ajaxCFC

Description

AJAX is Asynchronous JavaScript and XML. It is not a technology, but an umbrella that combines several concepts to enrich user experience by allowing server interaction without refreshing the browser.

ajaxCFC is a ColdFusion framework meant to speed up ajax application development and deployment by providing developers seamless integration between JavaScript and ColdFusion, and providing built-in functions to quickly adapt to any type of environment, security, and helping to overcome cross-browser compatibility problems.

ajaxCFC is designed as ColdFusion components, following the best practices of object oriented programming and design patterns. Programming with ajaxCFC involves extending components and creating your own ajax façades.

License and Credits

Copyright 2006 Rob Gonda

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Installation

ajaxCFC requires ColdFusion MX 6.0+ or Blue Dragon 6.1+ for the server side, and any browser with JavaScript 1.0+, XML Http object, or iFrame support. I have tested the framework with Internet Explorer 5.5+, Firefox 1.0+, Mozilla 1.7+, Nescape 7.0+, Safari 1.3+, and Opera 7.0+. It should also work variations or branches of these browsers as well.

When you unzip the installation package, you will have three main directories, core, documentation, and examples. The core folder represents the main application. The core folder contains everything you need: ajax.cfc and a js folder. The js folder has to be web accessible, while the ajax.cfc does not necessarily have to be.

You may unzip the examples anywhere under your web root and they should run without any modifications.

Because all your ajax component have to extend ajax.cfc, if you decide not to place it in the same folder you will probably need a ColdFusion mapping to access ajax.cfc wherever it is.

Extending ajax.cfc

To create ajax listeners you will need to extend ajax.cfc.

For example:

```
<cfcomponent extends="ajax">
	<cffunction name="echo" output="no" access="private">
		<cfargument name="str" required="Yes" type="string">
	<cfreturn arguments.str />
	</cffunction>
</cfcomponent>
```

If the arguments passed by JavaScript are unnamed, ajaxCFC will match them to your by the order they were passed. You may also pass an arguments collection to ajaxCFC by passing a single JavaScript object and ajaxCFC will match the object's keys to your arguments.

If you decide to store ajax.cfc in a different folder than your components, you will need to extend it with a mapping:

```
<cfcomponent extends="cf.mapping.to.ajax">
	<cffunction name="echo" output="no" access="private">
		<cfargument name="str" required="Yes" type="string">
	<cfreturn arguments.str />
	</cffunction>
</cfcomponent>
```

Note also that your methods do not need to have remote access, making them more secure.

The functions can output any type of simple or complex variables: strings, numbers, arrays, structures, and queries. The return will be serialized and picked up natively by JavaScript.

Including the JavaScript files

There is only need to initialize one object and include one file:

```
<script type='text/javascript'>
```

```
_ajaxConfig = {'_cfscriptLocation':'echoTest.cfc',
               '_jsscriptFolder':'../js',
               'debug':true};</script>
```

<script type='text/javascript' src='../js/ajax.js'></script>

'_ajaxConfig' will store the location of the listener CFC and location of the 'js' folder that was included in the ajax.zip file you downloaded. You are not forced to use the '_cfscriptLocation' variable as you may specify the CFC location directly in the ajax call. 'debug' mode will include and enable log4javascript. By default, log4javascript will show you all invoked calls, responses, as well as server side trapped errors.

The JavaScript functions structure

Typically you will have an ajax calling function and a call-back function; however, none of them are mandatory, because depending on your needs you may not need to know the result of your call, or you may not need to enclose your ajax call in a function. Nonetheless, for simplicity let's assume you will be using a typical ajax call.

To invoke an ajax component you need to use 'DWREngine._execute' in the following format:
DWREngine._execute(_ajaxConfig._cfscriptLocation, null, 'method', args1, args2, args3, callBackFunction);

'_execute' takes three or more arguments:
1st: path to listener CFC that contains the method you need to call
2nd: scriptName to execute; this argument will be null 99% of the times.
3rd: methodName; this is the function inside the CFC you want to execute.
4th: arguments; you may pass one or more arguments to the ColdFusion function. You can add additional arguments as needed. These arguments are optional.
5th: Call-back function; ajaxCFC will check if the very last argument is a function will asynchronously (by default) call that function after resolving the ColdFusion function and getting back a result.

The call-back function will always take one argument, which is the return of the method called within the CFC object.

Return Types

ajaxCFC will allow you to return almost any simple of complex object to the JavaScript callback function: strings, numbers, arrays, structures, and queries.

The callback function will always take one argument, which is the return of the invoked CF function. (Assume that the return is passed to the callback function with the name of 'result').

Receiving **simple values** is straight forward; the JavaScript function will receive the number or string just like if it were sent from any regular JavaScript call.

Structures and Arrays will function exactly the same way. Structure elements will maintain their case, unlike receiving them in Flash Remoting where all elements become upper-case. Arrays of one or more dimensions will be treated just like a regular JavaScript array as well.

Queries are extremely easy to receive too. There are received by JavaScript as a structure of arrays, not as an array of structures. The correct way of evaluating a field is result.columName[row], not ~~result[row].columnName~~. To help you loop through your data there is a built-in function to retrieve the recordcount. You can use result. getRowCount(), which will return the equivalent to query.recordcount. For debugging, there is also another built-in function to dump the entire recordset to the screen. result.dump(); will return the content, not dump it to the screen. You can use document.write(result.dump()); to see the entire dump.

In case of doubt, I included a really handy JavaScript function called sDumper, which behaves exactly like cfdump. You can always just type sDumper(result); and it will show you exactly what is being sent back to your handler.

A different simplistic approach is to use the DWRUtil.addRows() function. I modified the original DRW Utils class to accept a serialized dataset and automatically populate a table without the need of cell functions. Simply create a table and provide it a unique ID; or optionally, create a thead and tbody as well and pass it to the addRows() function along with the dataset.

e.g.

```
function doQueryResults (r) {
        DWRUtil.removeAllRows("tableBody");
        DWRUtil.addRows("tableBody", r);
}
<table><tbody id="tableBody"></tbody> </table>
```

Enforcing Security

ajaxCFC provides essential methods to enforce protect your request calls. Two new methods are currently available: enforcing ‘get’ or ‘post’ verbs, and enforcing the check for the http referer to match your site.

Both methods will prevent people monitoring http request packets to send unauthorized requests to your server. It is recommended to use get verbs and allow blank referers for development environments, but use post and enforce referers for production environment.

The following example will force all ajax calls to be ‘posted’ and will also check the ‘http-referer’ to match the site:

```
<cfcomponent extends="ajax">

	<cfscript>
		setAllowedVerbs('POST');
		setCheckHTTPReferer(true);
	</cfscript>

	<cffunction name="echo" output="no" access="private">
		<cfargument name="str" required="Yes" type="string">

		<cfreturn arguments.str/>
	</cffunction>

</cfcomponent>
```

setAllowedVerbs takes for arguments, ‘get’, ‘post’, and ‘get,post’ to allow both verbs and it is defaulted to accept both.
Note: for change your ajax request to use ‘post’ type DWREngine.setVerb('POST'); before your DWREngine._execute() command. ajaxCFC defaults to the ‘get’ verb.

Server Side Debugging

Sometimes it becomes difficult to decipher what’s happening on the server side. In an effort to help the developer, I added basic debugging ability to dump some essential information to a debug file.

The setDebugMode() method was introduced, and it is disabled by default. You can enable to debug the ajaxCFC init method, the request sent from JS, or result returned from your function. setDebugMode currently takes ‘init’, ‘result’, ‘request’ or ‘’ to disable it.

Debugging will create a ‘debug.html’ file in the same folder where your ajaxCFC resides. If your ajaxCFC component is web accessible, just open a browser and point it to the same path, but open debug.html instead.

The debug.html file contains a dump of the URL, FORM, and CGI scope, plus a custom argument that varies depending on the placement of the debug method.

The following example will enable debugging for the ajaxCFC result method.

```
<cfcomponent extends="ajax">

	<cfscript>
		setDebugMode('result');
	</cfscript>
```

```
    <cffunction name="echo" output="no" access="private">
        <cfargument name="str" required="Yes" type="string">

        <cfreturn arguments.str />
    </cffunction>

</cfcomponent>
```

Client Side Debugging

Simply add ‘debug’:true into your _ajaxConfig settings to automatically enable a not obtrusive pop-up window that will log your every ajax call.

Special Notes

- Do not implement the onRequest method in any Application.cfc file that affects .cfc files that implement AJAX requests, web services, Flash Remoting, or event gateway requests; ColdFusion MX will not execute the requests if you implement this method.
- If you are using HomeSite+ for development, do not try to use HomeSite’s browse window; AJAX calls will not work from there.
- If your application.cfc diplays html code onRequestEnd, you may use the setAbortAfterRequest(true) function to force an abort immediately after the request is completed.