# Zend_Amf - Wade Arnold

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

## Zend Framework: Zend_Amf Component Proposal

| | |
|---|---|
| **Proposed Component Name** | Zend_Amf |
| **Developer Notes** | http://framework.zend.com/wiki/display/ZFDEV/Zend_Amf |
| **Proposers** | Wade Arnold |
| **Zend Liaison** | Stanislav Malyshev |
| **Revision** | 1.0 - 1 January 2008: Initial Draft. (wiki revision: 68) |

## Table of Contents



## 1. Overview

Zend_Amf_Server provides Action Message Format (AMF3) support for the Zend Framework that is compatible with the Flash Player 9 and above. The Adobe Integrated Runtime (AIR) and Flash Player uses AMF to communicate between an application and a remote server. AMF encodes remote procedure calls (RPC) into a compact binary representation that can be transferred over HTTP/HTTPS protocol. Objects and data values are serialized into this binary format, which increases performance as the AMF serialization is a highly optimized procedure in the Flash Player. Zend_Amf_Server will act as an AMF gateway to the Zend Framework by exposing through introspection custom class objects and functions that will respond as callbacks through the Zend_Amf_Server gateway.

### What is AMF?

Action Message Format(AMF) is a binary file format representing a serialized ActionScript object. The AMF file type is used throughout the Flash Player for data storage and data exchange. For example in the Flash Player AMF is used in SharedObjects, RemoteObjects, LocalConnection, ByteArray, RTMP, and all RPC operations. Some of the benefits of AMF include:

- File Size - AMF objects are very small and are compressed using zlib.
- Fast Serialization/ Deserialization - AMF is transformed using native C code in the Flash Player making it very fast. The AMF format was designed to serialize and deserialize quickly under low memory and slower CPU conditions making it perfect for the web. AMF data is parsed directly into objects, meaning there is no lag for interpretation or parsing of AMF making the creation of objects complete in a single pass.
- Native Types and Custom classes supported - You can serialize any object in Flash Player with the only exception being a displayObject.you can also map serialized objects back to custom class instanced provided the custom class is in the Flash Player when

the AMF object is deserialized.

AMF existed in ActionScript 2 and was just called AMF as of ActionScript 3 the AMF protocol has been updated and is referred to as AMF3. For historical reasons the original AMF format is now referred to as AMF0. One of the main upgrades to AMF3 is that the object is now zlib compressed for faster transfer do to the smaller file size and the additional of data types that were released with ActionScript 3.

#### Model-View-Controller

The architectural role of Flex/Flash/Air in an MVC architecture is of the View. We need to assume that the server has more resources than the client. We can not explicitly trust the client. We assume that the view of the client will change in the future. With these assumptions in mind it is important that the end service that Zend_Amf provides is that of a controller. The controller may need to format data to prepare it for its presentation in the view and will also sanitize any incoming data and authorize the request. If you are use to working with php you should think of the Flash Player application like you would think of a smarty template or Zend_View.

It will be tempting to bypass the controller in RIA development or to place the functionality of a controller into action script. Remember our assumptions about the client and also remember that just because you are capable of functionality in ActionScript does not mean that you should.

## 2. References

- Adobe AMF Specification

## 3. Component Requirements, Constraints, and Acceptance Criteria

*PHP5
**Standard PHP Library (SPL)
**PHP5 Reflection extension
*Zend Framework

## 4. Dependencies on Other Framework Components

- Zend_Server_Interface
- Zend_Server_Reflection
- Zend_Server_Reflection_Function_abstract
- Zend_Server_Reflection_Method

## 5. Theory of Operation

Zend_Amf_Server is composed of several components, ranging from the server itself to request, response, encoding, decoding, and fault objects.

To implement Zend_Amf_Server, the developer must attach one or more classes or functions to the server instantiation, via the setClass() and addFunction() methods.

The Zend_Amf_Server will decode request objects that are received via php://input and deserialized the request and check for an existing instantiated session. Zend_Amf_Request will check the AMF header for the matching name space of the requested remote object to be called. The remote class will be instantiated and the result of the method call will be serialized via Zend_Amf_Vaslue and returned to the requested http connection via Zend_Amf_Response.

### Type Mapping Table

The following mapping defines what the resulting data type would be when sent from ActionScript to PHP and PHP to ActionScript. If a data type that is not specified in this grid is sent to or from Zend_amf it will be handle as a generic object. You can then type cast the object if necessary.

| **ActionScript to PHP mapping** | |
|---|---|
| **ActionScript** | **PHP** |
| undefined | null |
| null | null |
| int | integer |
| number | float |
| boolean | boolean |
| String | string |
| array | array |
| xml | DomDocument |
| flash.utils.ByteArray | string |
| uint | float |
| object | object |
| *RemoteClass* Object | class mapped object |
| date | DateTime |
| mx.collections.ArrayCollection | object |

| **PHP to ActionScript mapping** | |
|---|---|
| **PHP** | **ActionScript** |
| null | null |
| boolean | boolean |
| string | string |
| DomDocument | xml |
| DateTime | date |
| float | number |
| integer | number |
| Associative Array w/ mix of keys | Object |
| Associative Array w/ Numeric index of keys | Array |
| object | object |
| *RemoteClass* Zend_Amf_Value_TypedObject | typed object |
| Zend_Amf_Value_ByteArray | flash.utils.ByteArray |
| Zend_Amf_Value_ArrayCollection | mx.collections.ArrayCollection |

## Example of serialization/deserialization

The following is used to showcase passing an array to and from the Zend Framework. It is not a tutorial for how data from ActionScript we have a RemoteObject called myservice that's endpoint is Zend_Amf (use case below). We are calling a remote method called sortArray which takes an array as an argument.

```
// ActionScript
private var myArray:Array;
//~~~
private function callSortArray():void {
myArray = new Array("b", "a", "d","c");
myservice.sortArray(myArray);
}
```

In PHP we create a simple class that just re-orders the array and return an array. This is showing how the data types are handled. Zend_AMF hides all serialization from the end user on both sides. This allows the client and server application developers to leverage all of there existing capabilities and does not inject a new set of rules to make the php class Zend_AMF specific.

```
class ArrayUtil {
public function sortArray($newArray) {
sort($newArray);
return $newArray;
}
}
?>
```

The result is returned to ActionScript through its callback handler. There is no type handling the method just receives an array.

```
// ActionScript
private function sortArrayHandler(event:ResultEvent):void {
myArray = event.result;
trace(myArray.length); // 4
trace(myArray);   // Output: a,b,c,d
}
```

### Handling Exceptions via Faults

Zend_Amf_Server catches Exceptions generated by a dispatched method, and generates an Zend_Amf_Server_Fault response when such an exception is caught. The exception messages and codes are not used in a fault response which is return to the client as a Fault event. These events will be logged only as information about the error could potentially compromise the application.

## 6. Milestones / Tasks

## 7. Class Index

- Zend_Amf_Fault
- Zend_Amf_Request
- Zend_Amf_Request_Http
- Zend_Amf_Response
- Zend_Amf_Server
- Zend_Amf_Server_Exception
- Zend_Amf_Server_Fault
- Zend_Amf_Util
- Zend_Amf_Util_BinaryStream
- Zend_Amf_Value

- Zend_Amf_Value_ArrayCollection
- Zend_Amf_Value_ByteArray
- Zend_Amf_Value_TypedObject
- Zend_Amf_Value_ITypedObject
- Zend_Amf_Value_Message
- Zend_Amf_Value_Message_AcknowledgeMessage
- Zend_Amf_Value_Message_CommandMessage
- Zend_Amf_Value_Message_ErrorMessage
- Zend_Amf_Value_Message_RemotingMessage
- Zend_Amf_Parse
- Zend_Amf_Parse_Number
- Zend_Amf_Parse_Boolean
- Zend_Amf_Parse_String
- Zend_Amf_Parse_Object
- Zend_Amf_Parse_Null
- Zend_Amf_Parse_Undefined
- Zend_Amf_Parse_Refrence
- Zend_Amf_Parse_MixedArray
- Zend_Amf_Parse_Array
- Zend_Amf_Parse_Date
- Zend_Amf_Parse_Xml
- Zend_Amf_Parse_TypedObject
- Zend_Amf_Parse_Amf3_Integer
- Zend_Amf_Parse_Amf3_Date
- Zend_Amf_Parse_Amf3_String
- Zend_Amf_Parse_Amf3_Xml
- Zend_Amf_Parse_Amf3_ByteArray
- Zend_Amf_Parse_Amf3_Array
- Zend_Amf_Parse_Amf3_Object

## 8. Use Cases

Getting started with Zend_Amf requires a basic installation of the Zend Framework on your server. Review the getting started guide for setting up your webserver. http://framework.zend.com/manual/en/introduction.installation.html

For a reference we are going to assume that your web server folder is setup in the following manor.

/data/
/public or /htdocs or /www /include/
/library/Zend/Amf/
/include/
*Where the Zend Framework is placed into the /library directory*

You will then need to create an index.php where all of your AMF calls will be handled. For this example all calls to the Zend Framework are from an application running in the flash player. If you are using Flash and html views together you will need to create a gateway file independent of the index.php page that has the same content.

The following is an example of your index.php file that will act as the endpoint for AMF calls. The following code instantiates the Zend_Amf server class. You then tell the class what remote services or names spaces that you want to allow your Flasher Player application to access. In the following example we have specified the HelloWorld class to be added into the Zend_Amf server to be remotely exposed. With your webserver properly configured the following file should be accessible at http://localhost/index.php

```
require_once 'Zend/Amf/Server.php';
require_once 'include/services/HelloWorld.php'
// Instantiate server
$server = new Zend_Amf_Server();
$server->setClass('HelloWorld');
// Handle request
$server->handle();
```

There is nothing special to Zend_Amf for creating service classes. It is just a standard PHP class file. The following class HelloWorld has one public function say which takes in a string and returns a string that has been concatenated with today's date. Note that private methods will not be exposed through Zend_Amf. Place the file into /include/services/HelloWorld.php

```
class HelloWorld {
/**
* Say hello!
*
* @param  string $sMessage
* @return string
**/
public function say($sMessage) {
$date = getdate();
return 'You said: ' . $sMessage .' on '.$date[weekday];
}
}

?>
```

We will use Flex to connect to the application. The following code creates a RemoteObject endpoint that terminates at http://localhost/ which is where our following example is located. The code creates a simple input field and a button that send that data to the server on the click event.

Start with a new flex project from inside of flex builder and call it zend. You should now have an open project called product in your Navigator window. Right click on the zend project name and select 'properties'. In the Project properties dialog go into 'Flex Build Path' menu, 'Library path' tab and be sure the 'rpc.swc' file is added to your projects path. Press Ok to close the window.

We now need to tell Flex which services configuration file to use for inspecting our remote methods. For this reason create a new 'services-config.xml' file into your Flex project root folder. To do this right click on the zend project folder and select 'new' 'File' which will popup a new window. Select the product folder and then name the file 'services-config.xml' and press finish.

Flex has created the new services-config.xml and has it open. Use the following example text for your services-config.xml file. Make sure that you update your endpoint to match that of your testing server. Make sure you save the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
 <services>
  <service id="amfphp-flashremoting-service"
class="flex.messaging.services.RemotingService"
messageTypes="flex.messaging.messages.RemotingMessage">
   <destination id="zend">
    <channels>
     <channel ref="my-zend"/>
    </channels>
    <properties>
     <source>*</source>
    </properties>
   </destination>
  </service>
 </services>
 <channels>
  <channel-definition id="my-zend" class="mx.messaging.channels.AMFChannel">
   <endpoint uri="http://localhost/" class="flex.messaging.endpoints.AMFEndpoint"/>
  </channel-definition>
 </channels>
</services-config>
```

Now open your project properties panel again by right clicking on the project folder from your Navigator and selecting properties. From the properties popup select 'Flex Compiler' and add the string: -services "services-config.xml". Press Apply then OK to return to update the option. What you have just done is told the Flex compiler to look to the services-config.xml file for runtime variables that will be used by the RemotingService class.

The following code is just in MXML and you can add it to your default application. Run the code and you will remoting with the Zend Framework via Zend_Amf.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
 <mx:RemoteObject id="myservice" fault="faultHandler(event)" showBusyCursor="true"
source="HelloWorld" destination="zend">
        <mx:method name="say" result="resultHandler(event)" />
    </mx:RemoteObject>
     <mx:Script>
        <![CDATA[
      import mx.managers.CursorManager;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;
            private function faultHandler(fault:FaultEvent):void
            {
                CursorManager.removeBusyCursor();
                trace("code:\n" + fault.fault.faultCode + "\n\nMessage:\n" +
fault.fault.faultString + "\n\nDetail:\n" + fault.fault.faultDetail);
            }
            private function resultHandler(event:ResultEvent):void
            {
             response_txt.text = event.result.toString();
            }
        ]]]><![CDATA[>
    </mx:Script>
 <mx:TextInput x="10" y="20" id="server_txt" text="Connect to Zend Amf" />
 <mx:TextArea x="10" y="50" id="response_txt" width="278"/>
 <mx:Button x="178" y="20" label="Send to Server" id="send_btn"
click="myservice.say(server_txt.text)"/>
</mx:Application>
```

It should be noted that additional examples should be constructed that will properly utilize an MVC architecture and folder names. A good database example would be a great next step.

## 9. Class Skeletons

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>