ezAJAX™ Community Edition Framework Programmer's Guide v0.92

Top of the Document

Preface

ezAJAX™ is Easy to use for many reasons you will be reading about within this document. The sole purpose for ezAJAX™ to exist is to make your development efforts *Easy.*

You will soon see upon, reading this document, that you will be required to write very little code for an ezAJAX™ Application because the Framework takes care of the rest of the details for you automatically.

We are giving you more than 10,000 lines of code which is a mixture of 80% JavaScript and 20% ColdFusion MX 7. You are allowed to use all this code for as long as you have a valid Runtime License and you are able to redistribute this code along with the code you write for your ezAJAX™ Application.

How long would it take you to write 10,000 lines of debugged JavaScript and ColdFusion MX 7 code ? We think you will save time and money by using our Framework. We think you will save far more money by using our Framework than the relatively small fee we ask for our Runtime Licenses.

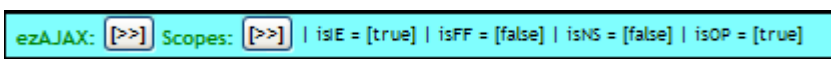
As this product matures, the developers of this Framework, will from time to time add *abstractions* to the product to allow more and more tasks to be handled automatically. The goal of all this is to create more of an Application Generator than a simple Application Framework. To that end you may begin using the ezAJAX™ Framework because your development efforts will be greatly reduced and your productivity will be greatly enhanced.

Debugging Support

Floating Debug Menu

The "floating" Debug Menu is perhaps one of the more useful features found in the ezAJAX™ Framework.

The "floating" Debug Menu typically appears at the top of the browser's client window and looks as follows:



The "floating" Debug Menu will extend from the left side of the browser's client area to the right side.

You can change the appearance of the "floating" Debug Menu as well as cause it to "float" or not by modifying the styles for the element that has the id of "const_div_floating_debug_menu". Consider the following code fragment:

```
var dObj = _$(const_div_floating_debug_menu);
if (!!dObj) {
	dObj.style.backgroundColor = 'lime';
	dObj.style.width = '500px';
}
```

You can place this code fragment within the body of the "oAJAXEngine.createAJAXEngineCallback" function that can be placed in any ColdFusion source file as specified in the file named "cfinclude_application.cfm" using the ColdFusion variable "Request.cfincludeCFM" which is a comma delimited list of ColdFusion source files that are automatically loaded at runtime using the <cfinclude> tag, one per file name. Take a look at the following code fragment:

```
function adjustFloatingMenuStyles() {
	var dObj =  $(const div floating debug menu);
	if (!!dObj) {
		dObj.style.backgroundColor = 'lime';
		dObj.style.width = '500px';
	}
}

oAJAXEngine.createAJAXEngineCallback = function () { adjustFloatingMenuStyles(); this.top
= '400px'; };
```

As you may notice upon placing this code in the appropriate place within the Framework the "floating" Debug Menu assumes a "lime" background color rather than the default "cyan" color as shipped with the Framework out-of-the-box.

You can place this code fragment within the body of the "oAJAXEngine.showAJAXDebugMenuCallback" function. Take a look at the following code fragment:

```
function adjustFloatingMenuStyles() {
	var dObj = _$(const_div_floating_debug_menu);
	if (!!dObj) {
		dObj.style.backgroundColor = 'lime';
		dObj.style.width = '500px';
	}
}

oAJAXEngine.showAJAXDebugMenuCallback = function () { adjustFloatingMenuStyles(); return
true; };
```

You can place this code fragment within the body of the "oAJAXEngine.showAJAXScopesMenuCallback" function. Take a look at the following code fragment:

```
function adjustFloatingMenuStyles() {
	var dObj = _$(const_div_floating_debug_menu);
	if (!!dObj) {
		dObj.style.backgroundColor = 'lime';
		dObj.style.width = '500px';
	}
}

oAJAXEngine.showAJAXScopesMenuCallback = function () { adjustFloatingMenuStyles(); return
true; };
```

You can place this code fragment within the body of the "oAJAXEngine.showAJAXBrowserDebugCallback" function. Take a look at the following code fragment:

```
function adjustFloatingMenuStyles() {
	var dObj =  $(const div floating debug menu);
	if (!!dObj) {
		dObj.style.backgroundColor = 'lime';
		dObj.style.width = '500px';
	}
}

oAJAXEngine.showAJAXBrowserDebugCallback = function () { adjustFloatingMenuStyles();
return true; };
```

As you can see there can be many ways to achieve the same effect by using the API from the ezAJAX™ Framework. All three of the preceding Call-Back functions are executed within the same scope and therefore all three could be used to make changes to the "floating" Debug Menu's styles.

Replace the Floating Debug Menu

You can also replace the default "floating" Debug Menu with whatever content you may wish by using the following code fragment:

```
function replaceFloatingMenuContent() {
	var dObj = _$(const_div_floating_debug_menu_content);
	if (!!dObj) {
		dObj.innerHTML = '<b>New Menu Content</b>';
	}
}

oAJAXEngine.createAJAXEngineCallback = function () { replaceFloatingDebugMenu(); this.top
= '400px'; };
```

As you can see you now have the ability to create your own horizontal Site Menu by writing very little code to modify some already existing elements that are part of the ezAJAX™ Framework.

Add to the Floating Debug Menu

You can also add content to the default "floating" Debug Menu by using the following code fragment:

```
function populateExtraMenuContainer() {
        var dObj = _$('div_extraContainer');
        if (!!dObj) {
                dObj.innerHTML = ' <b>Extra Menu Content</b>';
        }
}

oAJAXEngine.createAJAXEngineCallback = function () { populateExtraMenuContainer();
this.top = '400px'; };
```

This code fragment will use the existing default "floating" Debug Menu and add some customized content to it.

It would be quite easy to dynamically program your ezAJAX™ Application to remove the Debug Menu content in your Production App but keep it for your Development App.

Remove or Hide the Floating Debug Menu

You can also remove or hide the default "floating" Debug Menu by using the following code fragment:

```
var dObj = $(const div floating debug menu);
if (!!dObj) {
        dObj.style.display = const none style;
}
```

This code fragment when placed within the body of the "ezWindowOnscrollCallback" function will cause the default "floating" Debug Menu to be hidden from view.

Recap the Floating Debug Menu

As you have seen the default "floating" Debug Menu can be modified, added to, removed from and hidden using very little code. Now it is up to you, the programmer, to choose how you wish to use the "floating" Debug Menu or not.

ezappname.cfm

The file "ezappname.cfm", if it exists, can be used to define the name of the Application as well as the <cfapplication> tag. The default behavior is for the Framework to create a <cfapplication> tag using an abstract Application Name that is based on the CGI.Script_NAME for the specific instance of the Framework.

If you create a file called "ezappname.cfm" you should place it in the root folder where the application.cfm file is located for the Framework and it must define a <cfapplication> tag using whatever options are desired.

The ezAJAX_title variable

The "ezAJAX_title" variable can be defined to reside within the appname.cfm file to allow it to be defined with whatever name as may be desired. This is the name that appears on the browser's title bar at runtime.

Top of the Document

The Default <cfapplication> tag

The default <cfapplication> tag uses the following syntax:

```
<cfapplication name="#myAppName#" clientmanagement="Yes" sessionmanagement="Yes"
clientstorage="clientvars" setclientcookies="No" setdomaincookies="No"
scriptprotect="All" sessiontimeout="#CreateTimeSpan(0,1,0,0)#"
applicationtimeout="#CreateTimeSpan(1,0,0,0)#" loginstorage="Session">
```

You may choose to use the default or make whatever changes may be desired to achieve the goals of your specific instance of an ezAJAX™ Application.

The ezAJAX_webRoot variable

The ezAJAX_webRoot variable is used to store the webRoot for the current application instance.

Let's say you want to place the ezAJAX™ Community Edition Framework in a subfolder called "myAJAX" which is immediately located beneath the webroot for your web server. The ezAJAX_webRoot variable will contain the following string at runtime: "http://your-domain-name/myAJAX/".

The ezAJAX_webRoot variable takes the CGI.SCRIPT_NAME and removes the last element that is delimited by the "/" character. This allows an ezAJAX™ Application to be placed in any folder structure as may be desired with a common method for determining the webroot for any URL one may wish to create at runtime.

This becomes a useful construct whenever the need arises to move an ezAJAX™ Application from one folder to another or from one web server to another. If one refers to the webRoot using the variable "ezAJAX_webRoot" one soon finds that one need not recode any references to those URLs that depend on the folder path that is used to form URLs at runtime.

The Request.ezAJAX_isDebugMode variable

The "Request.ezAJAX_isDebugMode" variable is used to determine when your ezAJAX™ Application is running in Debug Mode.

You may already know there is a method for making this determination using the ColdFusion API, isDebugMode(), however from time to time it is necessary to force the issue so that applications once deployed to Production can be debugged based on certain criteria such as the IP Address of the user rather than just the presence of a Debugging setting at runtime.

Top of the Document

The ezAJAX_getDebugMode Custom Function

You may define a custom function within your appname.cfm file to help you determine when your ezAJAX™ Application is running in Debug Mode.

Here is a suggestion for how your "ezAJAX_getDebugMode" custom function may be written:

```
<cfscript>
	function ezAJAX_getDebugMode(ipAddress) {
		return ( (FindNoCase('192.168.1.', ipAddress) gt 0) OR
(FindNoCase('127.0.0.1', ipAddress) gt 0) );
}
</cfscript>
```

At runtime the "ezAJAX_getDebugMode" custom function will be passed the value from the CGI.REMOTE_ADDR variable. You may use any criteria as may be desired to determine when your instance of an ezAJAX™ Application should consider itself to be in Debug Mode. The Boolean value you return from your "ezAJAX_getDebugMode" custom function will be combined with the isDebugMode() value using the "OR" operator. If any errors occur while your "ezAJAX_getDebugMode" custom function is executing the default behavior is to simply use the isDebugMode() value as the only means to determine when your ezAJAX™ Application is in Debug Mode.

You may ignore this mechanism and define your own if you wish however this mechanism has been provided for you in case you wish to use it.

The Request.ezAJAX_Cr Constant

The "Request.ezAJAX_Cr" constant has the value of CHR(13) assigned to it.

From time to time one may deploy code to an OS that requires a different value for Carriage Return than a simple CHR(13) and when this happens one may wish to use a constant value that can be more easily changed at runtime than having to track-

down every instance of a CHR(13) which may need to be made into a CHR(13) & CHR(10) for instance.

The Request.ezAJAX_Lf Constant

The "Request.ezAJAX_Lf" constant has the value of CHR(10) assigned to it.

The Request.ezAJAX_CrLf Constant

The "Request.ezAJAX_CrLf" constant has the value of CHR(13) & CHR(10) assigned to it.

Top of the Document

The Request.AUTH_USER variable

The "Request.AUTH_USER" variable is used to transmit information about the currently logged-in user to the ezAJAXEngine which is then transmitted to the ezAJAX Server.

Some user authentication systems will set the "CGI.AUTH_USER" variable with the currently logged-in user name. This value is stored in the "Request.AUTH_USER" which is then stored in a JavaScript variable called "js_AUTH_USER" which is later passed along to the ezAJAXEngine via the URL parameter called "AUTH_USER", more about this later on in this document.

For now simply be aware that the "Request.AUTH_USER" variable is being used when it exists. The type of user authentication systems that use this variable are those that use the NT Authentication model such as some LDAP based user authentication systems. It is not the purview of this product to define the user authentication systems it may interface with but rather to define the means to allow any suitable user authentication system to be used.

It left up to the programmer to determine best how to create or interface with a usable user authentication system. In some cases it is easier to use the Session Scope to store information about the currently logged-in user while it other cases it is easier to use a database to store such data. In either case a programmer may choose to use the "Request.AUTH_USER" variable to convey the name of the currently logged-in user through the ezAJAXEngine to the ezAJAX Server.

The "err_ajaxCode" and "err_ajaxCodeMsg" variables

The "err_ajaxCode" and "err_ajaxCodeMsg" variables store the state of the ezAjaxCode component object. If the ezAjaxCode component has been properly created the "err_ajaxCode" variable will have the value false and the "err_ajaxCodeMsg" variable will have the value of an empty string. If there were

any errors handled during the creation of the ezAjaxCode component object then the "err_ajaxCode" variable will have the value true and the "err_ajaxCodeMsg" variable will have the value of an error message that indicates the nature of the error. The error message will also be displayed in the browser to tell the end-user that something went wrong.

Top of the Document

The "Request.ezAJAX_functions_cfm" variable

The "Request.ezAJAX_functions_cfm" variable stores the value that describes the location (URL) of the ColdFusion file that processes ezAJAX™ commands. You may notice the ColdFusion file known as "ezAJAX_functions.cfm" which was delivered in an encrypted format. You may also notice the ColdFusion file known as "userDefinedAJAXFunctions.cfc" which is not encrypted. You may place your exAJAX™ Server code in the "userDefinedAJAXFunctions.cfc" file using the format you see already expressed within that file. You can read more about this later on in this document.

The "application.isColdFusionMX7" variable

The "application.isColdFusionMX7" variable stores a Boolean value that indicates whether or not ColdFusion MX 7 is being used. It should be noted that ezAJAX™ works only with ColdFusion MX 7 and has been successfully tested with ColdFusion MX 7.0.2.

It would be quite easy to develop an intelligent Proxy process using some other CGI Language such as PHP or ASP.Net or the like that would allow any version of ColdFusion MX 7 to be used as an ezAJAX™ Server. This would allow the Developer's Edition of ColdFusion MX 7 to be used in a Production environment, if necessary, and although this might cause some difficulties in relation to the License Agreements published by Adobe/Macromedia as to the proper usage of the Developer's Edition of ColdFusion MX 7 the technology could easily exist to allow this to be done with little effort. ezAJAX™ makes it quite easy to leverage the power of ColdFusion MX 7 using techniques that are both simple as complex at the same time.

The "userDefined_application.cfm" file

The "userDefined_application.cfm" ColdFusion file is not encrypted to allow the programmer to place user-defined code within the scope of the Application.cfm file which has been encrypted. The "userDefined_application.cfm" file is executed at the bottom of the application.cfm file.

Some of our readers may be wondering why we chose to use the application.cfm model rather than the more robust application.cfc model. This choice was made to allow the Community Edition to serve as an entry-point ezAJAX™ product whereas

the Enterprise Edition will use the more robust application.cfc model with better error handling and recovery built-in in addition to greater optimizations.

Top of the Document

The "Request.cfincludeCFM" variable

The "Request.cfincludeCFM" variable stores a list of ColdFusion files that are automatically included via <cfinclude> within the scope of the index.cfm file. We have provided you with a sample ColdFusion file called "cfinclude_index_body.cfm" that contains application specific code. You may choose to use the "cfinclude_index_body.cfm" file as desired or create any suitable file using whatever naming convention you may desire to use for this purpose.

The "Request.DOCTYPE" variable

The "Request.DOCTYPE" variable stores the DOCTYPE you wish to use instead of the default which is '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">'. ezAJAX™ will use the default DOCTYPE whenever there is the "Request.DOCTYPE" variable is missing or is not defined or has an empty string value.

ezCompiler™

ezCompiler™ is a JavaScript Compiler that not only obfuscates JavaScript files but it also selectively encrypts specific JavaScript files while encoding the rest to make it a bit more difficult for curious eyes to get a peek at your valuable intellectual property.

ezCompiler™ uses a specific folder that contains individual JavaScript files that should each be quite small since the techniques employed by ezCompiler™ work better when it is given small JavaScript files to work on. In real terms each JavaScript file should be less than 8k bytes.

Each of the JavaScript files is also named using a technique that allows ezCompiler™ to understand the dependencies between files. The first two characters of a JavaScript file serve as a filter that allows ezCompiler™ to load the least dependent files first followed by those that depend on the earlier files. For instance, 00_constants.js would be processed before 01_moreCode.js and so on until all the available files have been processed by ezCompiler™.

ezCompiler™ only executes when it is required to do so based on the date/time of each JavaScript file and the resulting file which is known as "javascript.js". Whenever any of the source files are newer than the "javascript.js" file ezCompiler™ executes to produce a new "javascript.js" file from the available source files.

When deploying your ezAJAX™ Application you would deploy only the "javascript.js" file rather than the source JavaScript files – this allows you to keep your source files under wraps by exposing only your compiled "javascript.js" file to end-users.

ezCompiler™ is available as an add-on to ezAJAX™ for a nominal License Fee.

Top of the Document

index.CFM

StyleSheet.css

The StyleSheet.css file contains a set of CSS definitions that may prove useful to those who wish to develop ezAJAX™ Applications. You s the programmer may choose to use these definitions or create your own; you may of course place your own CSS definitions in the StyleSheet.css file using whatever methods you may choose to use.

The "htmlHeader" variable

The "htmlHeader" variable stores additional HTML head elements as may be desired for the specific application you are coding.

The "javascript.js" file

The "javascript.js" file that was shipped with the ezAJAX™ Community Edition Framework contains the core ezAJAX™ client code that allows ezAJAX™ Application to be created and deployed.

You may feel free to deploy the ezAJAX™ Community Edition Framework with your application specific code including whatever modifications or additions you made using those unencrypted source files we made available to you. Your end-users will of course be required to obtain Runtime License files from you or us depending on how you choose to do this.

ezLicenser™

ezLicenser™ is another add-on for ezAJAX™ we can license to you for your use. ezLicenser™ allows you to sell Runtime Licenses to your end-users assuming you plan on selling Licenses for your ezAJAX™ Application. There would be no need to use ezLicenser™ if you plan on deploying your ezAJAX™ Application via your web server because you would be using your own Runtime License when running ezAJAX™ Application on your own web server.

JavaScript Objects

ezAnchorPosition

The "ezAnchorPosition" Object performs the function of determining the browser coordinates for an anchor HTML element that has both an "id" and "name" both of which must have the same element identifier.

Top of the Document

Constructor Method

ezAnchorPosition.get$ (anchorname)

The "ezAnchorPosition.get$ (anchorname)" function takes one argument that is the anchor element's identifier or name. The value that is returned is an ezAnchorPosition Object instance.

All Objects that are defined to be part of the ezAJAX™ Community Edition Framework share some common architectural similarities such as the following:

- Constructor Method
- Destructor Method
- Object instance cache
- Object instances Destructor Method
- Class variables (where applicable)
- Instance variables (where applicable)
- Class Methods (where applicable)
- Instance Methods (where applicable)

Object Instance Cache

The Object Instance Cache for the "ezAnchorPosition" object is stored in the variable "ezAnchorPosition.$" which is an Array Object instance that holds all the instances of ezAnchorPosition Objects that are created at run-time. This makes it easy to perform the appropriate clean-up functions before closing the browser whenever the window.onUnload event is fired. The technique of caching all the instances of certain Objects also aids in developing applications that use these Objects because if one can access certain Object instances at run-time one can leverage this ability to make certain development tasks easier.

Destructor Method

ezAnchorPosition.remove$(id)

The "ezAnchorPosition.remove$(id)" function takes one argument which is the "id" of the ezAnchorPosition Object instance to be removed.

Object Instances Destructor Method

Top of the Document

ezAnchorPosition.remove$s()

The "ezAnchorPosition.remove$s()" function takes no arguments and performs the function of removing all Instances of the ezAnchorPosition Object. This function is automatically called whenever the window.onUnload event is fired for ezAJAX™ Community Edition Applications.

Instance variables

The "id" instance variable stores the instance identifier for each object instance; this is also the index within the object's instance cache for this specific object instance.

The "x" instance variable stores the X coordinate for the client coordinate of the specific named anchor element that is identified by the "anchorname" instance variable.

The "y" instance variable stores the Y coordinate for the client coordinate of the specific named anchor element that is identified by the "anchorname" instance variable.

The "anchorname" instance variable stores the name of the anchor element for which X and Y coordinates are being retrieved.

The "use_gebi" instance variable stores a Boolean value that is either "true" or "false" depending on whether the browser responds to the "document.getElementById" function. This instance variable is used internally by the ezAnchorPosition Object to perform the processing it is required to perform.

The "use_css" instance variable stores a Boolean value that is either "true" or "false" depending on whether the browser does not respond to the "document.getElementById" but does respond to the "document.all" function. This instance variable is used internally by the ezAnchorPosition Object to perform the processing it is required to perform.

The "use_layers" instance variable stores a Boolean value that is either "true" or "false" depending on whether the browser does not respond to the "document.getElementById" and does not respond to the "document.all" and does respond to the "document.layers" function. This instance variable is used internally by the ezAnchorPosition Object to perform the processing it is required to perform.

Top of the Document

Instance methods

toString()

The "toString()" instance method provides an easy way to debug the contents of an Object instance via the "alert()" function or the "_alert()" function. Whenever an Object instance is printed via some type of writeln() method the toString() method for each object is fired to obtain a String representation of the object.

getAnchorPosition(anchorname)

The "getAnchorPosition(anchorname)" instance method takes one argument which is the name of the anchor element and sets the "x" and "y" instance variables with the browser client coordinates for the specific named anchor element. This instance method is automatically fired whenever an Instance of the ezAnchorPosition Object is created using the aforementioned constructor method.

ezDictObj

The "ezDictObj" Object performs the function of providing an Abstract Dictionary Object that essentially stores key-value pairs via a technique that provides more power and flexibility than what normally may be expected.

Constructor Method

ezDictObj.get$(aSpec)

The "ezDictObj.get$(aSpec)" function takes one argument that is "aSpec" String Object instance that specifies a standard "Query String" that names key-value pairs in the form of the following: "key1=value1&key2=value2…" or "key1=value1,key2=value2…". The value that is returned is an ezDictObj Object instance.

All Objects that are defined to be part of the ezAJAX™ Community Edition Framework share some common architectural similarities such as the following:

- Constructor Method
- Destructor Method

- Object instance cache
- Object instances Destructor Method
- Class variables (where applicable)
- Instance variables (where applicable)
- Class Methods (where applicable)
- Instance Methods (where applicable)

Top of the Document

Object Instance Cache

The Object Instance Cache for the "ezDictObj" object is stored in the variable "ezDictObj.$" which is an Array Object instance that holds all the instances of ezDictObj Objects that are created at run-time. This makes it easy to perform the appropriate clean-up functions before closing the browser whenever the window.onUnload event is fired. The technique of caching all the instances of certain Objects also aids in developing applications that use these Objects because if one can access certain Object instances at run-time one can leverage this ability to make certain development tasks easier.

Destructor Method

ezDictObj.remove$(id)

The "ezDictObj.remove$(id)" function takes one argument which is the "id" of the ezDictObj Object instance to be removed.

Object Instances Destructor Method

ezDictObj.remove$s()

The "ezDictObj.remove$s()" function takes no arguments and performs the function of removing all Instances of the ezDictObj Object. This function is automatically called whenever the window.onUnload event is fired for ezAJAX™ Community Edition Applications.

Instance variables

The "id" instance variable stores the instance identifier for each object instance; this is also the index within the object's instance cache for this specific object instance.

The "bool_returnArray" instance variable stores a Boolean value which when "true" causes the return values from the "getValueFor(aKey)" method to always return Array Object instances rather than the default behavior which is to only return Array Object instances when more than one value satisfies the "getValueFor(aKey)" method invocation.

The "keys" instance variable stores the Array Object instance that holds the key values.

Top of the Document

Instance methods

toString()

The "toString()" instance method provides an easy way to debug the contents of an Object instance via the "alert()" function or the "ezAlert()" function. Whenever an Object instance is printed via some type of writeln() method the toString() method for each object is fired to obtain a String representation of the object.

fromSpec(aSpec)

The "fromSpec(aSpec)" instance method takes one argument which is "aSpec" String Object instance that specifies a standard "Query String" that names key-value pairs in the form of the following: "key1=value1&key2=value2…" or "key1=value1,key2=value2…".

URLDecode()

The "URLDecode()" instance method takes no arguments and performs the function of sending a "ezURLDecode()" message to all the values stored within the ezDictObj.

asQueryString(ch_delim)

The "asQueryString(ch_delim)" instance method takes one argument that is "ch_delim" String Object instance that specifies a single character delimiter that is by default the "," character.

push(key, value)

The "push(key, value)" instance method takes two arguments that are "key" String Object instance that specifies a key that is associated with "value" String Object instance that specifies a value. This method works in a similar manner as the "push(value)" method for the Array Object instance in that new values are pushed into an ezDictObj using this method. Many values can be associated with a single "key" which means whenever a "key" for which there are many values stores is requested the Array Object instance that holds the values is returned rather than a single String Object instance unless the appropriate Boolean flag has been set to always return an Array Object instance.

Top of the Document

put(key, value)

The "put(key, value)" instance method takes two arguments that are "key" String Object instance that specifies a key that is associated with "value" String Object instance that specifies a value. This method is used to replace an already extant "value" for a new "value" for a specific "key"; all other uses will throw an error by popping-up an "alert()" dialog.

drop(key)

The "drop(key)" instance method takes one argument that is "key" String Object instance that specifies a key that is to be dropped or removed from the ezDictObj along with the associated value(s).

getValueFor(key)

The "getValueFor(key)" instance method takes one argument that is "key" String Object instance that specifies a key for which the associated value(s) are to be retrieved from the ezDictObj. If the "bool_returnArray" Boolean has been set "true" then this method always returns an Array Object instance even when there is only one value to return.

getKeysMatching(aFunc)

The "getKeysMatching(aFunc)" instance method takes one argument that is "aFunc" Function Object instance that specifies a Function that takes two arguments which are the "key, value" for each key-value pair. If "aFunc" returns "true" then the "key, value" pair is returned in the form of an Array Object instance that holds all the "key" String Object instances that caused "aFunc" to return a "true" value. Keep in mind the fact that value(s) can be Array Object instances or String Object instances unless the "bool_returnArray" Boolean has been set "true" in which case the value(s) are always Array Object instances. This method allows the programmer to construct simple Query of ezDictObj's contents which can extend the power of the ezAJAX™ Community Edition Framework considerably when properly used.

getKeys()

The "getKeys()" instance method takes no arguments and returns an Array Object instance that holds the array of "key" String Object instances that specify each "key" that is associated with a "value" within the ezDictObj instance.

Top of the Document

adjustKeyNames(aFunc)

The "adjustKeyNames(aFunc)" instance method takes one argument that is "aFunc" Function Object instance that specifies a Function that takes one argument that is the "key" from the array of keys. The user-defined "aFunc" should return the "key" with some kind of transformation applied to each "key". This method does not modify the keys within the ezDictObj but it does allow the array of keys to be modified in a temporary manner as-needed. This method allows the programmer to construct simple Query of ezDictObj's contents which can extend the power of the ezAJAX™ Community Edition Framework considerably when properly used.

length()

The "length()" instance method takes no arguments and returns the number of keys that are stored within the ezDictObj instance.

keyForLargestValue()

The "keyForLargestValue()" instance method takes no arguments and returns the greatest number from all those values that are associated with "key" Strings. This method assumes there is one and only one value for each key and that the value is already numeric and represents a 32 bit value at most.

intoNamedArgs()

The "intoNamedArgs()" instance method takes no arguments and converts an ezDictObj instance that contains arguments from the ezAJAX™ Server Call-Back into the same ezDictObj instance that holds "named" arguments since the first "arg" from ezAJAX™ Server is the name of the argument and the next "arg" is the value for the argument. This method is used automatically whenever the simpler ezAJAX™ Server Call-Back is being used otherwise the programmer is responsible for taking care of this assuming this is a desired action to perform on behalf of the programmer.

init()

The "init()" instance method takes no arguments and performs the action of initializing the keys Array and the values cache within the ezDictObj instance. Any previously existing "key, value" pairs will be lost when this method completes.

Top of the Document

ezAJAXEngine Behaviors

register_ezAJAX_function[1]

One of the more powerful features of ezAJAX™ is the ability to queue-up specific units of processing which can be Server Commands or any other JavaScript statement(s).

Use the "oAJAXEngine.register_ezAJAX_function(s)" function to queue-up a unit of processing in the form of a String object instance that contains one or more valid JavaScript statement(s).

The ezAJAXEngine will attempt to process all the available queued-up processing units that are present whenever a block of data is received by the ezAJAXEngine.

This allows the programmer to do such things as simulate a series of client-side actions that are to be performed whenever the next block of data is received by the ezAJAXEngine. This can be quite useful when a series of client-side actions are to be performed as soon as the ezAJAXEngine receives the first block of data from the ezAJAX™ Server to kick-start some kind of processing or to enable GUI elements or some other type of processing.

This feature could also be used to queue-up ezAJAX™ Server Commands that may need to be queued-up while the Server is busy processing a prior Server Command.

The programmer could issue the "oAJAXEngine.isIdle()" function call to determine if the ezAJAXEngine instance is busy processing a prior Server Command. This could be done whenever a Server Command is about to be sent to the server in an asynchronous manner. This level of programmer would be considered quite advanced as most programmers will probably simply disable those GUI functions that cannot be performed while the ezAJAX™ Server is busy. In fact the programmer could simply queue-up function calls that enable GUI elements that need to be disabled while the ezAJAX™ Server is busy processing a Server Command.

[1] This feature was added in version 0.91.

Improved Error Handling in Version 0.91

The ezAJAXEngine Object instances will automatically react in a more robust manner whenever an error occurs in a Server Command Call-Back function.

Previously the behavior was to display a pop-up Error panel that did identify the error but did not specify the specific Call-Back that generated the error.

This level of error handling is only activated whenever a previously queued-up unit of processing encounters a JavaScript Error of some kind.

JavaScript Object Instances

oAJAXEngine

oAJAXEngine is a JavaScript Object of type ezAJAXEngine. oAJAXEngine is the default primary instance of the ezAJAXEngine Object which is used to coordinate and control the AJAX communications between the client and the server. This object will be covered in more detail later on in this document.

JavaScript Variables

const_inline_style

The "const_inline_style" variable stores the value of 'inline'. This value is used to set the display style to inline to allow an HTML element to be shown rather than be hidden.

const_none_style

The "const_none_style" variable stores the value of 'none'. This value is used to set the display style to none to allow an HTML element to be hidden rather than be shown.

const_absolute_style

The "const_absolute_style" variable stores the value of 'absolute'. This value is used to set the position style to absolute to allow an HTML element to be dynamically positioned.

const_function_symbol

The "const_function_symbol" variable stores the value of 'function'. This value is used to test the variable type of a JavaScript variable to determine if the variable holds a pointer to a function.

const_object_symbol

The "const_object_symbol" variable stores the value of 'object'. This value is used to test the variable type of a JavaScript variable to determine if the variable holds a pointer to an object.

Top of the Document

const_number_symbol

The "const_number_symbol" variable stores the value of 'number'. This value is used to test the variable type of a JavaScript variable to determine if the variable holds a pointer to a number.

const_string_symbol

The "const_string_symbol" variable stores the value of 'string'. This value is used to test the variable type of a JavaScript variable to determine if the variable holds a pointer to a string.

const_simpler_symbol

The "const_simpler_symbol" variable stores the value of 'simpler'. This value is used to signal to the ezAJAXEngine that the "simpler" method of handling Call-Backs should be used rather than the more complex method. The "simpler" method provides for automatic handling or error conditions as well as the automatic handling of arguments that were passed to the AJAX™ Server.

jsBool_isColdFusionMX7

The "jsBool_isColdFusionMX7" variable stores the value from the ColdFusion variable called "application.isColdFusionMX7".

jsBool_isDebugMode

The "jsBool_isDebugMode" variable stores the value from the ColdFusion variable called "Request.ezAJAX_isDebugMode".

jsBool_isServerLocal

The "jsBool_isServerLocal" variable stores the value from the ColdFusion function return value from "Request.commonCode.isServerLocal()".

fqServerName

The "fqServerName" variable stores the return value from the JavaScript function fullyQualifiedAppPrefix() which is used to determine the fully qualified application prefix which is the same value as the ColdFusion variable called "ezAJAX_webRoot". The function fullyQualifiedAppPrefix() returns a dynamically calculated value that is derived from the appropriate JavaScript object.

Top of the Document

cfinclude_index_body.cfm

oAJAXEngine.timeout

The "oAJAXEngine.timeout" variable stores the time-out value in seconds that the ezAJAXEngine will wait after each AJAX Request has been issued. If the AJAX Server is offline or has experienced some kind of ColdFusion Error that inhibits the response from the AJAX Server this time-out value will expire and the ezAJAXEngine will fire a Call-Back function to allow the client to regain control in a more graceful manner than simply displaying a pop-up message of some kind.

oAJAXEngine.showFrameCallback

The oAJAXEngine.showFrameCallback is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to show the hidden <iFrame> to allow debugging to be performed on the AJAX Server. Normally the hidden <iFrame> is hidden from view however it can be quite beneficial to debug the AJAX Server's activity via the <iFrame> once it is visible.

oAJAXEngine.hideFrameCallback

The oAJAXEngine.hideFrameCallback is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to hide the <iFrame> to allow debugging activities to be closed or ceased.

oAJAXEngine.createAJAXEngineCallback

The "oAJAXEngine.createAJAXEngineCallback" is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to "create". The "create" action must be accomplished before the ezAJAXEngine will begin operating. It may be necessary to make some adjustments to the ezAJAXEngine instance when the "create" action is performed such as you see in the sample code found in the cfinclude_index_body.cfm ColdFusion file. This Call-Back allows the required adjustments to be performed as-needed.

oAJAXEngine.showAJAXBeginsHrefCallback[2]

The "oAJAXEngine.showAJAXBeginsHrefCallback" is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to display the Server Busy pop-up window that appears in the upper right corner of the client's browser window. The purpose for this Call-Back is to allow the programmer to override the URL the system uses when specifying the location of the "ezAjaxStyles.css" file. The programmer may code a Call-Back that returns a String object instance that contains the fully qualified URL that points to the folder in which the "ezAjaxStyles.css" file resides. This Call-Back is useful whenever a custom URL is used to access the application code because the browser simply inherits the custom URL which may not be useful when forming ancillary URLs such as the one(s) that are used by the Server Busy indicator panel.

oAJAXEngine.showAJAXDebugMenuCallback[3]

The "oAJAXEngine.showAJAXDebugMenuCallback" is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to "create". The purpose of the "oAJAXEngine.showAJAXDebugMenuCallback" is to allow the programmer to determine whether or not the "ezAJAX:" Debug Menu will appear on the floating Menu Bar that can be made to either "float" as the browser window scrolls or remain stationary near the top of the browser window. When the "oAJAXEngine.showAJAXDebugMenuCallback" returns "false" the "ezAJAX:" Debug Menu will not appear otherwise it will appear. This can be useful when a programmer wants to be able to perform certain debugging functions using a development installation of ezAJAX™ versus a production release of an ezAJAX™ at which time the "ezAJAX:" Debug Menu should not be shown. It is left in the hands of the programmer to determine when the "ezAJAX:" Debug Menu is to be shown or not.

---

[2] This Call-Back was added to version 0.91.

[3] This Call-Back was added to version 0.92.

showAJAXScopesMenuCallback[4]

The "oAJAXEngine.showAJAXScopesMenuCallback" is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to "create". The purpose of the "oAJAXEngine. showAJAXScopesMenuCallback" is to allow the programmer to determine whether or not the "Scopes:" Debug Menu will appear on the floating Menu Bar that can be made to either "float" as the browser window scrolls or remain stationary near the top of the browser window. When the "oAJAXEngine. showAJAXScopesMenuCallback" returns "false" the "Scopes:" Debug Menu will not appear otherwise it will appear. This can be useful when a programmer wants to be able to perform certain debugging functions using a development installation of ezAJAX™ versus a production release of an ezAJAX™ at which time the "Scopes:" Debug Menu should not be shown. It is left in the hands of the programmer to determine when the "Scopes:" Debug Menu is to be shown or not.

ezWindowOnLoadCallback

The "ezWindowOnLoadCallback" is a Call-Back function that fires whenever the window.onLoad event is fired. We have implemented the event handler as part of this Framework and we have exposed the Call-Back function to allow programmers to implement their own window.onLoad event handler function as-needed.

Top of the Document

ezWindowOnUnloadCallback

The "ezWindowOnUnloadCallback" is a Call-Back function that fires whenever the window.onUnload event is fired. We have implemented the event handler as part of this Framework and we have exposed the Call-Back function to allow programmers to implement their own window.onUnload event handler function as-needed.

ezWindowOnReSizeCallback

The "ezWindowOnReSizeCallback" is a Call-Back function that fires whenever the window.onResize event is fired. We have implemented the event handler as part of this Framework and we have exposed the Call-Back function to allow programmers to implement their own window.onResize event handler function as-needed.

---

[4] This Call-Back was added to version 0.92.

ezWindowOnscrollCallback

The "ezWindowOnscrollCallback" is a Call-Back function that fires whenever the window.onScroll event is fired. We have implemented the event handler as part of this Framework and we have exposed the Call-Back function to allow programmers to implement their own window.onScroll event handler function as-needed.

handleSampleAJAXCommand

The "handleSampleAJAXCommand" is a Call-Back function that fires whenever the ezAJAXEngine completes an AJAX command and returns control back to the client. We have implemented this function as part of this Framework to demonstrate how easy it is to code a function like it. Programmers will want to copy the same structure for each ezAJAX™ Command Call-Back function coded. Notice the rich architecture of the ezAJAX™ Command Call-Back function and the weight of the data being sent back to the client. ezAJAX™ provides a robust Error handling mechanism that allows AJAX Server Errors to be handled in a graceful manner that allows either text or HTML error messages to be sent back to the client with automatic handling of those error conditions by the Framework. If the programmer uses the same structure for each and every ezAJAX™ Command Call-Back function coded then it is necessary to provide very little new code other than that which we have provided as part of this Framework. Suitably skilled JavaScript programmers will notice the opportunity to code abstract handlers for their ezAJAX™ Command Call-Back functions all of which could easily be handled by the same ezAJAX™ Command Call-Back function or by an abstraction that can be easily coded and used repeatedly.

Top of the Document

simplerHandleSampleAJAXCommand

The "simplerHandleSampleAJAXCommand" is a Call-Back function that fires whenever the ezAJAXEngine completes an AJAX command and returns control back to the client much like the "handleSampleAJAXCommand" Call-Back function except the "simplerHandleSampleAJAXCommand" Call-Back requires much less code to perform the same functions.

It is left in the hands of the programmer to determine which method to use when coding your ezAJAXEngine Call-Back functions. You may use either however when you are using the pattern used by the "simplerHandleSampleAJAXCommand" Call-Back function you are also using the built-in Abstract Handler that determines when an error has occurred based on the returned Query Object and it handles the Error condition by popping-up a dialog box that indicates what the error was so the end-user will remain aware of what has happened.

It is possible to by-pass the built-in Abstract Call-Back Handler by coding your own function for the ezAJAXEngine.receivePacketMethod(). The ezAJAXEngine.receivePacketMethod() is a Class Method for the ezAJAXEngine Object

that by default, unless otherwise coded, tells the ezAJAXEngine to use the simpler method for handling ezAJAX™ Server Call-Backs.

To by-pass the built-in Abstract Call-Back Handler code the following method in the code you write:

```
ezAJAXEngine.receivePacketMethod = function() { return ''; }
```

We have provided you with a template for doing this in the sample code shipped with the product so you can use this technique if you wish however be aware that when the "simpler" method is disabled the result will be that the simpler handlers will of course fail.

JavaScript Functions

All functions presented herein are known to be cross-browser compatible for the following browsers only: IE 6.x, FireFox 1.5.0.4, Netscape 8.1 and Opera 9.

ezSelectionsFromObj(obj)

The "ezSelectionsFromObj(obj)" function is used to get the selections from a selection object known as the <select> element. The returned value is an array of selections in the form of the selected values or error conditions.

Top of the Document

ezUUID$()

The "ezUUID$()" function returns a value that is known to be globally unique however it is not a traditional GUID or UUID value in the ColdFusion sense or otherwise. The value returned by this function is simply globally unique in that it should not return like values regardless of how often this function is called. You may consider this to be a temporally based globally unique value.

$(id, _frame)

The "$(id, _frame)" function takes two arguments, the "id" of the element whose Object instance is to be returned and the "_frame" the object inhabits if any. If the DHTML Object inhabits the default frame then the argument "_frame" can be a "null" value or simply not specified. Because this function will be used quite often the name of the function has been truncated to simply "$" rather than something like "GetElementByID" which would work the same but have a longer name. This function caches requests for Object instances to allow future requests for the same Object instances to be accelerated. If you wish to query for Object instances without those requests being cached then use the "_$(id, _frame)" function to do so. If the intent is to use dynamically created DHTML elements then use the "_$(id, _frame)"

to query for them otherwise previously cached Object instances will be returned erroneously. Use the "flushCache$(oO)" function to clear the cache of Object instances that have been cached.

_$(id, _frame)

The "_$(id, _frame)" function takes the same two arguments as the "$(id, _frame)" function. Use the "_$(id, _frame)" function when the DHTML elements are known to be dynamically created or when caching is not desired.

ezClickRadioButton(id)

The "ezClickRadioButton(id)" function is used to process the action of clicking the radio button element which is to say the "checked = true" attribute is processed. The use of this function saves some coding effort since it is easier to use this function than to code the action each time this action is needed. The typical use of this function is at times when one wishes to cause a radio button to become checked at times when the radio button itself was not clicked by the user.

ezClientHeight()[5]

The "ezClientHeight()" function returns the height of the client area for the browser window.

Top of the Document

ezClientWidth()

The "ezClientWidth()" function returns the width of the client area for the browser window.

ezDisableAllButtonsLike (id, bool)

The "ezDisableAllButtonsLike (id, bool)" function takes an argument which is the "id" of the element. All BUTTON elements that have an "id" that contains the "id" of the argument will be disabled by this function. This function allows buttons to be given similar "id" attributes along functional lines to allow all BUTTON elements that share a similar naming convention to be disabled. Elements can be enabled by using the 2nd argument "bool" which is optional. When "bool" is false the elements are not

[5] All references to eClientHeight() were changed to ezClientHeight() to correct a known bug, in version 0.91.

disabled which means they are enabled otherwise when "bool" is not used the elements that match the criteria for this function are disabled.

ezUnHookAllEventHandlers(anObj)

The "ezUnHookAllEventHandlers(anObj)" function takes one argument which is an Object instance for a DHTML element. This function unhooks the following event handlers for the element: "Abort, AfterUpdate, BeforeUnload, BeforeUpdate, Blur, Bounce, Click, Change, DataAvailable, DataSetChanged, DataSetComplete, DblClick, DragDrop, Error, ErrorUpdate, FilterChange, Focus, Help, KeyDown, KeyPress, KeyUp, Load, MouseDown, MouseMove, MouseOut, MouseOver, MouseUp, MouseWheel, Move, ReadyStateChange, Reset, Resize, RowEnter, RowExit, Scroll, Select, SelectStart, Start, Submit, Unload". Additional event names can be added to the "const_events_list" variable if desired.

ezFullyQualifiedAppUrl()

The "ezFullyQualifiedAppUrl()" function returns the fully qualified URL where the application is running at the moment. The returned value is the window.location.href without the Query String sans the last element from the list that is delimited by the "/" character.

Top of the Document

ezFullyQualifiedAppPrefix()

The "ezFullyQualifiedAppPrefix()" function returns the same value as the "ezFullyQualifiedAppUrl()" function except that the value that follows the "http://" symbol is "clusterized" in such a manner to allow the URL to always refer to the Cluster Manager for an ezCluster™ web server cluster.

ezCluster™ is a product that was also developed by Hierarchical Applications Limited that allows web server clusters to be constructed using nothing but the web servers themselves plus at least one database server but without the need for any Networking hardware or OS support. ezCluster™ provides the lowest cost solution for web server clustering there is because it requires no extra hardware other than the web servers that comprise the cluster plus at least one database server. ezCluster™ is U.S. Patent pending at this time.

ezURLPrefixFromHref(hRef)[6]

The "ezURLPrefixFromHref(hRef)" function takes one argument that is typically the window.location.href value and returns all but the last List element as delimited by "/" characters.

ezFirstFolderAfterDomainNameFromHref(hRef)

The "ezFirstFolderAfterDomainNameFromHref(hRef)" function takes one argument that is typically the window.location.href value and returns the folder name that follows the domain name from the "hRef" argument. This function works best when the application is hosted in a folder that is not the webroot folder.

ezFilePath2HrefUsingCommonFolder(fPath, hRef, commonFolder)[7]

The "ezFilePath2HrefUsingCommonFolder(fPath, hRef, commonFolder)" function takes three arguments which are the "fPath" or file path, the "hRef" or window.location.href and the "commonFolder" which is the name of the folder that is returned from the "firstFolderAfterDomainNameFromHref(hRef)" function. This function is useful when the goal is to deploy a web app in any folder that is not the webroot for a server that has several web apps hosted on it without the need to know about or care about which specific folder the web app has been deployed into.

ezGetStyle(el, style)

The "ezGetStyle(el, style)" function takes two arguments which are the "el" element object and the "style" name returning the value of the specific "style" from the "el" element.

[6] This function was added in version 0.91.

[7] A known issue was fixed in version 0.92.

Top of the Document

ezInsertArrayItem(a,newValue,position)

The "ezInsertArrayItem(a,newValue,position)" function takes three arguments which are the "a" array object, the "newValue" and the "position" within the array and performs the action of inserting the "newValue" into the array at the specified "position".

ezInt(i)

The "ezInt(i)" function takes one argument such as a floating point numerical value and returns the integer portion of the argument's value.

ezLocateArrayItems(a, what, start)

The "ezLocateArrayItems(a, what, start)" function takes three arguments which are the "a" array object, "what" item to locate and the "start" position within the array for the search operation. The index within the array for "what" is returned or "-1" is returned whenever "what" is not found within the array object.

ezHex(ch)

The "ezHex(ch)" function takes one argument which is the "ch" character value that is converted to a hexadecimal byte value composed of two hexadecimal values.

ezColorHex(cVal)

The "ezColorHex(cVal)" function takes one argument that is the non-hexadecimal representation of an RGB color and returns the hexadecimal representation of the RGB color value.

ezObjectDestructor(oO)

The "ezObjectDestructor(oO)" function takes one argument that is the pointer to an instance of any of the custom objects that are part of this product. The ezAJAX™ Community Edition Framework employs JavaScript Object definitions that use constructor and destructor methods to enable the programmer to properly clean-up object instances before closing the application. The window.onUnload method provided by ezAJAX™ uses the "ezObjectDestructor(oO)" function to perform the required clean-up actions.

Top of the Document

ezObjectExplainer(obj)

The "ezObjectExplainer(obj)" takes one argument that is the pointer to an object instance. The value returned is a string that contains the explanation of the object's contents. This function can explain very complex objects such as those that are composed of DHTML elements to allow the programmer to determine how to use the API for these objects.

ezRemoveArrayItem(a,i)

The "ezRemoveArrayItem(a,i)" function takes two arguments that are the "a" array object pointer and the "i" index of the array item to be removed from the array. Nothing is returned from this function.

ezRemoveEmptyItemsFromArray(ar)

The "ezRemoveEmptyItemsFromArray(ar)" function takes one argument which is the "ar" array object pointer and returns the array with the empty items removed from the array. The items in the array are assumed to be String objects.

ezSetFocus(pObj)

The "ezSetFocus(pObj)" function takes one argument that is the pointer to the element to which focus is to be set. When this function concludes focus will be set to that element if focus could be set otherwise no action is taken.

ezSetStyle(aStyle, styles)

The "ezSetStyle(aStyle, styles)" function takes two arguments which are "aStyle" that is a pointer to a Style object and "styles" that is a string that describes a series of styles that can be applied to "aStyle" object. Style definitions are typically delimited by the ";" character.

ezSimulateCheckBoxClick(id)

The "ezSimulateCheckBoxClick(id)" function takes one argument that is the "id" of the check box element that is to be clicked. The action this function performs is to fire the "onclick" event handler if there is an "onclick" event handler function defined for the element that is defined by "id".

Top of the Document

String.prototype.ezClipCaselessReplace(keyword, sText)

The "ezClipCaselessReplace(keyword, sText)" function takes two arguments that are the "keyword" to be clipped out of the String and the "sText" that replaces the "keyword". This function is defined as a Prototype function for the String class which means it is necessary to send this method invocation as a message to a String object instance using the following syntax:

```
var x = '12345'.ezClipCaselessReplace('23', 'AB');
```

The "x" variable will contain the string value of '1AB45' when the code fragment above executes.

String.prototype.ezFormatForWidth(iWidth)

The "ezFormatForWidth(iWidth)" function takes one argument that is the maximum width the comma delimited string is to be formatted to be once the function concludes. If one has a comma delimited string and one wishes to display the comma delimited string on a number of lines without having to use a "-" character to split words and still maintain strings that are no longer than "iWidth" then one would want to use this function to do so. No actions are performed on the String object instance to which this method is sent as a message however the return value is the comma delimited string with "\n" characters placed where it makes sense to do so to achieve the desired goal.

String.prototype.ezIsAlpha(iLoc)

The "ezIsAlpha(iLoc)" function takes one argument that is the "iLoc" position within the string of the character that is to be tested to determine if that character is an alpha within the range of "a" through "z" inclusive.

String.prototype.ezIsNumeric(iLoc)

The "ezIsNumeric(iLoc)" function takes one argument that is the "iLoc" position within the string of the character that is to be tested to determine if that character is a numeric within the range of "0" through "9" inclusive.

String.prototype.ezIsNumeric(iLoc)

The "ezIsNumeric(iLoc)" function takes one argument that is the "iLoc" position within the string of the character that is to be tested to determine if that character is a numeric within the range of "0" through "9" inclusive.

Top of the Document

String.prototype.ezReplaceSubString(i, j, s)

The "ezReplaceSubString(i, j, s)" function takes three arguments that are the "i" position within the string for the sub-string to be replaced and "j" the end position within the string for the "s" string to replace the characters between "i" and "j".

String.prototype.ezStripCrLfs()

The "ezStripCrLfs()" function takes no arguments and strips or removes all the Carriage Return and Line Feed characters from the String object to which this message is sent.

String.prototype.ezStripHTML()

The "ezStripHTML()" function takes no arguments and strips or removes all the HTML tags from the String object to which this message is sent.

String.prototype.ezStripSpacesBy2s()

The "ezStripSpacesBy2s()" function takes no arguments and strips or removes all the pairs of spaces from the String object to which this message is sent. This function is useful for situations where runs of more than one space are to be removed at a time thus leaving runs of single spaces intact.

String.prototype.ezStripTabs(s)

The "ezStripTabs(s)" function takes one argument which is the string that is used to replace each of the Tab characters within the String object to which this message is sent.

String.prototype.ezTrim()

The "ezTrim()" function takes no arguments and strips or removes all the leading and trailing white space from the String object to which this message is sent.

ezStyle2String(aStyle)

The "ezStyle2String(aStyle)" function takes one argument which is "aStyle" and returns "aStyle" as a String object instance. This function is useful for situations where the goal is to create a String representation of the CSS styles for any DHTML element that can have CSS Styles.

Top of the Document

ez2CamelCase(sInput)

The "ez2CamelCase(sInput)" function takes one argument which is "sInput" a String object instance that contains sub-strings delimited by the "-" character and returns a String object instance that has the "-" characters removed with the character before each "-" capitalized. This function is useful for situations where the goal is to convert certain DHTML CSS Style definitions to the form that uses Camel Case rather than the format that uses the "-" character.

ezURLDecode(encoded)

The "ezURLDecode(encoded)" function takes one argument which is an "encoded" String object instance that contains URL Encoded Strings and returns the String object instance with the URL Encoded Strings replaced by the literal strings they represent. This function is 100% compatible with the ColdFusion function "URLEncodedFormat()" or any other function that performs the same or similar function.

ezURLEncode(plaintext)

The "ezURLEncode(plaintext)" function takes one argument which is a "plaintext" String object instance that contains Strings that could be URL Encoded and returns the String object instance that contains URL Encoded Strings. This function is 100% compatible with the ColdFusion function "URLDecode()" or any other function that performs the same or similar function.

ezCfString()

The "ezCfString()" function takes no arguments and returns the String object instance as a series of comma delimited characters that are delimited by the single tick mark.

ezFlushCache$(oO)

The "ezFlushCache$(oO)" function takes one argument that is the pointer to a DIV element. The typical use of DHTML elements is to place them into DIV elements and thus the reason to flush the cache would be whenever a given DIV's contents are being cleared to make way for more dynamically create DHTML elements.

Top of the Document

ezButtonLabelByObj(btnObj)

The "ezButtonLabelByObj(btnObj)" function takes one argument that is the "btnObj" pointer to a BUTTON element object instance returns the button label using a cross-browser technique that works for all four major browsers: IE 6.x, FireFox 1.5.0.4, Netscape 8.1 and Opera 9.

ezLabelButtonByObj(bObj, sLabel)

The "ezLabelButtonByObj(bObj, sLabel)" function takes two arguments which are the "bObj" pointer to a BUTTON element object instance and "sLabel" String Object instance pointer to the new label for the button and sets the BUTTON label with the "sLabel" String value. Optionally "sLabel" can be a pointer to a function in which case the old BUTTON label is passed to the function that "sLabel" points to and the return value is used to set the BUTTON label. This function of course uses a cross-browser technique that works for all four major browsers: IE 6.x, FireFox 1.5.0.4, Netscape 8.1 and Opera 9.

ezSetFocusById(id)

The "ezSetFocusById(id)" function takes one argument that is the "id" of the DHTML element to which focus is to be set. This function uses the "ezSetFocus(pObj)" function albeit for situations when the "id" of the element is known but the pointer to that element is not known.

String.prototype.ezFilterInAlpha()

The "ezFilterInAlpha()" function takes no arguments and returns a String Object instance that contains only characters that respond "true" to the "ezIsAlpha()" Boolean test.

String.prototype.ezFilterInNumeric()

The "ezFilterInNumeric()" function takes no arguments and returns a String Object instance that contains only characters that respond "true" to the "ezIsNumeric()" Boolean test.

String.prototype.ezURLDecode()

The "ezURLDecode()" function takes no arguments and returns a String Object instance that has been sent the "ezURLDecode()" message.

Top of the Document

String.prototype.ezURLEncode()

The "ezURLEncode()" function takes no arguments and returns a String Object instance that has been sent the "ezURLEncode()" message.

String.prototype.ezStripIllegalChars()

The "ezStripIllegalChars()" function takes no arguments and returns a String Object instance that has been sent the "ezURLEncode()" message. This function is a synonym for the "ezURLEncode()" function.

ezAlert(s)[8]

The "ezAlert(s)" function takes one argument that is the "s" pointer to a String Object instance that is known to not be composed of HTML elements and displays a pop-up window using DHTML that displays the word "DEBUG" at the top of a dismissable window that automatically repositions as the client browser window is scrolled. The "ezAlert(s)" function can be combined with the "ezAlertHTML(s)" function to allow non-HTML messages to be displayed with HTML messages to achieve a unique look-and-feel. For those who are familiar with the "alert(s)" function and how it works the use of the "ezAlert(s)" function will seem natural. Since it is not possible to copy-and-paste messages that are displayed using the built-in "alert(s)" function and it is possible to copy-and-paste messages that are displayed using the "ezAlert(s)" function savvy programmers will notice right away how useful it could be to use the "ezAlert(s)" function to DEBUG their code. Also it should be noted that usage of the "ezAlert(s)" function is asynchronous whereas usage of the "alert(s)" function is not asynchronous which means whenever the "alert(s)" function is used processing stops until that message is dismissed. It can be quite useful to use the "ezAlert(s)" function to display messages in a manner that allows the underlying processing to proceed until the process completes meanwhile collecting-up DEBUG messages that can help to illuminate the flow of control among other things.

[8] The following functions behavior was modified in version 0.91 to fix some known bugs and to cause the pop-up panel to be sized correctly according to the actual size of the client's width and height: ezAlert(),ezAlertHTML(), ezDispaySysMessages(), ezDispayHTMLSysMessages(), ezAlertError() and ezAlertHTMLError().

ezAlertHTML(s)

The "ezAlertHTML(s)" function takes one argument that is the "s" pointer to a String Object instance that is known to be composed of HTML elements and displays a pop-up window using DHTML that displays the word "DEBUG" at the top of a dismissable window that automatically repositions as the client browser window is scrolled. The "ezAlert(s)" function can be combined with the "ezAlertHTML(s)" function to allow non-HTML messages to be displayed with HTML messages to achieve a unique look-and-feel.

ezAlertCODE(s)[9]

The "ezAlertCODE(s)" function takes one argument that is the "s" pointer to a String Object instance that is known to be composed of source code and displays a pop-up window using DHTML that displays the title "SOURCE CODE" at the top of a dismissable window that automatically repositions as the client browser window is scrolled.

Top of the Document

ezDispaySysMessages(s, t)

The "ezDispaySysMessages(s, t)" function is the abstract function the "ezAlert(s)" function uses; the "t" argument is the String value that appears at the top of the pop-up window panel. This function is documented here for those who wish to use the "ezAlert(s)" function to display messages that are not of a "DEBUG" nature.

ezDispayHTMLSysMessages(s, t)

The "ezDispayHTMLSysMessages(s, t)" function is the abstract function the "ezAlertHTML(s)" function uses; the "t" argument is the String value that appears at the top of the pop-up window panel. This function is documented here for those who wish to use the "ezAlertHTML(s)" function to display messages that are not of a "DEBUG" nature.

---

[9] This function was added in version 0.92.

ezAlertError(s)

The "ezAlertError(s)" function takes one argument that is the "s" pointer to a String Object instance that is known to not be composed of HTML elements and displays a pop-up window using DHTML that displays the word "ERROR" with a bright red background at the top of a dismissable window that automatically repositions as the client browser window is scrolled. This function uses the "ezDispaySysMessages(s, t)" function except that the keyword "ERROR" clues the system into the fact that the window's title bar should be bright red.

ezAlertHTMLError(s)

The "ezAlertHTMLError(s)" function takes one argument that is the "s" pointer to a String Object instance that is known to be composed of HTML elements and displays a pop-up window using DHTML that displays the word "ERROR" with a bright red background at the top of a dismissable window that automatically repositions as the client browser window is scrolled. This function uses the "ezDispayHTMLSysMessages(s, t)" function except that the keyword "ERROR" clues the system into the fact that the window's title bar should be bright red.

ezErrorExplainer(errObj, funcName, bool_useAlert)

The "ezErrorExplainer(errObj, funcName, bool_useAlert)" takes three arguments that are the "errObj" a pointer to a JavaScript Error Object instance, the "funcName" a pointer to a String Object instance that identifies the error and "bool_useAlert" that when "true" uses the "ezAlertError(s)" function to display the meaning of the JavaScript Error in a pop-up window.

Top of the Document

JavaScript Abstract Event Handlers

window.onresize

The "window.onresize" event handler performs the function of handling the "onresize" event for the "window" object by firing the "ezWindowOnReSizeCallback(width, height)" Call-Back if that Call-Back function has been implemented.

window.onscroll[10]

The "window.onscroll" event handler performs the function of handling the "onscroll" event for the "window" object by firing the "ezWindowOnscrollCallback(scrollTop, scrollLeft)" Call-Back if that Call-Back function has been implemented. The floating debug menu, as it is called, is repositioned as the browser client window is scrolled along with the pop-up system message panel and the System Busy indicator panel. You can inhibit or enable the floating debug menu reposition action by setting the "bool_isDebugPanelRepositionable" variable as desired.

The const_div_floating_debug_menu variable[11]

It should be noted that the programmer can use the "const_div_floating_debug_menu" JavaScript constant to control where the floating debug menu is positioned during the "ezWindowOnscrollCallback(scrollTop, scrollLeft)" Call-Back assuming the "bool_isDebugPanelRepositionable" JavaScript variable is set to "false". Consider the following code sample:

```
var dObj = _$(const_div_floating_debug_menu);
if (!!dObj) {
        dObj.style.position = const_absolute_style;
        dObj.style.top = '100px';
        dObj.style.left = '100px';
        dObj.style.width = (ezClientWidth() - 175) + 'px';
}
```

Notice how easy it is to place this code fragment into the body of the "ezWindowOnscrollCallback(scrollTop, scrollLeft)" Call-Back. Once the "ezWindowOnscrollCallback" fires the floating debug menu will float to a position you choose.

The typical way to control how the floating debug menu floats is to choose a position relative to the top and left of the current browser window and then reposition the floating menu every time the "ezWindowOnscrollCallback" fires.

---

[10] A known bug was fixed in version 0.91.

[11] This feature was added to the version 0.92 release.

ezAJAX™ Processing Model

The ezAJAX™ Processing Model is based on a client-server or RPC (Remote Procedure Call) model that causes the client and the server to be as tightly coupled as possible. This is to say whatever data the server returns to the client is able to consume directly with no intermediate conversion step. Or to put another way, the server is used to perform SQL Queries which means the server has access to one or more Query Objects which are then transferred to the client to allow the client to have access to the same Query Objects the server had access to upon returning control to the client.

The ezAJAX™ Processing Model allows Query of Queries to be performed on the client as well as on the server. The methods one uses to perform Query of Queries in ColdFusion does differ somewhat from the method one uses to perform Query of Queries using the ezAJAX™ API but both the client and server allow Query of Queries to be executed against similar datasets.

Additionally just as the ColdFusion processing model allows Query of Queries to be executed against in-RAM Query Objects which can greatly accelerate Queries that are considered to be sub-queries of already existing queries so also the ezAJAX™ Processing Model provides this same level of acceleration.

Top of the Document

Query of Queries

Query of Queries in the ezAJAX™ Processing Model are constructed using "iterator" functions that are sent as a parameter to the iterator method of the Query Object. This will be discussed in greater detail shortly however for now you should simply be aware of the fact that ezAJAX™ provides this level of power and flexibility that is lacking from other AJAX Frameworks of which you may be aware.

Iterator functions are JavaScript functions that are executed against one Query row at a time until the entire Query Object contents have been processed.

Iterator functions can be used to perform Query of Queries for the purpose of collecting selected records from a Query Object.

Iterator functions can be used to perform aggregate functions such as SUM or AVERAGE of certain fields of a Query Object one row at a time.

Iterator functions are very powerful when used correctly.

Client-Server or RPC Processing

The ezAJAX™ Processing Model is said to be Client-Server in that there is a client, the browser, and there is a server, the ezAJAX™ Community Edition Framework Server.

The ezAJAX™ Client uses RPC (Remote Procedure Calls) to execute procedures found on the server each of which must return at least one Query Object to the client.

Keep in mind, the Community Edition Trial is limited to only one Query Object returned to the client at a time. The Community Edition Annual or Perpetual License allows one or many Query Objects to be returned to the client at a time and the ezAJAX™ Client properly processes both scenarios with equal ease.

It can be very useful to have the ability to return more than one Query Object from the server at a time since doing so allows the server-side logic to be simpler than if the same processing were done using a single Query Object.

The client uses the front-end of the ezAJAX™ Community Edition Framework while the server uses the back-end of the ezAJAX™ Community Edition Framework.

The front-end is composed of the following files: application.cfm, userDefined_application.cfm[12], index.cfm, cfinclude_index_body.cfm[13], javascript.js, StyleSheet.css, images (folder).

The back-end is composed of the following files: ezAJAX (folder). The userDefinedAJAXFunctions.cfc file is the only file that can contain user-defined code supplied by the programmer. The userDefinedAJAXFunctions.cfc file extends the ezAjaxCode.cfc file which contains the ezAJAX™ ColdFusion function library which is documented in more details below.

Top of the Document

---

[12] **userDefined_application.cfm** contains code supplied by the programmer who wishes to add code that is to be executed within the context of the application.cfm file to customize the Framework.

[13] **cfinclude_index_body.cfm** contains code supplied by the programmer who wishes to add application specific code that is to be executed within the context of the **index.cfm** file to make a specific application that uses the Framework. The list of files that are considered to be in the same category as this file are specified by the **Request.cfincludeCFM** variable that is defined within the userDefined_application.cfm file. The programmer could rename the cfinclude_index_body.cfm file to be any name desired so long as the reference to this file are changed in the **Request.cfincludeCFM** variable.

ezAJAX™ Call-Back Functions

The ezAJAX™ Community Edition Framework makes use of JavaScript based Call-Back functions the ezAJAX™ Server uses when transferring control back to the client from the server once the server concludes processing.

There are two models for Call-Back functions, the Complex Model and the Simpler Model.

The Complex Model is only a bit more complex than the Simpler Model but it allows the programmer to take more control over the flow of control once the Call-Back is fired by the ezAJAX™ Server.

The Simpler Model is much "simpler" than the Complex Model because it assumes the same processing is to be done just prior to the application specific code the programmer wanted to use for each Call-Back.

Of course it should be noted the programmer could, if desired, provide only one Call-Back through which all Call-Back processing could be done using some kind of Abstract Model the programmer may wish to define. Many Call-Back functions could just as easily be used depending on the wishes of the programmer and the goals the programmer is trying to achieve. For situations where there are very few differences between how certain Call-Backs should be coded it is useful to Abstract them into a single Call-Back.

Complex Model

A typical Complex Model Call-Back sample can be found in the cfinclude_index_body.cfm file that was shipped with the ezAJAX™ Community Edition Framework.

You may wish to refer to the cfinclude_index_body.cfm file while reading the following sections.

Top of the Document

handleSampleAJAXCommand(qObj)

Notice the JavaScript function "handleSampleAJAXCommand(qObj)". This is a sample of a Complex Model Call-Back function that takes one argument that is the "qObj" pointer to an instance of the ezAJAXObj Object.

Notice the line of code "qStats = qObj.named('qDataNum');" that is used to access the statistics from the ezAJAXObj Object instance.

Notice the line of code "nRecs = qStats.dataRec[1];" that is used to access the number of records that are stored in the ezAJAXObj Object instance. The ezAJAX™ Community Edition is limited to one data record which is to say the programmer will be allowed to return one (1) Query Object to the client. This limitation is removed for the ezAJAX™ Enterprise Edition along with some additional features such as the ability to return "Named" Query objects to the client and some performance improvements to make returning Query objects faster and more efficient. The limitation of being able to return one Query object instance should not pose much of a problem for the more skilled programmers because ColdFusion Query Objects can easily be modified and combined to allow almost any goal to be easily accomplished. It is more convenient, however, to have the ability to return many Named Query Objects back to the client as this can make the application specific code that goes in the userDefinedAJAXFunctions.cfc file easier to maintain as well as more powerful.

Notice the line of code "qData1 = qObj.named('qData1');" that is used to access the Query Object that was returned from the ezAJAX™ Community Edition Server.

Once the Query Object has been accessed from the ezAJAXObj Object instance one can process Query of Queries using aforementioned "iterator" functions using code such as the following: "qData1.iterateRecObjs(anyErrorRecords);" or "qData1.iterateRecObjs(searchForStatusRecs);".

The line of code "qData1.iterateRecObjs(anyErrorRecords);" is used to determine if there are any Error conditions that were returned within the Query Object instance that was transmitted back to the client from the server. Error conditions are flagged as being such whenever any of the following Column Names are used in the Query Object: 'ERRORMSG', 'ISPKVIOLATION' or 'ISHTML'. The reference to "anyErrorRecords" is an Abstract function that performs the processing to determine if the Query Object contains any Error Conditions.

The line of code "oParms = qObj.named('qParms');" is used to access the "arguments" that were passed to the ezAJAX™ Server along with the ezAJAX™ Command.

The line of code "oParms.iterateRecObjs(searchForArgRecs);" is used to perform a Query of Queries to collect-up the "arguments" from the Arguments Query Object instance.

Notice the function "searchForArgRecs(_ri, _dict)" that is modeled as a local function that takes two arguments which are the "_ri" the record index Integer and the "_dict" ezDictObj instance that contains the record or row of data. The keys from the "_dict" ezDictObj instance for "arguments" will always be 'NAME' and 'VAL'. It is however far more convenient to have an ezDictObj instance that has named arguments and thus the reason for performing the searchForArgRecs iterator on the oParms Query Object instance.

Notice the line of code "argsDict.intoNamedArgs();" that is used to convert the argsDict into a named args dictionary.

The line of code "qData1.iterateRecObjs(searchForStatusRecs);" is used to perform a Query of Queries and serves as a sample only to help illustrate how to construct an iterator function.

Notice the function "searchForStatusRecs(_ri, _dict)" that takes the same two arguments as the "searchForArgRecs(_ri, _dict)" function. These are the same two arguments that all Query of Queries iterator functions take.

Notice the line of code "aDict = ezDictObj.get$(_dict.asQueryString());" that is used within the "searchForStatusRecs(_ri, _dict)" function to convert the "_dict" ezDictObj from an instance pointer to a new ezDictObj instance. This has to be done within Query of Queries iterator functions because the "_dict" ezDictObj instance pointer points to an instance of a ezDictObj that is removed shortly after it is created within the code that fires the iterator function. As you can see it is possible and sometimes desirable to retrieve the whole Query Record Dictionary Object from the Query Object that record inhabits.

As you have seen it is quite easy to construct Query of Queries iterator functions that can access records from the Query Objects that are returned from the ezAJAX™ Server using whatever criteria one may desire to construct. This may not be as elegant as the way one constructs Query of Queries using ColdFusion but it gets the job done quite nicely nonetheless.

Notice the line of code "ezDictObj.remove$(argsDict.id);" that appears near the bottom of the Complex Model Call-Back function. This ensures we don't collect-up too many instances of ezDictObj Objects while processing Call-Back functions. Since there is little reason to leave the arguments dictionary in the Object cache beyond the scope of the Call-Back function we simply remove the whole ezDictObj instance using the appropriate function to deconstruct the object instance.

Top of the Document

Simpler Model

A typical Simpler Model Call-Back sample can be found in the cfinclude_index_body.cfm file that was shipped with the ezAJAX™ Community Edition Framework.

You may wish to refer to the cfinclude_index_body.cfm file while reading the following sections.

simplerHandleSampleAJAXCommand(qObj, nRecs, qStats, argsDict, qData1)

Notice the JavaScript function "simplerHandleSampleAJAXCommand(qObj, nRecs, qStats, argsDict, qData1)". This is a sample of a Simpler Model Call-Back function that takes five arguments which are the "qObj" pointer to an instance of the ezAJAXObj Object, "nRecs" Integer number of records or Query Objects returned, "qStats" pointer to a statistics Query Object, "argsDict" pointer to an ezDictObj that contains the named arguments that were passed to the ezAJAX™ Server and "qData1" pointer to the Query Object that stores the data that was returned from the ezAJAX™ Server.

Notice how much simpler it is to use the Simpler Model Call-Back. Error handling is done automatically. The various data elements are automatically separated from the Object that is returned from the ezAJAX™ Server.

The programmer should still write enough code to properly validate that the arguments to a Simpler Model Call-Back are in-fact not "null" values as seen in the sample function provided with the base product.

The Simpler Model Call-Back, when used, would result in far less code to write and therefore would result in faster access times for end-users due to the need to download less content for a typical ezAJAX™ web app that uses Simpler Model Call-Backs.

Top of the Document

ezAJAXEngine.receivePacketMethod()

The "ezAJAXEngine.receivePacketMethod()" function which is a Class method for the ezAJAXEngine Object that is used to define to the system which "method" is to be used when receiving packets from the ezAJAX™ Server. Depending on the value this function returns the system will use either the Simpler Model or Complex Model for Call-Backs. The decision to use either Model for Call-Backs is considered to be dynamic in that one could code the value returned in a dynamic manner.

To use the Simpler Model Call-Backs one would return the value "const_simpler_symbol" from the "ezAJAXEngine.receivePacketMethod()" function that would be implemented within the code file the programmer can supply to the ezAJAX™ Community Edition Framework. By default, there is already a "ezAJAXEngine.receivePacketMethod()" function implemented that causes the Simpler Model to be used however it is an easy thing to cause the system to use Complex Model Call-Backs if doing so becomes necessary.

To use the Complex Model Call-Backs one would return any String value other than "const_simpler_symbol" from the "ezAJAXEngine.receivePacketMethod()" function.

Any suitably skilled programmer should be able to code the "ezAJAXEngine.receivePacketMethod()" function as stated above to cause the desired effect of using the Complex Model Call-Backs should doing so become necessary.

ezAJAX™ Server Command Specifications

The ezAJAX™ Server has been constructed to respond to specific "commands". Each "command" is a String Object instance of any length desired.

A typical invocation of an ezAJAX™ Server Command is as follows: "oAJAXEngine.doAJAX('sampleAJAXCommand', 'handleSampleAJAXCommand', 'parm1', 'parm1-value', 'parm2', 'parm2-value', 'parm3', 'parm3-value', 'parm4', 'parm4-value');" or "oAJAXEngine.doAJAX('sampleAJAXCommand', 'simplerHandleSampleAJAXCommand', 'parm1', 'parm1-value', 'parm2', 'parm2-value', 'parm3', 'parm3-value', 'parm4', 'parm4-value');".

Let's take a closer look at how this is done.

Consider the following: "oAJAXEngine.doAJAX('sampleAJAXCommand', 'handleSampleAJAXCommand', 'parm1', 'parm1-value', 'parm2', 'parm2-value', 'parm3', 'parm3-value', 'parm4', 'parm4-value');"

The function being called is the "oAJAXEngine.doAJAX()" function which is an instance method for the ezAJAXEngine Object. The "doAJAX()" method would therefore be sent to the oAJAXEngine Object instance. The ezAJAX™ Community Edition is limited to only one instance of the ezAJAXEngine at a time however the Enterprise Edition does not have this limitation.

The ezAJAX™ Server Command specification is "'sampleAJAXCommand'" which is a String Object instance that specifies the String literal of "sampleAJAXCommand". More will be said about the ezAJAX™ Server Command specification in later sections that describe how to code the back-end of the ezAJAX™ Server.

The ezAJAX™ Complex Model Call-Back specification is "'handleSampleAJAXCommand'" which is a String Object instance that specifies the String literal of "handleSampleAJAXCommand" that references a Function Object instance that in this case for this example, as taken from the code that was shipped with the product, is a Complex Model Call-Back.

The remaining arguments comprise a list of argument-name and argument-value specifications as follows: "'parm1', 'parm1-value'" where "'parm1'" is a String Object instance that specifies the String literal of "parm1" and "'parm1-value'" is a String Object instance that specifies the String literal of "parm1-value". A total of four (4) such arguments were specified by the sample ezAJAX™ Server Command as shown above in this section.

You may notice there is no difference between the Simpler Model Call-Back, as shown above which is as follows: "oAJAXEngine.doAJAX('sampleAJAXCommand', 'simplerHandleSampleAJAXCommand', 'parm1', 'parm1-value', 'parm2', 'parm2-value', 'parm3', 'parm3-value', 'parm4', 'parm4-value');" and the Complex Model Call-Back we just discussed as they both use the same ezAJAX™ Server Command format. This allows the programmer to dynamically change the Call-Back Model used without having to recode the Server Command specifications. This too makes the ezAJAX™ and easy product to use going forward.

Top of the Document

ezAJAX™ ColdFusion Function Library

You may access the ezAJAX™ ColdFusion Function Library from within the "ezAJAX.cfc.userDefinedAJAXFunctions.cfc" file which has been made available to you in which you can implement your ezAJAX™ Server Commands.

ezCfMail(toAddrs,fromAddrs,theSubj,theBody,optionsStruct)[14]

The "ezCfMail(toAddrs,fromAddrs,theSubj,theBody,optionsStruct)" function takes four required arguments which are "toAddrs" the email address to which the email is to be sent, the "fromAddrs" email address the email was sent from, the "theSubj" subject of the email and "theBody" the body of the email which can contain HTML. The optional argument "optionsStruct" allows a MIME attachment to be added to an email. Request.anError is a Boolean variable that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.errorMsg is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown.

The following code sample shows how a MIME attachment can be added to an email:

```
optionsStruct = StructNew();
optionsStruct.bcc = 'email@domain.com';
optionsStruct.cfmailparam = StructNew();
optionsStruct.cfmailparam.type = 'text/plain';
optionsStruct.cfmailparam.file = 'http://www.domain.com/file-to-send-as-attachment.html';
ezCfMail('toAddrs@domain.com', 'fromAddrs@domain.com', 'See the Attached file.', 'body of message', optionsStruct);
```

[14] Optional parameter "**optionsStruct**" was added to version 0.92

ezExecSQL(qName,DSN,sqlStatement)

The "ezExecSQL(qName,DSN,sqlStatement)" function takes three required arguments which are "qName" the name of the Query Object that holds the results of the ColdFusion Query, "DSN" the ColdFusion Data Source Name and "sqlStatement" the SQL Statement to be executed. It works best to make the Query Name a Global such as "Request.qName" to make it easier to perform Query of Queries using ColdFusion after this function completes. Request.errorMsg is a Boolean variable that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.dbError is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown. Request.isPKviolation is a Boolean variable that will be "true" if the SQL Statement resulted in a Primary Key Violation Error or "false" if no Primary Key Violation Error occurred. Request.explainError is a String variable that contains the ColdFusion Error explanation when Request.dbError is "true". Request.explainErrorHTML is a String variable that contains the ColdFusion Error explanation that contains when Request.dbError is "true". Request.moreErrorMsg is a String variable that contains a more verbose ColdFusion Error explanation when Request.dbError is "true".

Top of the Document

ezCfDirectory(qName,pathname,filter,recurse)

The "ezCfDirectory(qName,pathname,filter,recurse)" function takes three required arguments and one optional argument which are "qName" the name of the Query Object that holds the results of the <cfdirectory> function, the "pathName" fully qualified name of the directory, "filter" pattern of file names for which to Query and "recurse" optional Boolean that when "true" causes the <cfdirectory> function to search subdirectories in addition to the named directory. Request.directoryError is a Boolean variable that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.directoryErrorMsg is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown.

ezFilePathFromUrlUsingCommonFolder(url,path,cName)[15]

The "ezFilePathFromUrlUsingCommonFolder(url,path,cName)" function takes three arguments which are "url" the URL for a document for which a fully qualified path is desired, "path" for a known path that resides within the same webspace as the file

[15] This function was added to version 0.92.

the URL points to and "cName" that is a known common name that exists within the fully qualified path names for both the "url" and the "path" and returns the fully qualified path for the "url".

ezScopesDebugPanelContent()

The "ezScopesDebugPanelContent()" function takes no arguments and returns the HTML content for the following Scopes via a formatted <cfdump>: Application Scope, Session Scope, CGI Scope and Request Scope.

ezCfFileDelete(fName)

The "ezCfFileDelete(fName)" function takes one required argument that is "fName" the fully qualified file name to be deleted. Request.fileError is a Boolean variable that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.errorMsg is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown.

ezCfFileRead(fName,vName)

The "ezCfFileRead(fName,vName)" function takes two required arguments which are the "fName" fully qualified name of the file to be read and "vName" the variable name into which the contents of "fName" is placed. Request.fileError is a Boolean variable that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.errorMsg is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown.

Top of the Document

ezCfFileWrite(fName,sOutput)

The "ezCfFileWrite(fName,sOutput)" function takes two required arguments which are the "fName" fully qualified name of the file to be written and "sOutput" the variable name from which the contents of "fName" is to be written. Request.fileError is a Boolean variable that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.errorMsg is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown.

ezCfExecute(exeName,sArgs,iTimeout)

The "ezCfExecute(exeName,sArgs,iTimeout)" function takes three required arguments which are the "exeName" fully qualified name of the file to be executed by the OS, "sArgs" String value that specifies the Arguments to be passed to the executable file and "iTimeout" the number of seconds after which the file execution request is to be considered to be timed-out. Request.execError is a Boolean variable

that will be "true" if a ColdFusion was thrown while the email was being sent or "false" if no Error was thrown. Request.errorMsg is a String variable that is either blank "" or contains an error message from the ColdFusion Error that was thrown.

ezCfLog(sTextMsg)

The "ezCfLog(sTextMsg)" function takes one required argument that is "sTextMsg" the message that is to be logged in the standard ColdFusion Message Logging system.

ezCFML2WDDX(oObj)

The "ezCFML2WDDX(oObj)" function takes one required argument that is "oObj" a pointer to the ColdFusion Object that is to be converted into a WDDX data stream using the "CFML2WDDX" action. This function returns the WDDX data stream as requested.

ezWDDX2CFML(sWDDX)

The "ezWDDX2CFML(sWDDX)" function takes one required argument that is "sWDDX" a String value that points to a WDDX data stream that is converted back into a ColdFusion Object using the "WDDX2CFML" action. This function returns the ColdFusion Object as requested.

Top of the Document

ezGetToken(str, index, delim)

The "ezGetToken(str, index, delim)" function takes three required arguments which are "str" a String value, "index" the number of the "token" to be returned and "delim" the delimiter that delimits the tokens. This function runs much faster than the standard ColdFusion "GetToken()" function especially for larger chunks of strings.

ezIsBrowserIE()

The "ezIsBrowserIE()" function takes no arguments and returns a Boolean value which is "true" if the client browser is Internet Explorer or "false" if not. This function works with IE 6.x browsers.

ezIsBrowserFF()

The "ezIsBrowserFF()" function takes no arguments and returns a Boolean value which is "true" if the client browser is FireFox or "false" if not. This function works with FireFox 1.5.0.4 browsers.

ezIsBrowserNS()

The "ezIsBrowserNS()" function takes no arguments and returns a Boolean value which is "true" if the client browser is Netscape or "false" if not. This function works with Netscape 8.1 browsers.

ezIsBrowserOP()

The "ezIsBrowserOP()" function takes no arguments and returns a Boolean value which is "true" if the client browser is Opera or "false" if not. This function works with Opera 9 browsers.

ezIsTimeStamp(str)

The "ezIsTimeStamp(str)" function takes one required argument that is "str" a String value that may be a time stamp specification in the format of "{ts '2006-06-01 00:00:00'}". This function returns Boolean "true" or "false" depending on whether or not the "str" argument is a time stamp specification or not.

Top of the Document

ezFilterQuotesForSQL(s)

The "ezFilterQuotesForSQL(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the "'" single quote mark. This function returns the "s" with single quote marks replaced by double quote marks which seems to make SQL Server much happier especially if the goal is to make SQL Server accept the original single quote marks that are embedded within literal strings.

ezFilterIntForSQL(s)

The "ezFilterIntForSQL(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the alpha-numeric characters or non-numerics mixed with numeric characters. This function returns the "s" filtered so that the result is considered to be composed of characters that specify a numeric value that may be Integer or Floating point.

ezFilterQuotesForJS(s)

The "ezFilterQuotesForJS(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the "'" single quote mark. This function returns the "s" with single quote marks replaced by single quote marks properly coded to make JavaScript happy especially when instances of single quote marks are embedded within literal strings.

ezFilterQuotesForJSContent(s)

The "ezFilterQuotesForJSContent(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the "'" single quote mark. This function returns the "s" with single quote marks replaced by single quote marks properly coded to make the browser happy especially when instances of single quote marks are embedded within literal strings that are embedded with JavaScript content.

ezFilterDoubleQuotesForJSContent(s)

The "ezFilterDoubleQuotesForJSContent(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the double quote marks '"'. This function returns the "s" with double quote marks replaced by symbols properly coded to make the browser happy especially when instances of double quote marks are embedded within literal strings.

Top of the Document

ezFilterTradeMarkForJSContent(s)

The "ezFilterTradeMarkForJSContent(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the "™" symbol. This function returns the "s" with "™" replaced by symbols properly coded to make the browser happy especially when instances of "™" are embedded within literal strings.

ezFilterOutCr(s)

The "ezFilterOutCr(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the Chr(13) character. This function returns the "s" with Carriage Return characters removed.

ezFilterQuotesForSQL(s)

The "ezFilterQuotesForSQL(s)" function takes one required argument that is "s" a String value that may be composed of some instances of the "'" single quote mark. This function returns the "s" with single quote marks replaced by double quote marks which seems to make SQL Server much happier especially if the goal is to make SQL Server accept the original single quote marks that are embedded within literal strings.

ezListToSQLInList(sList)

The "ezListToSQLInList(sList)" function takes one required argument that is "sList" a String value that represents a comma delimited list of String literals. This function returns the "sList" coded in such a manner to allow SQL Server to use the result as a list of String literals.

ezCompressErrorMsgs(s)

The "ezCompressErrorMsgs(s)" function takes one required argument that is "s" a String value that contains strings that may be delimited by Carriage Returns and Line Feed characters. This function returns the "s" without the Carriage Returns and Line Feed Characters.

ezBrowserNoCache()

The "ezBrowserNoCache()" function takes no arguments and returns the required HTML code and header specifications to cause the client browser to not cache the HTML page for which this function is used.

Top of the Document

ezBeginJavaScript()

The "ezBeginJavaScript()" function takes no arguments and returns the required HTML code for a JavaScript 1.2 <script> tag.

ezEndJavaScript()

The "ezEndJavaScript()" function takes no arguments and returns the required HTML code that closes a JavaScript 1.2 <script> tag.

ezStripCommentBlocks(s)

The "ezStripCommentBlocks(s)" function takes one required argument that is "s" a String value that contains JavaScript comment blocks. This function returns the "s" without the JavaScript comment blocks.

ezStripComments(s)

The "ezStripComments(s)" function takes one required argument that is "s" a String value that contains JavaScript comments. This function returns the "s" without the JavaScript comments.

ezClusterizeURL(sURL)

The "ezClusterizeURL(sURL)" function takes one required argument that is "sURL" a String value that a fully qualified URL that specifies a domain name that references a numbered web server that participates in an ezCluster™ Web Server Cluster. This function returns the "sURL" such it references the Cluster Manager rather than the numbered web server it originally referenced. ezCluster™ is a product designed by Hierarchical Applications Limited that provides the means to create and deploy a fully functional web server cluster using as few as four off-the-shelf computers, if desired, with absolutely no OS support for clustering or expensive networking equipment using state-of-the-art techniques. ezCluster™ requires that all in-bound web traffic be directed to the Cluster Manager which serves as a central point of focus for the web server cluster. This function "ezClusterizeURL(sURL)" ensures all URLs for all web pages references the Cluster Manager so that subsequent web hits will be spread-out across the web servers that participate within a ezCluster™.

Top of the Document

ezProcessComplexHTMLContent(sHTML)

The "ezProcessComplexHTMLContent(sHTML)" function takes one required argument that is "sHTML" a String value that contains a mixture of complex HTML mixed with JavaScript and Style tags such as what may be created when a <cfdump> tag is used within a <cfsavecontent> tag. This function returns "aStruct" which is a ColdFusion Structure with two members, "aStruct.styleContent" that is the content from the Style tags, "aStruct.jsContent" that is the content from the <script> tags and "aStruct.htmlContent" that is everything that is neither Style nor JavaScript content. This function makes it possible to use ezAJAX™ to fetch <cfdump> content from the server to be displayed by the client as HTML-only content without the Styles and JavaScript.

ezRegisterQueryFromAJAX(qObj)

The "ezRegisterQueryFromAJAX(qObj)" function takes one required argument that is "qObj" a pointer to a ColdFusion Query Object. This function returns nothing but it does place the ColdFusion Query Object into the Queue of Query Objects to be returned to the ezAJAX™ Client. The ezAJAX™ Community Edition is limited to allowing one (1) ColdFusion Query Object at a time to be passed back to the ezAJAX™ Client however the Enterprise Edition does not have this limitation along with performance enhancements that compresses the data stream sent back to the ezAJAX™ Client by as much as 60% or more. The Enterprise Edition also allows named Queries to be sent back to the ezAJAX™ Client. Named Queries can be useful for those who wish to leverage the ability to code Abstract ezAJAX™ Server Call-Backs to maximize code reuse.

Top of the Document

ezAJAX™ Server Command Handlers

The ezAJAX™ Community Edition Framework makes it very easy to code server-side command handlers. It is possible to code a server-side command handler using as few as a dozen lines of code.

A typically simple server-side command handler would look like the following:

```
function userDefinedAJAXFunctions(qryStruct) {
        switch (qryStruct.ezCFM) {
                case 'sampleAJAXCommand':
                        qObj = QueryNew('id, status');
                        QueryAddRow(qObj, 1);
                        QuerySetCell(qObj, 'id', qObj.recordCount, qObj.recordCount);
                        QuerySetCell(qObj, 'status', 'OK', qObj.recordCount);
                        ezRegisterQueryFromAJAX(qObj);
                break;
        }
}
```

The above sample server-side command handler performs a very simple task of doing nothing more than creating a small Query Object that is passed back to the ezAJAX™ client. This is what makes ezAJAX™ so easy. The last thing any server-side command handler does is to "register" a Query Object that is sent back to the client.

Automatic Argument or Parameter Handling

ezAJAX™ provides a very easy to use argument or parameter handling mechanism that takes all the "named" parameters from the "oAJAXEngine.doAJAX()" invocation and collects them into an easy to use structure the programmer can use when coding server-side command handlers that reside in the "userDefinedAJAXFunctions.cfc" file.

The ColdFusion variable "Request.qryStruct.namedArgs" holds the named arguments that were passed from JavaScript to the ezAJAX™ Server via the "oAJAXEngine.doAJAX()" invocation. This allows the programmer to quickly and easily manipulate the arguments that were passed to the ezAJAX™ Server . This also makes it easy to use the "IsDefined()" ColdFusion function to determine when named argument has in-fact been passed to the server or not.

The "argsDict" parameter to a Simpler Model Call-Back is populated with the contents of the "Request.qryStruct.namedArgs" ColdFusion Structure. This makes it easy for the programmer to write abstract code that can be made aware of the arguments that were passed to the ezAJAX™ Server without having to necessarily have access to the original code that was written to interface with the ezAJAX™ Server.

Automatic Error Handling

ezAJAX™ provides a very easy to use Error Handling system for communicating errors from the ezAJAX™ Server to the client.

Consider the following server-side command handler that notifies the client that an error happened:

```
function userDefinedAJAXFunctions(qryStruct) {
        switch (qryStruct.ezCFM) {
                case 'sampleAJAXCommand':
                        qObj = QueryNew('id, errorMsg, moreErrorMsg, explainError,
isPKviolation');
                        QueryAddRow(qObj, 1);
                        QuerySetCell(qObj, 'id', qObj.recordCount, qObj.recordCount);
                        QuerySetCell(qObj, 'errorMsg', 'An Error occurred.',
qObj.recordCount);
                        QuerySetCell(qObj, 'moreErrorMsg', 'Verbose Error Message',
qObj.recordCount);
                        QuerySetCell(qObj, 'explainError', '', qObj.recordCount);
                        QuerySetCell(qObj, 'isPKviolation', false, qObj.recordCount);
                        ezRegisterQueryFromAJAX(qObj);
                break;
        }
}
```

The ezAJAX™ client when using the Simpler Call-Back Model keys on the following Query Column names when deciding when to pop-up an automatic Error Message Panel (Query Column Names are converted to upper-case once the Query Object reaches the client): ERRORMSG, ISPKVIOLATION, or ISHTML.

Any Query Object that uses any of the three reserved Column Names will be interpreted by the Simpler Call-Back Model as an Error Message and an automatic pop-up panel will be displayed to communicate the error condition to the end-user. Whenever the ISHTML Column Name is used it is assumed to be a Boolean value represented as a String Object instance which allows the ERRORMSG to contain HTML which is then displayed as HTML in the automatic pop-up panel that is used to communicate the Error condition to the end-user.

As the programmer you may choose to use the Complex Call-Back Model and handle the Error Conditions yourself using whatever technique meets your individual needs and goals.

ezAJAX™ stands ready to make this as easy as is desired or as powerful and flexible as is desired.

Default Error Handling (New for version 0.91)

ezAJAX™ automatically detects when the programmer issued a Server Command but then forgot to use the "ezRegisterQueryFromAJAX()" function to pass a Query Object back from the server to the ezAJAX™ Call-Back and issues an automatic Query Object that specifies an error message that states "**ezAJAXEngine Server Error - No Server Command implementer was detected. Double-check the userDefinedAJAXFunctions.cfc file to ensure you have implemented the command (*name-of-command-goes-here*).**".

Top of the Document

A Word about XML

XML is a nice way to handle data when taken in moderation.

ezAJAX™ allows XML to be used as-desired or as-needed as long as the XML is packaged as textual data and communicated back to the client via a Query Object. The client can easily access the XML and use it as XML once the XML has been pulled out of the client-side Query Object.

Programmers may notice it is easier to use Query Objects to pass data from the server to the client because doing so allows the client to cache Query Objects which can then be Queried using the Query of Queries feature that is built-into ezAJAX™.

ColdFusion, as well as other CGI Languages, was not designed to manipulate Query Objects using XML and it can take some extra programming effort to turn a Query Object into XML therefore the ability to easily transmit ColdFusion Query Objects to the client may result in faster application development and saved time and money.

JavaScript was not designed to consume XML directly so it can take extra programming time to make JavaScript consume data that is expressed as XML therefore it can be easier and faster to simply consume a Query Object using ezAJAX™ which can make development go faster which can save time and money.

ColdFusion can handle Query Objects faster than it can handle XML – this makes the server-side run faster when XML is not used.

JavaScript can handle ezAJAX™ Query Objects faster than it can handle XML – this can make the client-side run faster when XML is not used.

ezAJAX™ allows XML to be used which means the programmer is able to leverage the best of both worlds in which XML can be used without the performance penalties of using XML.

Top of the Document

A Word about JSON

JSON is a nice way to handle data when taken in moderation.

ezAJAX™ allows JSON to be used as-desired or as-needed as long as the JSON is packaged as textual data and communicated back to the client via a Query Object. The client can easily access the JSON and use it as JSON once the JSON has been pulled out of the client-side Query Object.

ColdFusion, as well as other CGI Languages, was not designed to manipulate Query Objects using JSON and it can take some extra programming effort to turn a Query Object into JSON therefore the ability to easily transmit ColdFusion Query Objects to the client may result in faster application development and saved time and money.

JavaScript was not designed to consume JSON directly so it can take extra programming time to make JavaScript consume data that is expressed as JSON therefore it can be easier and faster to simply consume a Query Object using ezAJAX™ which can make development go faster which can save time and money.

ColdFusion can handle Query Objects faster than it can handle JSON – this makes the server-side run faster when JSON is not used.

JavaScript can handle ezAJAX™ Query Objects faster than it can handle JSON – this can make the client-side run faster when JSON is not used.

ezAJAX™ allows JSON to be used which means the programmer is able to leverage the best of both worlds in which JSON can be used without the performance penalties of using JSON.

Top of the Document

A Word about the DoJo Toolkit[16]

The DoJo Toolkit was easily integrated with ezAJAX™ with minimal code changes.

Keep in mind the Dojo Toolkit reworks how the browser rendering engine works which means DoJo Apps take a bit more time to start-up than a standard DHTML or ezAJAX™ native App and certain ezAJAX™ may not work as one expects when running with the DoJo Toolkit. Specifically those ezAJAX™ client functions that use DHTML will possibly seem to fail but they are not failing per-se. The problem is the fact that the DoJo Toolkit performs all DHTML rendering itself once the DoJo Toolkit App has taken control.

If there is enough end-user support we may consider working-up a DoJo Toolkit specific version of ezAJAX™ that will properly work with the DoJo Toolkit using the DoJo Toolkit as-needed to perform all client-side GUI functions.

We have not created a DoJo Toolkit specific version of ezAJAX™ yet because the DoJo Toolkit works fine as-is with ezAJAX™ so long as ezAJAX™ is used to perform the RPC functions rather than GUI functions when running with the DoJo Toolkit.

The DoJo Toolkit tends to make what would be DHTML based processing appear quite a bit slower than native DHTML function and this is fine if this is what is desired. ezAJAX™ will eventually support the same level of GUI support found in the DoJo Toolkit but ezAJAX™ will always use native DHTML to do so. We are providing support for the DoJo Toolkit as a way to make it easy for those using the DoJo Toolkit to migrate into using ezAJAX™ since ezAJAX™ has a much more robust RPC Model than the DoJo Toolkit has.

[16] DoJo Toolkit is supported by changes made to version 0.92 – see the sample Mail App that proves functionality with the DoJo Toolkit.

A Word about the ezAJAX™ Enterprise Edition

ezAJAX™ Enterprise Edition will have more features and faster performance.

ezAJAX™ Enterprise Edition will provide Drag-n-Drop support just like DoJo.

ezAJAX™ Enterprise Edition will provide a JavaScript based Charting and Graphing API that is very powerful and allows high-impact charts and graphs to be displayed right in your end-user's browsers without the need for Flash or a fancy server-side image creator.

ezAJAX™ Enterprise Edition will provide a JavaScript based API for 3D Charts and Graphs that includes 3D Animation of graphical data.

Additional features can be added to the ezAJAX™ Enterprise Edition upon request as long as there is enough support from the user community to do so.

A Word about using ezAJAX™ to Code Games

Yes, we do have plans for releasing an Interactive Gaming API for the ezAJAX™ Enterprise Edition that leverages the power of the JavaScript Graphical API to make Sprites come to life using JavaScript.

We envision ezAJAX™ Enterprise Edition becoming a very nice platform for Interactive Game Development. There is every reason to expect us to publish this level of support for ezAJAX™ going forward because we want our customers to be able to deploy web based games that do not require the use of Flash or Director while achieving faster performance which means games that load-up faster and are more fun to play.

Top of the Document

ezAJAX™ and PHP

We do have plans for publishing a PHP Connector for ezAJAX™ that not only allows PHP Programmers to use EZAJAX™ but it also allows the FREE Developer Edition of ColdFusion MX 7, which is a full Enterprise version, run in a Production mode at NO COST.

If we get enough requests from our customers who want us to produce a PHP specific version of the ezAJAX™ Server then we may do this however it will be far easier for us to simply make the PHP Connector for ezAJAX™ come to life sooner than the PHP specific version.

ezAJAX™ and ASP

We do have plans for publishing an ASP Connector for ezAJAX™ that not only allows ASP Programmers to use EZAJAX™ but it also allows the FREE Developer Edition of ColdFusion MX 7, which is a full Enterprise version, run in a Production mode at NO COST.

If we get enough requests from our customers who want us to produce an ASP specific version of the ezAJAX™ Server then we may do this however it will be far easier for us to simply make the ASP Connector for ezAJAX™ come to life sooner than the ASP specific version.

ezAJAX™ and .Net

We do have plans for publishing a .Net Connector for ezAJAX™ that not only allows .Net Programmers to use EZAJAX™ but it also allows the FREE Developer Edition of ColdFusion MX 7, which is a full Enterprise version, run in a Production mode at NO COST.

If we get enough requests from our customers who want us to produce a .Net specific version of the ezAJAX™ Server then we may even do this however it will be far easier for us to simply make the .Net Connector for ezAJAX™ come to life sooner than the .Net specific version.

ezAJAX™ and ASP.Net

We do have plans for publishing a ASP.Net Connector for ezAJAX™ that not only allows ASP.Net Programmers to use EZAJAX™ but it also allows the FREE Developer Edition of ColdFusion MX 7, which is a full Enterprise version, run in a Production mode at NO COST.

If we get enough requests from our customers who want us to produce an ASP.Net specific version of the ezAJAX™ Server then we may even do this however it will be far easier for us to simply make the ASP.Net Connector for ezAJAX™ come to life sooner than the ASP.Net specific version.

Top of the Document

ezAJAX™ and Dreamweaver

ezAJAX™ is 100% compatible with Dreamweaver.

ezAJAX™ and Homesite

ezAJAX™ is 100% compatible with Homesite.

ezAJAX™ and Eclipse

ezAJAX™ is 100% compatible with Eclipse.

Top of the Document

Ask us to add new features to ezAJAX™

That's right. Go ahead and ask us to add new features to ezAJAX™. We have a number of very skilled software engineers in-house who just love to whip-up ColdFusion and JavaScript code. If you think of some nifty feature you would like to see in ezAJAX™ just go ahead and ask us. If there is enough support for adding the features you want added then we will whip-up the code and make those added features happen for you.