:     public class EmailVO
 5:     {
 6:        public var email_to:String;
 7:        public var email_from:String;
 8:        public var email_subject:String;
 9:        public var email_message:String;
10:
11:        public function EmailVO()
12:        {
13:        }
14:     }
15:  }
```

And the view, **EmailForm.mxml:**

```
 1:  <?xml version="1.0" encoding="utf-8"?>
 2:  <mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="350"
 3:     showCloseButton="true"
 4:     close="PopUpManager.removePopUp( this )"
```

```
    creationComplete="init()"
    title="Send a Email">

    <mx:Script>
       <![CDATA[
          import mx.controls.Alert;
          import mx.validators.Validator;
          import mx.managers.PopUpManager;
          import com.jonniespratley.snippr.vo.EmailVO;
          import com.jonniespratley.snippr.services.SnipprService;


          /* Our ServiceProxy variable */
          private var service:SnipprService;

          /* Our validation array to hold the values of our validators */
          [Bindable] private var validators:Array = new Array();

          /* When the component creation completes create a new service */
          private function init():void
          {
             service = new SnipprService();

             validators = [ toV, fromV, subjectV, messageV ];
          }

          /*
           * -> When the save button is clicked instead of sending the data right away
           * we first check it to see if it is indeed valid. If our validation array
           * is empty, then we can go ahead and send our value object to amfphp, other
           * wise we need to alert the user that there are some errors in the form
           */
          private function checkForm():void
          {
             var vals:Array = new Array();
                vals = Validator.validateAll( validators );

             //If no errors
             if ( vals.length == 0 )
             {
                //Send the Email
                sendEmail();

             } else {
                Alert.show( "Please correct invalid form inputs", "Validation Error" );
             }
          }

          /*
           * -> When called we take our emailVO and build it up with the
           * values from the inputs then we call the sendEmail(email) function
           * in our SnipprService file, passing the packaged emailVO as the
           * required argument.
           */
          private function sendEmail():void
          {
             /* Build up the value object for sending */
          var   emailVO:EmailVO = new EmailVO()
             emailVO.email_to       = txt_emailTo.text
```

```
64:              emailVO.email_from       = txt_emailFrom.text
65:              emailVO.email_subject    = txt_emailSubject.text
66:              emailVO.email_message    =   txt_emailMessage.text
67:
68:              /* Call the service passing the emailVO as the argument */
69:              service.sendEmail( emailVO );
70:           }
71:
72:
73:           /* Resets the form and clears any validation */
74:           private function resetForm():void
75:           {
76:              txt_emailTo.text = "";
77:              txt_emailFrom.text = "";
78:              txt_emailMessage.text = "";
79:              txt_emailSubject.text = "";
80:              txt_responseMessage.text = "";
81:           }
82:
83:
84:        ]]>
85:     </mx:Script>
86:
87:     <mx:Form width="100%" height="100%">
88:        <!--Email To-->
89:        <mx:FormItem label="To:"
90:           width="100%" required="true">
91:           <mx:TextInput id="txt_emailTo"
92:              width="100%"/>
93:        </mx:FormItem>
94:
95:        <!--Email From-->
96:        <mx:FormItem label="From:"
97:           width="100%" required="true">
98:           <mx:TextInput id="txt_emailFrom"
99:              width="100%"/>
100:       </mx:FormItem>
101:
102:       <!--Email Subject-->
103:       <mx:FormItem label="Subject:"
104:          width="100%" required="true">
105:          <mx:TextInput id="txt_emailSubject"
106:             width="100%"/>
107:       </mx:FormItem>
108:
109:       <!--Email Message-->
110:       <mx:FormItem label="Message:"
111:          width="100%"
112:          height="100%" required="true">
113:          <mx:TextArea id="txt_emailMessage"
114:             width="100%"
115:             height="100%"/>
116:       </mx:FormItem>
117:    </mx:Form>
118:
119:    <mx:ControlBar horizontalAlign="right" verticalAlign="middle">
120:
121:       <!--Cancel Button-->
122:       <mx:Button id="btn_cancel"
```

```
123:            label="Cancel"
124:            click="resetForm()"/>
125:
126:        <!--Response Message-->
127:        <mx:Text id="txt_responseMessage"
128:            width="100%"
129:            fontSize="14"
130:            color="#4F0A59"
131:            fontWeight="bold"
132:            textAlign="center"/>
133:
134:        <!--Send Button-->
135:        <mx:Button id="btn_send"
136:            label="Send"
137:            click="checkForm()"/>
138:    </mx:ControlBar>
139:
140:    <!-- Validators -->
141:    <mx:EmailValidator id="toV"
142:        source="{ txt_emailTo }"
143:        required="true"
144:        property="text"/>
145:    <mx:EmailValidator id="fromV"
146:        source="{ txt_emailFrom }"
147:        required="true"
148:        property="text"/>
149:    <mx:StringValidator id="subjectV"
150:        source="{ txt_emailSubject }"
151:        minLength="3"
152:        maxLength="50"
153:        required="true"
154:        property="text"/>
155:    <mx:StringValidator id="messageV"
156:        source="{ txt_emailMessage }"
157:        minLength="5"
158:        maxLength="5000"
159:        required="true"
160:        property="text"/>
161: </mx:TitleWindow>
```

Recipe 1.9 - Sending Images with AMFPHP

Problem:

You want to be able to take a screenshot of your Air application because ZScreen does not capture it.

Solution:

Add another method to your server side PHP script, and add the function for accessing this service in ActionScript then call this method from the view.

Code:

You want to add this to your PHP service:

```
1: //Make sure this folder exsists and is writable
```

```
  2:  //Specify our output directory
  3:  var $output_dir = "screenshots";
  4:
  5:  //If the full path to the screenshots folder is http://website.com/amfphp/services/snippr/
  6:  //then just drop the snippr from the url, but leave the /
  7:  //Specify our output url
  8:  var $server_url = "http://WEBSITE.com/amfphp/services/snippr/";
  9:
 10:  /** ****************************************************************************
 11:  * -> Save image from the given bytearray and return the path of the saved image
 12:  ******************************************************************************/
 13:  public function takeScreenshot( $byteArray, $filename, $compressed = false )
 14:  {
 15:     /** -> Check if our folder exists, and also if it is writeable */
 16:     if( !file_exists( $this->output_dir ) || !is_writeable( $this->output_dir ) )
 17:
 18:        //If it is not there, throw a error
 19:        trigger_error ( "Please create a temp directory with write access", E_USER_ERROR );
 20:
 21:     //Set a data variable, and then set it to the value of byteArray (from Flex) the
data inside the bytearray
 22:     $data = $byteArray->data;
 23:
 24:     //If it is compressed
 25:     if( $compressed )
 26:     {
 27:        //Check if php server even has gzip installed
 28:        if( function_exists( gzuncompress ) )
 29:        {
 30:           //if so then uncompress it
 31:           $data = gzuncompress( $data );
 32:        } else {
 33:           //or throw a error
 34:           trigger_error ( "Gzuncompress method does not exists, please send
uncompressed data", E_USER_ERROR );
 35:        }
 36:     }
 37:     //Put the File in the Directory, and Rename it, what the User wanted the Name to be.
 38:     file_put_contents( $this->output_dir . "/$filename", $data );
 39:
 40:     //Return the url to the user, so we can pop up a window with the new image!
 41:     return $this->server_url . $this->output_dir . "/$filename";
 42:  }
```

Now add this to your ActionScript service:

```
  1:  /**
  2:   * We take two arguments here, one is a byte array and the other is the filename of the file
  3:   *
  4:   * @param bytes byte array
  5:   * @param filename filename of the file
  6:   *
  7:   */
  8:  public function takeScreenshot( bytes:ByteArray, filename:String ):void
  9:  {
 10:     _service.call( "snippr.SnipprService.takeScreenshot", new
```

```
Responder( snapshotResultHandler, snipprFaultHandler ), bytes, filename );
   trace( "Sending Screenshot" );
}

/**
 * We are taking the data object as the result, and we just
 * add an Alert with the result which is a string to the source
 * of the image.
 *
 * @param data image url
 */
private function snapshotResultHandler( data:Object ):void
{
   var result:ResultEvent = data as ResultEvent;

   Alert.show( "Nice shot, here is the link to your shot." + data, "Screenshot Saved" );

   trace( data );
}
```

Now the view and you are sending screenshots:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%"
   creationComplete="init()">

   <mx:Script>
      <![CDATA[
         import com.jonniespratley.snippr.services.SnipprService;
         import com.jonniespratley.snippr.events.SnapScreenshotEvent;

         import mx.utils.Base64Encoder;
         import mx.controls.Image;
         import mx.graphics.ImageSnapshot;
         import mx.graphics.codec.PNGEncoder;
         import mx.collections.ArrayCollection;
         import mx.utils.UIDUtil;

         //Filename variable
         private var fileName:String = "";
         //Our service
         private var service:SnipprService;

         //Connection
         private function init():void
         {
            service = new SnipprService();
         }


         /*
         The takeScreenShot function that will create a new bitmap data variable,
         and take the width and height of our stage (window) and create a image from it.
         Then we will add this to the clipboard for easy pasting as well as encoding it
         into a bytearray for easy transfering to amfphp.
         */
```

```
35:             private function takeScreenshot():void
36:             {
37:                //Set the filename param on amfphp to the textinput text, add a .png cause
its a png!
38:                 fileName = txt_filename.text +".png";
39:
40:                //New Bitmap data variable set the the height and with of the stage
41:                var bitmapData:BitmapData = new BitmapData( stage.width, stage.height );
42:                      //Draw the stage
43:                   bitmapData.draw( stage );
44:              //Create clipboard variable
45:              var clipboard:Clipboard = Clipboard.generalClipboard;
46:                   //Clear whatever was on there prior
47:                   clipboard.clear();
48:                   //Set the clipboard data to the bitmapData variable, notice how we
didnt encode it!
49:                   clipboard.setData( ClipboardFormats.BITMAP_FORMAT, bitmapData );
50:
51:                //Create a byte array variable, then PNGEncoder to encode what, the bitmapData
52:                var bytes:ByteArray = new PNGEncoder().encode( bitmapData );
53:
54:                //Dispatch the SnapScreenshotEvent passing the bytes(byteArray), and
the filename(string)
55:                  //(bytes, fileName);
56:                  //var evt:SnapScreenshotEvent = new SnapScreenshotEvent( bytes, fileName );
57:                     //evt.dispatch();
58:                  service.takeScreenshot( bytes, fileName );
59:             }
60:
61:        ]]>
62:     </mx:Script>
63:         <mx:HBox width="100%" verticalAlign="middle">
64:            <mx:Label text="Screenshot Name"/>
65:
66:            <!--The File name for the screenshot-->
67:            <mx:TextInput id="txt_filename"
68:               width="100%"/>
69:               <mx:Button id="btn_capture"
70:                   label="Capture"
71:                   width="100%"
72:                   click="takeScreenshot()"/>
73:        </mx:HBox>
74:  </mx:VBox>
```

Recipe 1.10. PHP to ActionScript Datatypes

Problem:

You want to know the datatype conversions of both languages.

Solution:

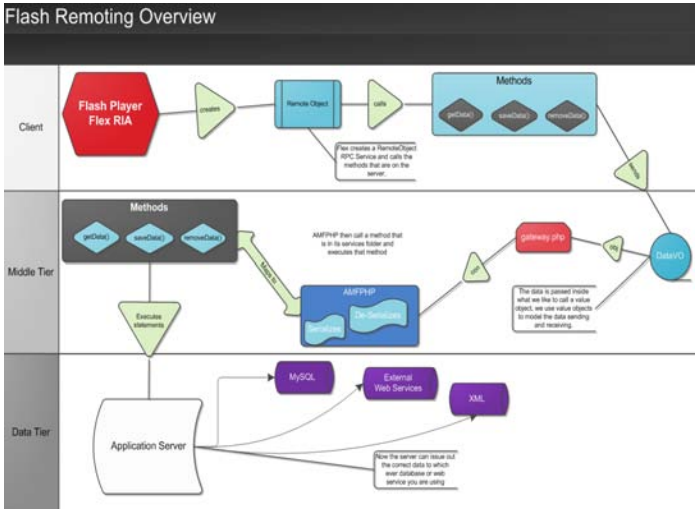
Read a table with that info!

PHP-to-ActionScript datatype conversion

PHP	Flash (ActionScript)
Null	Null
Integer	Integer
Double	Float
String	String
Array (normal)	Array
Array (associative)	Object
Object	Object
Resource	Recordset

Resources

AMFPHP diagram for the visual learners.



Here are the source files

- Snippr (Flex Source)
- SnipprService (amfphp services)
- AMFPHP Cookbook (PDF)

I did this for fun, and to teach others about the benifits of using AMFPHP with Flex. A fact is that you will create Rich Internet Applications faster, more efficient and more reliable then ever before. Enjoy!

-Jonnie Spratley
http://jonniespratley.com