



[ezAJAX™ Community Edition Framework Programmer's Guide v0.94](#)





Preface.....	1
The Agile Methodology	1
The Future – even more Agile than Agile	2
What do you mean, automatic code generation ?	2
Introducing the Geonosis Database Engine	3
The Geonosis Object-Oriented Data Model	3
The Framework	4
What to do after completing the installation.....	4
Editable Files	5
“cfinclude_index_body.cfm”	6
“userDefined_application.cfm”	6
“ezAJAX_Init.cfm”	6
“userDefinedAJAXFunctions.cfc”	7
This is as Easy as it gets !	7
Debugging Support.....	8
Floating Debug Menu	8
Replace the Floating Debug Menu	10
Add to the Floating Debug Menu.....	10
Remove or Hide the Floating Debug Menu	11
Recap the Floating Debug Menu	11
ezappname.cfm.....	11



The ezAJAX_title variable	11
The Default <cfapplication> tag	11
The ezAJAX_webRoot variable.....	12
The Request.ezAJAX_isDebugMode variable.....	12
The ezAJAX_getDebugMode Custom Function	12
The Request.ezAJAX_Cr Constant	13
The Request.ezAJAX_Lf Constant.....	13
The Request.ezAJAX_CrLf Constant	13
The Request.AUTH_USER variable	13
The "err_ajaxCode" and "err_ajaxCodeMsg" variables.....	14
The "Request.ezAJAX_functions_cfm" variable	14
The "application.isColdFusionMX7" variable.....	14
The "userDefined_application.cfm" file	15
The "Request.cfincludeCFM" variable.....	15
The "Request.DOCTYPE" variable.....	15
ezCompiler™	16
index.CFM	16
StyleSheet.css	16
The "htmlHeader" variable.....	17
The "javascript.js" file	17
ezLicenser™	17
JavaScript Objects	17



ezAJaxContextObj	18
Constructor Method	18
ezAJaxContextObj.get\$().....	18
Object Instance Cache	18
Destructor Method.....	19
ezAJaxContextObj.remove\$(id)	19
Object Instances Destructor Method	19
ezAJaxContextObj.remove\$s()	19
Instance variables	19
Instance methods.....	19
toString()	19
ezAJAXObj.....	20
Constructor Method	20
ezAJAXObj.get\$()	20
Object Instance Cache	20
Destructor Method.....	20
ezAJAXObj.remove\$(id)	20
Object Instances Destructor Method	21
ezAJAXObj.remove\$s()	21
Instance variables	21
Instance methods.....	21
toString()	21



push(aName, datum)	21
pop()	21
named(aName)	22
ezAnchorPosition	23
Constructor Method	23
ezAnchorPosition.get\$ (anchormame)	23
Object Instance Cache	23
Destructor Method	23
ezAnchorPosition.remove\$(id)	23
Object Instances Destructor Method	24
ezAnchorPosition.remove\$\$()	24
Instance variables	24
Instance methods	25
toString()	25
getAnchorPosition(anchormame)	25
ezDictObj	26
Constructor Method	26
ezDictObj.get\$(aSpec)	26
Object Instance Cache	26
Destructor Method	27
ezDictObj.remove\$(id)	27
Object Instances Destructor Method	27



ezDictObj.remove\$s()	27
Instance variables	27
Instance methods	27
toString()	27
fromSpec(aSpec)	27
URLDecode()	28
asQueryString(ch_delim)	28
push(key, value)	28
put(key, value)	28
drop(key)	28
getValueFor(key)	28
getKeysMatching(aFunc)	29
getKeys()	29
adjustKeyNames(aFunc)	29
length()	29
keyForLargestValue()	29
intoNamedArgs()	30
init()	30
ezGUIActsObj	31
Constructor Method	31
ezGUIActsObj.get\$()	31
Object Instance Cache	31



Destructor Method	31
ezGUIActsObj.remove\$(id)	31
Object Instances Destructor Method	32
ezGUIActsObj.remove\$s()	32
Instance variables	32
Instance methods	32
toString()	32
pushButton(id)	32
push(id)	32
revertAspectsDict(aDict, oObj)	32
revertStylesDict(aDict, oObj)	33
pop(aHandle)	33
popBtn(aHandle)	33
popAll()	33
popAllButtons()	33
replaceAspectNamedFor(aHandle, aName, aVal)	33
replaceStyleNamedFor(aHandle, aName, aVal)	33
length()	34
ezJSON	35
Constructor Method	35
ezJSON.get\$(aUrl)	35
Object Instance Cache	35



Destructor Method	36
ezJSON.remove\$(id)	36
Object Instances Destructor Method	36
ezJSON.remove\$s()	36
Instance variables	36
Instance methods	36
toString()	36
ezThoughtBubbleObj	37
Constructor Method	37
ezThoughtBubbleObj.get\$(top, left, width, height).....	37
Object Instance Cache	37
Destructor Method	37
ezThoughtBubbleObj.remove\$(id)	37
Object Instances Destructor Method	38
ezThoughtBubbleObj.remove\$s()	38
Instance variables	38
Instance methods	38
toString()	38
QObj	39
Constructor Method	39
QObj.get\$(colNames)	39
Object Instance Cache	39



Destructor Method	40
QObj.remove\$(id)	40
Object Instances Destructor Method	40
QObj.remove\$s()	40
Instance variables	40
Instance methods	40
toString()	40
recordCount()	40
iterateRecObjs(func)	41
QueryAddRow()	41
getColNumFromColName(colName)	41
QuerySetCell(cName, vVal, rowNum)	41
ezAJAXEngine Behaviors	42
register_ezAJAX_function	42
Improved Error Handling in Version 0.91	43
ezAJAXEngine.browser_is_ff	43
ezAJAXEngine.browser_is_ie	43
ezAJAXEngine.browser_is_ns	43
ezAJAXEngine.browser_is_op	43
ezAJAXEngine.browserVersion()	44
ezAJAXEngine.isBrowserVersionCertified()	44
ezAJAXEngine.browserCertificationCallback()	44



JavaScript Object Instances	45
oAJAXEngine.....	45
JavaScript Variables.....	45
const_inline_style.....	45
const_none_style	45
const_block_style	45
const_absolute_style	45
const_relative_style.....	45
const_function_symbol.....	46
const_object_symbol	46
const_number_symbol	46
const_string_symbol.....	46
const_simpler_symbol.....	46
jsBool_isColdFusionMX7	46
jsBool_isDebugMode	46
jsBool_isServerLocal	47
fqServerName.....	47
cfinclude_index_body.cfm	47
oAJAXEngine.timeout	47
oAJAXEngine.showFrameCallback	47
oAJAXEngine.hideFrameCallback	47
oAJAXEngine.createAJAXEngineCallback	47



oAJAXEngine.showAJAXBeginsHrefCallback.....	48
oAJAXEngine.showAJAXDebugMenuCallback	48
oAJAXEngine.ezAJAX_serverBusyCallback.....	49
oAJAXEngine.ezAJAX_serverBusy_divName	50
oAJAXEngine.ezAJAX_serverBusy_bgColor.....	51
showAJAXScopesMenuCallback.....	52
ezWindowOnLoadCallback	52
ezWindowOnUnloadCallback	52
ezWindowOnReSizeCallback.....	52
ezWindowOnscrollCallback.....	53
handleSampleAJAXCommand.....	53
simplerHandleSampleAJAXCommand	53
JavaScript Functions	54
\$(id, _frame)	54
_\$\$(id, _frame)	54
ez2CamelCase(sInput).....	55
ezAlert(s)	55
ezAlertCODE(s)	55
ezAlertError(s)	56
ezAlertHTML(s)	56
ezAlertHTMLError(s)	56
ezButtonLabelByObj(btnObj).....	56



ezCfString()	56
ezClickRadioButton(id)	57
ezClientHeight()	57
ezClientWidth()	57
String.prototype.ezClipCaselessReplace(keyword, sText)	57
ezColorHex(cVal)	58
ezDeleteCookie(name, path)	58
ezDisableAllButtonsLike(id, bool)	58
ezDispayHTMLSysMessages(s, t)	59
ezDispaySysMessages(s, t)	59
ezDynamicObjectLoader(vararg_params)	59
ezElementPositonX(oObject)	59
ezElementPositonY(oObject)	59
ezErrorExplainer(errObj, funcName, bool_useAlert)	60
ezFirstFolderAfterDomainNameFromHref(hRef)	60
ezFilePath2HrefUsingCommonFolder(fPath, hRef, commonFolder)	60
String.prototype.ezFilterInAlpha()	60
String.prototype.ezFilterInNumeric()	60
ezFlushCache\$(oO)	61
String.prototype.ezFormatForWidth(iWidth)	61
ezFullyQualifiedAppUrl()	61
ezFullyQualifiedAppPrefix()	61



ezGetCookie(name)	62
ezGetFilename().....	62
ezGetPath()	62
ezGetScrollLeft()	62
ezGetScrollTop()	62
ezGetStyle(el, style)	62
ezGetViewportHeight().....	63
ezGetViewportWidth().....	63
ezHex(ch)	63
ezInsertArrayItem(a,newValue,position).....	63
ezInt(i)	63
String.prototype.ezIsAlpha(iLoc)	63
String.prototype.ezIsNumeric(iLoc)	63
String.prototype.ezIsNumeric(iLoc)	63
ezLabelButtonByObj(bObj, sLabel).....	64
ezLocateArrayItems(a, what, start).....	64
ezObjectDestructor(oO)	64
ezObjectExplainer(obj).....	64
ezRemoveArrayItem(a,i)	65
ezRemoveEmptyItemsFromArray(ar)	65
String.prototype.ezReplaceSubString(i, j, s)	65
ezSelectionsFromObj(obj)	65



ezSetCookie(name, value, path).....	65
ezSetFocus(pObj)	65
ezSetFocusById(id).....	65
ezSetStyle(aStyle, styles).....	66
ezSimulateCheckBoxClick(id)	66
String.prototype.ezStripCrLfs()	66
String.prototype.ezStripHTML()	66
String.prototype.ezStripIllegalChars()	66
String.prototype.ezStripSpacesBy2s()	66
String.prototype.ezStripTabs(s)	66
ezStyle2String(aStyle)	67
ezBeginTable(aPrompt, vararg_params).....	67
String.prototype.ezTrim()	67
ezUnHookAllEventHandlers(anObj)	68
ezURLDecode(encoded).....	68
String.prototype.ezURLDecode()	68
ezURLEncode(plaintext)	68
String.prototype.ezURLEncode().....	68
ezURLPrefixFromHref(hRef)	69
ezUUID\$()	69
String.prototype.ezZeroPadLeading(num).....	69
JavaScript Abstract Event Handlers.....	69



window.onresize.....	69
window.onscroll	69
The const_div_floating_debug_menu variable	70
ezAJAX Processing Model.....	70
Query of Queries	71
Client-Server or RPC Processing	71
ezAJAX Call-Back Functions	72
Complex Model	73
handleSampleAJAXCommand(qObj).....	73
Simpler Model	75
simplerHandleSampleAJAXCommand(qObj, nRecs, qStats, argsDict, qData1) ..	75
ezAJAXEngine.receivePacketMethod()	76
ezAJAX Server Command Specifications	76
Let's take a closer look at how this is done.	77
ezAJAX ColdFusion Function Library	77
ezCfMail(toAddrs,fromAddrs,theSubj,theBody,optionsStruct)	78
ezExecSQL(qName,DSN,sqlStatement)	78
ezCfDirectory(qName,pathname,filter,recurse).....	79
ezFilePathFromUrlUsingCommonFolder(url,path,cName).....	79
ezScopesDebugPanelContent().....	79
ezCfFileDelete(fName)	79
ezCfFileRead(fName,vName).....	80



ezCfFileWrite(fName,sOutput).....	80
ezCfExecute(exeName,sArgs,iTimeout)	80
ezCfLog(sTextMsg)	80
ezCFML2WDDX(oObj)	80
ezWDDX2CFML(sWDDX).....	81
ezGetToken(str, index, delim).....	81
ezIsBrowserIE()	81
ezIsBrowserFF()	81
ezIsBrowserNS()	81
ezIsBrowserOP()	81
ezIsTimeStamp(str)	81
ezFilterQuotesForSQL(s).....	82
ezFilterIntForSQL(s)	82
ezFilterQuotesForJS(s)	82
ezFilterQuotesForJSContent(s)	82
ezFilterDoubleQuotesForJSContent(s).....	82
ezFilterTradeMarkForJSContent(s)	83
ezFilterOutCr(s)	83
ezFilterQuotesForSQL(s).....	83
ezJSONencode(arg)	83
ezListToSQLInList(sList)	83
ezCompressErrorMsgs(s)	84



ezBrowserNoCache()	84
ezBeginJavaScript()	84
ezEndJavaScript()	84
ezStripCommentBlocks(s)	84
ezStripComments(s)	84
ezClusterizeURL(sURL)	84
ezProcessComplexHTMLContent(sHTML)	85
ezRegisterQueryFromAJAX(qObj)	85
ezAJAX Server Command Handlers	85
Automatic Argument or Parameter Handling	86
Automatic Error Handling	86
Default Error Handling (New for version 0.91)	87
A Word about XML	88
A Word about JSON	89
A Word about the DoJo Toolkit	90
A Word about the ezAJAX Enterprise Edition	91
A Word about using ezAJAX to Code Games	91
ezAJAX and PHP	92
ezAJAX and ASP	92
ezAJAX and .Net	92
ezAJAX and ASP.Net	93
ezAJAX and Dreamweaver	93



ezAJAX and Homesite.....	93
ezAJAX and Eclipse	93
Ask us to add new features to ezAJAX.....	93



[Top of the Document](#)

Preface

ezAJAX is Easy to use Web 2.0 Product that will decrease your web development effort, decrease your time to market and increase the quality of your AJAX Products.

ezAJAX developers are required to produce only the code that is specific to the application they are developing rather than the foundation of the product.

ezAJAX provides more than 15,000 lines of code which is a mixture of 60% JavaScript and 40% 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 more than 15,000 lines of robust debugged JavaScript and ColdFusion MX 7 code ? We think you will save time and money by using our Framework. We think you will spend far less time and money by using our Framework than the relatively small fees we ask for our Runtime Licenses.

As this product matures, the developers of ezAJAX, will from time to time add *abstractions* to the product to allow an increasing number of 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.

[Top of the Document](#)

The Agile Methodology

ezAJAX fully supports the Agile Methodology as defined by [Kent Beck and others](#).

ezAJAX provides the means to make the Agile Methodology come to life quickly and easily.

ezAJAX provides the means to perform incremental development using small focused teams. Individual developers can make ezAJAX produce far more application functionality than may be possible using any other competing AJAX Framework. ezAJAX is able to automatically produce server-side code stubs that form the framework for whatever server-side logic your developers may wish to code.

ezAJAX fully supports the concept of converting currently existing non-AJAX web apps into powerful AJAX powered web apps by converting small chunks of functionality to use ezAJAX one GUI screen at a time.

ezAJAX can be installed into any folder or sub-folder within any structure of folders thus allowing ezAJAX to be used to convert an existing non-AJAX web app into an ezAJAX based web app.

ezAJAX was designed to do most of the work for the programmer which means the programmer need only provide a very small amount of code to handle ezAJAX Server Command Call-Backs. The ezAJAX Framework provides the rest of the code required to handle the bulk of the processing required to handle the actual Call-Back logic.

The Future – even more Agile than Agile

This version of the ezAJAX Framework produces ezAJAX Server Command code stubs whenever a previously unknown Server Command is presented to the ezAJAX Server at run-time. This allows your ezAJAX developers produce more application functionality per unit of time than would be possible using any other competing AJAX Framework.

Just how Agile can Agile be ?

Here's an example of just how Agile the word "agile" can be when used in conjunction with ezAJAX.

Every AJAX Framework must properly handle the browser back button and forward button problem. In case you were not aware, your browser and mine allow us to navigate backwards to previous URLs we have visited when we are clicking on links. Our browsers also allow us to navigate forwards after we have navigated backwards. AJAX tends to break this model because those who develop AJAX apps tend to use techniques that do not allow the browser's back and forward buttons to operate properly. Our development staff has researched many schemes for handling the browser back and forward button problem and our development staff has seen some very elaborate methods for handling this problem. Just about every method we have seen requires a bunch of code that doesn't seem to make any sense, even to our highly trained developers.

After some study into this problem our developers learned that ezAJAX has always handled the browser back and forward buttons when our iFrame AJAX method was used so no work was required to make that AJAX method work properly. After a bit more study our development staff learned they only had to code 2 little lines of code to make the browser back and forward button work for our xmlhttpRequest AJAX method.

Two little lines of code enabled our ezAJAX Framework to properly handle the major problem that has bothered AJAX developers from the dawn of the AJAX era in web development.

Some of you who may be reading this may be a bit incredulous about hearing it takes only 2 lines of code to handle the browser back and forward button problem but this is completely true for ezAJAX due to the way we architected our AJAXEngine.

The fact of the matter is it should never take more than 2 lines of code to make the browser back and forward buttons work for any AJAX Framework as long as those who developed that framework understood how the browser works in relation to the back and forward buttons.

The other thing an AJAX Developer might want to know about AJAX is that AJAX works best when AJAX is asked to handle the kinds of tasks we use AJAX for when we created our site at <http://www.ez-ajax.com> – go ahead and navigate around and use the browser back and forward buttons – like magic the content will appear and disappear under your command.

What do you mean, automatic code generation ?

During the development phase, the programmer can code the invocation for an unknown ezAJAX Server Command such as "FetchDataFromDb", for instance. As soon as the client-side code has been executed but before the ezAJAX Server Command has been coded the normal behavior for the ezAJAX Framework is to throw an error on the server that is automatically handled by the ezAJAX Server – the ezAJAX Server simply produces the stub code for the unknown Server Command. This allows the programmer to focus entirely on coding the specific application logic rather than the infrastructure that is required by the Framework. Every single automatically generated Server Command stub provides the required level of error handling ezAJAX is optimized to use and this too make the programmer more efficient and more focused on coding the application logic. Additionally the automatically generated stub code is composed of highly optimized Java code rather than tag language. ColdFusion is able to more quickly execute Java code at runtime thus allowing ezAJAX Server Commands to execute as quickly as possible.

Introducing the Geonosis Database Engine

Future versions of the ezAJAX Framework will provide a layer of functionality known as the Geonosis Database Engine. The Geonosis Database Engine provides an Object-Oriented Networked Database *abstraction* that effectively removes the need to hand-code SQL Statements for your data-driven applications.

The Geonosis Object-Oriented Data Model

The Geonosis Data Model is composed of an Object Class. Geonosis Objects are classified using an Object Class. Each Object can have one or many Attributes. Each Object also has a Value. Objects can be connected to other Objects.

ezAJAX Enterprise Edition will provide a GUI the programmer would use to manipulate Geonosis Objects. Geonosis Objects will also be manipulated using the

ezAJAX Geonosis API. The programmer will not be required to code any actual SQL Statements when using the Geonosis Data Model. No special database support will be required in the form of any Database Professionals because ezAJAX handles everything automatically. All that is required is a ColdFusion Data Source that points to an empty relational database. Any ODBC Data Source that ColdFusion can use can be used by the Geonosis API.

Geonosis is available only to those who purchase runtime licenses for the ezAJAX Enterprise Edition. Geonosis is not available to those who are using the ezAJAX Community Edition.

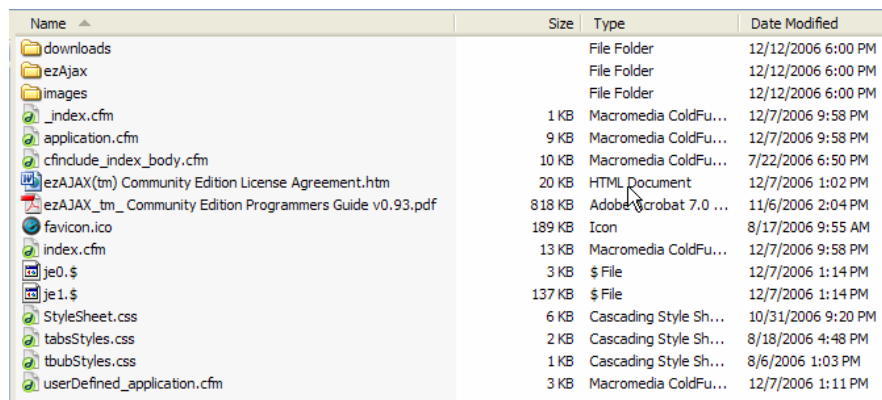
Geonosis also works with the rest of the ezAJAX API that already provides a very nice SQL Programming Layer. Developers can use traditional SQL Databases right alongside Geonosis Databases. ezAJAX works with any ODBC or JDBC Data Source; if you can get a JDBC driver for ColdFusion you can use that database engine with ezAJAX and the Geonosis Database Engine.

The primary goal for the ezAJAX Framework is to make the programming effort as efficient as possible. Reduce development time to as little programming time as necessary. Allow code to be reused as much as possible. And fully support the Agile Methodology or any other Rapid Application Development Methodology as much as possible. Geonosis will help to make ezAJAX a far more productive web development tool.

The Framework

What to do after completing the installation

Upon completing the Installation of the ezAJAX Community Edition Framework you will notice the following folder structure:



Name	Size	Type	Date Modified
downloads		File Folder	12/12/2006 6:00 PM
ezAjax		File Folder	12/12/2006 6:00 PM
images		File Folder	12/12/2006 6:00 PM
_index.cfm	1 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
application.cfm	9 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
cfindude_index_body.cfm	10 KB	Macromedia ColdFu...	7/22/2006 6:50 PM
ezAJAX(tm) Community Edition License Agreement.htm	20 KB	HTML Document	12/7/2006 1:02 PM
ezAJAX_tm_Community Edition Programmers Guide v0.93.pdf	818 KB	Adobe Acrobat 7.0 ...	11/6/2006 2:04 PM
favicon.ico	189 KB	Icon	8/17/2006 9:55 AM
index.cfm	13 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
je0.\$	3 KB	\$ File	12/7/2006 1:14 PM
je1.\$	137 KB	\$ File	12/7/2006 1:14 PM
StyleSheet.css	6 KB	Cascading Style Sh...	10/31/2006 9:20 PM
tabsStyles.css	2 KB	Cascading Style Sh...	8/18/2006 4:48 PM
tbodyStyles.css	1 KB	Cascading Style Sh...	8/6/2006 1:03 PM
userDefined_application.cfm	3 KB	Macromedia ColdFu...	12/7/2006 1:11 PM

Figure 1 - Contents of the installation folder

The installation folder is the folder into which the product was installed at the time the Installer Program was executed and all the prompts were properly handled by the user.

Let's assume the installation folder was named "wwwroot2". The folder structure would look like this:

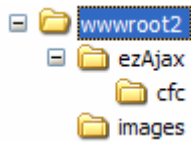


Figure 2 – ezAJAX Folder Structure

Name	Size	Type	Date Modified
cfc		File Folder	12/12/2006 6:00 PM
application.cfm	2 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
blank.html	1 KB	HTML Document	8/18/2006 8:54 PM
ezAJAX_End.cfm	9 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
ezAJAX_functions.cfm	18 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
ezAJAX_Init.cfm	1 KB	Macromedia ColdFu...	12/7/2006 1:00 PM
ezAjaxStyles.css	1 KB	Cascading Style Sh...	7/29/2006 3:36 PM
onError.cfm	3 KB	Macromedia ColdFu...	12/7/2006 9:58 PM

Figure 3 - The contents of the (installation folder)\ezAjax folder.

Name	Size	Type	Date Modified
ezAbstractCode.cfc	1 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
ezAjaxCode.cfc	1,079 KB	Macromedia ColdFu...	12/7/2006 9:58 PM
sampleAJAXCommand.cfc	4 KB	Macromedia ColdFu...	12/7/2006 7:43 PM
userDefinedAJAXFunctions.cfc	9 KB	Macromedia ColdFu...	12/7/2006 9:58 PM

Figure 4 - The contents of the (installation folder)\ezAjax\cfc folder.

[Top of the Document](#)

Editable Files

The following files can be edited by the programmer when adding application specific code to the framework:

Folder Name	File Name	Purpose
(installation folder)	cfinclude_index_body.cfm	Application Body
(installation folder)	userDefined_application.cfm	Application.cfm code
(installation folder)\ezAjax	ezAJAX_Init.cfm	AJAX Server Initialization Code

Figure 5 - User definable source files.

"cfinclude_index_body.cfm"

Notice the file named "cfinclude_index_body.cfm". This file is the default ColdFusion MX 7 source file into which you may place application specific code for your ezAJAX Application. The file named "cfinclude_index_body.cfm" is used exactly like the typical file named "index.cfm" except you are not allowed to edit the "index.cfm" that is part of the Framework.

You can define other files that have the same purpose as the "cfinclude_index_body.cfm" file by editing the file named "userDefined_application.cfm".

[Top of the Document](#)

"userDefined_application.cfm"

The file named "userDefined_application.cfm" is to be used exactly like the file named "application.cfm" file except you are allowed to edit file named "userDefined_application.cfm" but you cannot edit the file named "application.cfm".

Take a look at the contents of the file named "userDefined_application.cfm" and you will see what values and variables are used to define additional files that serve the same purpose as the file named "cfinclude_index_body.cfm".

"ezAJAX_Init.cfm"

The file named "ezAJAX_Init.cfm" contains AJAX Server Initialization Code statements that are executed every time an AJAX Server Command is transmitted to the ezAJAX Server. As you can see from looking at the contents of the "ezAJAX_Init.cfm" file there is not much going on with this file other than to enable or disable Debugging Information when AJAX Server Commands are executed using the <iframe> method. It is not meaningful to enable Debugging of AJAX Server Commands when the <iframe> method is not being used because it is not possible for you as the programmer to "see" or make use of Debugging information when the <iframe> method is not being used. It is advantageous to enable Debugging Information when writing your own AJAX Server Commands because you can use the Floating Debugging Menu (see the following section of the Programmer's Guide for the details) to show the contents of the hidden <iframe> during development. You would not want to allow the hidden <iframe> to be shown when your application is running on your Production Server however – this is when you would want to use the "oAJAXEngine.isXmlHttpRequestPreferred" Boolean flag which when set to "true" causes the AJAX Engine to use the "XmlHttpRequest" method rather than the hidden <iframe> method for transmitting AJAX Server Commands to the AJAX Server. The "XmlHttpRequest" method is much faster than the hidden <iframe> method but it is more difficult to debug AJAX Server Commands using the "XmlHttpRequest" method. Just to recap the "XmlHttpRequest" method uses whichever one of the following the client browser allows in order of appearance: "Msxml2.XMLHTTP" or "Microsoft.XMLHTTP" or "XMLHttpRequest()".

Both the "XmlHttp" and hidden <iframe> method are known to be 100% compatible with IE 6.x, FireFox 1.5.x, Netscape 8.x and Opera 8.x/9.x.

[Top of the Document](#)

"userDefinedAJAXFunctions.cfc"

The file named "userDefinedAJAXFunctions.cfc" is no longer where will place your application specific AJAX Server Commands – this change was made to allow the ezAJAX Server to automatically create Server Command stub files in the form of CFC's, one CFC for each Server Command.

We use the Java coding technique for all our CFC's because Java code that is placed between <cfscript> tags executes faster at runtime than any other coding technique for ColdFusion. We have provided generous support within the Framework to allow the Java coding technique to be used exclusively for maximum performance. If you encounter any ColdFusion functions that are not accessible via the Java coding technique just drop our [Support Department](#) an email and make a request and our programming staff will be happy to provide the level of support you require for your specific application.

Notice the Server Command known as "sampleAJAXCommand" that resides in the CFC called "sampleAJAXCommand.cfc" which has been placed into the folder named "\\ezAjax\\cfc".

Each specific Server Command has a corresponding CFC. The name of the CFC is the same as the Server Command identifier. You can use any valid CFC name (minus the .CFC file type) to name your Server Commands.

This makes it easier to make your ezAJAX Apps modular and it makes it easier for you to manage your server-side logic since now each Server Command goes into a separate CFC.

We have included the whole ezAJAX Site as a working sample which you can download from the Downloads Section of our site so you can see how we used ezAJAX to give you more ideas as to how you can use ezAJAX.

Keep in mind, ezAJAX is quickly heading in the direction of providing an architecture that lends itself to becoming an Automatic Code Generator. Beginning with Version 0.93 ezAJAX begins to make good on the promise of becoming an Automatic Code Generator by automatically creating Server Command Stub CFC's for you to help make your development tasks move right along.

This is as Easy as it gets !

As you can see there are only four (4) source files (see also: **Figure 5 - User definable source files**) you need be concerned with when coding your own ezAJAX Applications.

You may place your JavaScript code in any file other than those *.js files that are part of the ezAJAX Application. We have provided you with the ability to place custom code within the <head> of the default web document within the body of the "userDefined_application.cfm" file using the ColdFusion variable called "htmlHeader".

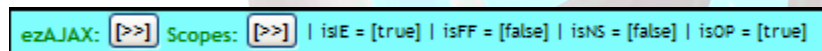
[Top of the Document](#)

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'; };
```

[Top of the Document](#)

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.

[Top of the Document](#)

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 = '&nbsp;<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.

[Top of the Document](#)

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 in 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 ezAJAX™ 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.

Very soon our development staff will release a modification to the ezCompiler™ that will optimize how the JavaScript loads at run-time based on the specific modules the programmer is interested in using rather than all the JavaScript the framework uses.

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.

ezLicenser™ allows anyone to produce an unlimited number of Runtime Licenses but only for a limited time based on the specific Runtime License for the copy of ezLicenser™ that has been licensed to you for your specific needs.

Every copy of ezLicenser™ is different and unique in that one ezLicenser™ cannot produce Runtime Licenses for another Application or another user's Application. This guarantees that ezLicenser™ cannot cross-license Applications.

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

JavaScript Objects

ezAJaxContextObj

The "ezAJaxContextObj" Object performs the function of creating a context in which AJAX parameters can be passed from Server Command to Server Command without having to specify the parameters each time a Server Command is issued.

[Top of the Document](#)

Constructor Method

ezAJaxContextObj.get\$()

The "ezAJaxContextObj.get\$()" function takes no arguments. The value that is returned is an ezAJaxContextObj 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 "ezAJaxContextObj" object is stored in the variable "ezAJaxContextObj.\$" which is an Array Object instance that holds all the instances of ezAJaxContextObj 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

ezAjaxContextObj.remove\$(id)

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

Object Instances Destructor Method

[Top of the Document](#)

ezAjaxContextObj.remove\$s()

The "ezAjaxContextObj.remove\$s()" function takes no arguments and performs the function of removing all Instances of the ezAjaxContextObj 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.

[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.

ezAJAXObj

The "ezAJAXObj" Object performs the function of aggregating data returned from the ezAJAX Server.

[Top of the Document](#)

Constructor Method

ezAJAXObj.get\$()

The "ezAJAXObj.get\$()" function takes no arguments. The value that is returned is an ezAJAXObj Object instance. It is not necessary for the programmer to ever create any ezAJAXObj Object instances because this is done automatically by the framework.

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 "ezAJAXObj" object is stored in the variable "ezAJAXObj.\$" which is an Array Object instance that holds all the instances of ezAJAXObj 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

ezAJAXObj.remove\$(id)

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

Object Instances Destructor Method

[Top of the Document](#)

ezAJAXObj.remove\$s()

The "ezAJAXObj.remove\$s()" function takes no arguments and performs the function of removing all Instances of the ezAJAXObj 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 "data" instance variable stores the instances of data objects that are typically QObj instances that hold Query Objects.

The "names" instance variable stores the names of the QObj instances.

[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.

push(aName, datum)

The "push(aName, datum)" instance method provides an easy way to add a Query Object instance to the list of QObj instances this object knows how to manage.

pop()

The "pop()" instance method pops a named QObj from the list of Query Objects that are managed by this object.

named(aName)

The "named(aName)" instance method returns a named QObj from the list of Query Objects this object manages.



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\$ (anchortname)

The "ezAnchorPosition.get\$ (anchortname)" 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 "anchormame" 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 "anchormame" instance variable.

The "anchormame" 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](#)

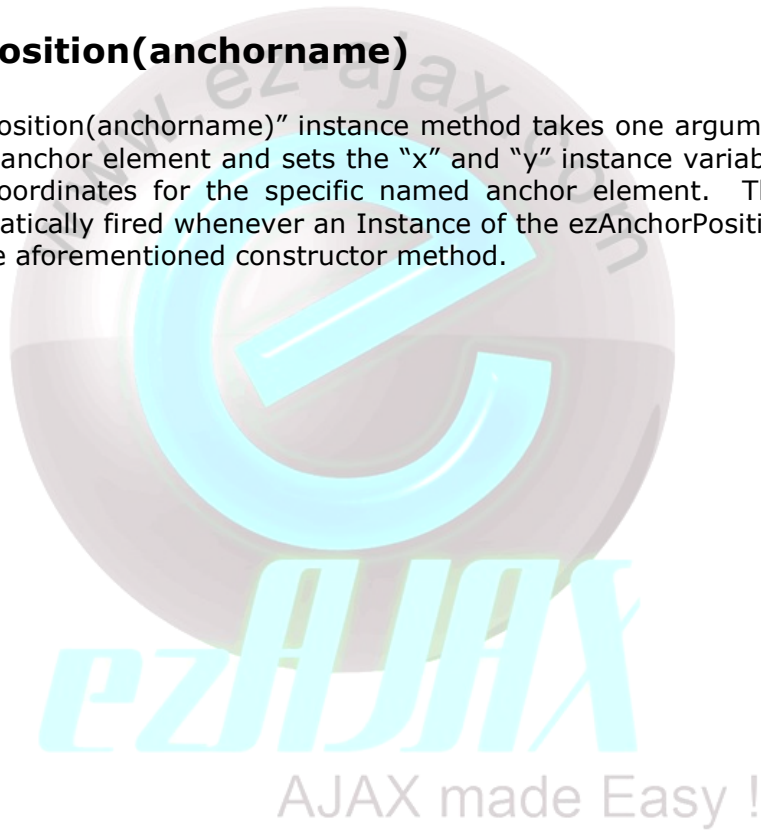
Instance methods

toString()

The "toString()" instance method provides an easy way to debug the contents of an Object instance via the "alert()" function or the "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.

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](#)

AJAX made Easy !

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.

[Top of the Document](#)

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.

[Top of the Document](#)

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.



ezGUIActsObj

The "ezGUIActsObj" Object performs the function of aggregating GUI actions that can be automated to make GUI programming easier to handle.

[Top of the Document](#)

Constructor Method

ezGUIActsObj.get\$()

The "ezGUIActsObj.get\$()" function takes no arguments. The value that is returned is an ezGUIActsObj 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 "ezGUIActsObj" object is stored in the variable "ezGUIActsObj.\$" which is an Array Object instance that holds all the instances of ezGUIActsObj 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

ezGUIActsObj.remove\$(id)

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

Object Instances Destructor Method

[Top of the Document](#)

ezGUIActsObj.remove\$s()

The "ezGUIActsObj.remove\$s()" function takes no arguments and performs the function of removing all Instances of the ezGUIActsObj 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.

[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.

pushButton(id)

The "pushButton(id)" instance method performs the function of stacking up a BUTTON object for later actions. This method returns a handle value that can be used to manipulate the stacked BUTTON.

push(id)

The "push(id)" instance method performs the function of stacking up a GUI object for later actions. This method returns a handle value that can be used to manipulate the stacked object.

revertAspectsDict(aDict, oObj)

The "revertAspectsDict(aDict, oo)" instance method performs the function of replacing a series of values that were previously stored in an ezDictObj instance. Each key from aDict is associated with a value such that each key is also an Array subscript for the oObj object instance.

revertStylesDict(aDict, oObj)

The "revertStylesDict(aDict, oo)" instance method performs the function of replacing a series of styles that were previously stored in an ezDictObj instance. Each key from aDict is associated with a value such that each key is also a CSS Style for the oObj object instance.

pop(aHandle)

The "pop(aHandle)" instance method performs the function of un-stacking a previously stacked object. The Aspects and Styles are automatically reverted when a pop is performed.

popBtn(aHandle)

The "popBtn(aHandle)" instance method performs the function of un-stacking a previously stacked BUTTON. The BUTTON disabled flag is automatically replaced with the original state of the disabled flag.

popAll()

The "popAll()" instance method performs the function of un-stacking all previously stacked objects using the pop(aHandle) method but automatically rather than manually.

popAllButtons()

The "popAllButtons()" instance method performs the function of un-stacking all previously stacked BUTTONS using the popBtn(aHandle) method but automatically rather than manually.

replaceAspectNamedFor(aHandle, aName, aVal)

The "replaceAspectNamedFor(aHandle, aName, aVal)" instance method performs the function of replacing or overwriting a named aspect for a previously stacked object.

replaceStyleNamedFor(aHandle, aName, aVal)

The "replaceStyleNamedFor(aHandle, aName, aVal)" instance method performs the function of replacing or overwriting a named CSS Style for a previously stacked object.

length()

The "length()" instance method performs the function of returning the number of previously stacked objects regardless of whether they are BUTTON objects or not.



ezJSON

The "ezJSON" Object performs the function of providing an Abstract JSON Object that is able to consume Web 2.0 API's that need to be accessed using a Cross-Domain technique. Typically this sort of Web 2.0 API will return a JSON data block through the use of a JavaScript Callback. Yahoo is known to publish the type of Web 2.0 API the ezJSON Object can consume. In fact Yahoo contacted our development staff to have us code for them this very Cross-Domain technique that our developers made into the ezJSON Object.

Constructor Method

ezJSON.get\$(aUrl)

The "ezJSON.get\$(aUrl)" function takes one argument that is "aUrl" String Object instance that specifies a cross-domain URL that describes the entry-point for the specific Web 2.0 API you wish to consume.

The typical way to go about using the ezJSON Object is to use the following code block:

```
jsonObj =  
ezJSON.get$('http://local.yahooapis.com/LocalSearchService/V2/localSearch?appid=YahooDemo  
&query=pizza&zip=94306&results=2&output=json&callback=handleJSONCallBack');
```

This is an actual real Yahoo Web 2.0 API – the same one we feature at our web site to demonstrate the ease with which ezAJAX consumes Yahoo Web 2.0 API's.

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 "ezJSON" object is stored in the variable "ezJSON.\$" which is an Array Object instance that holds all the instances of ezJSON 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

ezJSON.remove\$(id)

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

Object Instances Destructor Method

ezJSON.remove\$s()

The "ezJSON.remove\$s()" function takes no arguments and performs the function of removing all Instances of the ezJSON 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.

[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.

ezThoughtBubbleObj

The "ezThoughtBubbleObj" Object performs the function of providing a way to display Thought Bubbles that hover over a point on the browser's viewport.

Constructor Method

ezThoughtBubbleObj.get\$(top, left, width, height)

The "ezThoughtBubbleObj.get\$(top, left, width, height)" function takes four arguments that are "top" or y-coordinate, "left" or x-coordinate, "width" and "height" are self-explanatory.

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 "ezThoughtBubbleObj" object is stored in the variable "ezThoughtBubbleObj.\$" which is an Array Object instance that holds all the instances of ezThoughtBubbleObj 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

ezThoughtBubbleObj.remove\$(id)

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

Object Instances Destructor Method

ezThoughtBubbleObj.remove\$s()

The "ezThoughtBubbleObj.remove\$s()" function takes no arguments and performs the function of removing all Instances of the ezThoughtBubbleObj 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 "top" instance variable stores the y-coordinate for the Thought Bubble.

The "left" instance variable stores the x-coordinate for the Thought Bubble.

The "width" instance variable stores the width of the Thought Bubble.

The "height" instance variable stores the height of the Thought Bubble.

The "thoughtBubbleContentsID" instance variable stores the DIV id for the contents of the Thought Bubble. After the Thought Bubble has been created textual contents for the Thought Bubble can be dynamically added using the thoughtBubbleContentsID.

[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.

QObj

The "QObj" Object performs the function of providing a Query Object into which sets of data from the server can be placed. Data sets arrive from the server as JSON data block that are then converted to Query Objects. Query Objects are easier to Query than JSON data blocks. Query Objects carry a schema that allow data elements to be identified by name rather than by position as is typical for JSON data blocks. The problem with basing data access of positional data elements is that data schemas tend to change over time and when the data schema changes the JSON data blocks change and this can cause code that consumes JSON data blocks to need to change to keep pace with the JSON data block schemas. When the schema of a Query Object changes the code that consumes the Query Object could be made to notice a null value more easily than may be possible with JSON data blocks. Query of Queries that consume Query Objects can be made to notice when certain named data elements are present and ignore those that are not present and this lends itself to the effective consumption of JSON data blocks than if JSON alone were used.

Constructor Method

QObj.get\$(colNames)

The "QObj.get\$(colNames)" function takes one argument that is a comma delimited list of column names. This matches the syntax for ColdFusion Query Objects that also takes a comma delimited list of column names to make it easier and more natural to use the JavaScript variant of a Query Object. It is typically not necessary for the programmer to create Query Objects because the framework performs this automatically when ColdFusion Query Objects are returned from a Server Command.

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 "QObj" object is stored in the variable "QObj.\$" which is an Array Object instance that holds all the instances of QObj 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

QObj.remove\$(id)

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

Object Instances Destructor Method

QObj.remove\$s()

The "QObj.remove\$s()" function takes no arguments and performs the function of removing all Instances of the QObj 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 "colNames" instance variable stores the column names for the schema of the Query Object.

[Top of the Document](#)

AJAX made Easy !

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.

recordCount()

The "recordCount()" instance method returns the record count for the Query Object in which each row of data is one record.

iterateRecObjs(func)

The "iterateRecObjs(func)" instance method provides a slick and easy way to iterate over all records in a Query Object. The "func" argument is a pointer to a function that is passed the following arguments, one set for each row of data from the Query Object: rowIndex, aDict, qObj in which rowIndex is the index of the row of data from 1 to the number of records in the Query Object; aDict is an ezDictObj object that holds all the data elements for the rowIndex where the key is the name of the data element and the value is the data element value; qObj is the pointer to the Query Object itself as the third argument.

QueryAddRow()

The "QueryAddRow()" instance method allows one empty record or row to be added to the Query Object. This works the same way the ColdFusion counterpart works when building ColdFusion Query Objects.

getColNumFromColName(colName)

The "getColNumFromColName(colName)" instance method returns the number of a named column from the array of named columns.

QuerySetCell(cName, vVal, rowNum)

The "QuerySetCell(cName, vVal, rowNum)" instance method takes three arguments that are the cName or column name, the vVal or value for the column and rowNum that is the row number. This function works the same as the ColdFusion counterpart that has a similar name.

[Top of the Document](#)

AJAX made Easy !

ezAJAXEngine Behaviors

register_ezAJAX_function¹

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.

[Top of the Document](#)

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.

ezAJAXEngine.browser_is_ff²

This class variable returns "true" when the client browser is known to be a version of FireFox.

ezAJAXEngine.browser_is_ie³

This class variable returns "true" when the client browser is known to be a version of MSIE.

ezAJAXEngine.browser_is_ns⁴

This class variable returns "true" when the client browser is known to be a version of Netscape.

ezAJAXEngine.browser_is_op⁵

This class variable returns "true" when the client browser is known to be a version of Opera.

² This class variable was added to version 0.93.

³ This class variable was added to version 0.93.

⁴ This class variable was added to version 0.93.

⁵ This class variable was added to version 0.93.

ezAJAXEngine.browserVersion()⁶

This class method returns the client browser version in the form of a floating point value. For instance, MSIE 6.0 returns the value of "6.0"; FireFox 1.5.0.6 returns the value of "1.506"; Netscape 8.1 returns the value of "8.1" and Opera 9.1 returns the value of "9.1".

ezAJAXEngine.isBrowserVersionCertified()⁷

This class method returns "true" when the client browser is known to be fully functional from the perspective of ezAJAX otherwise "false" is returned.

ezAJAXEngine.browserCertificationCallback()⁸

This Call-Back method is fired whenever the client browser is known to be outside the boundaries of those browsers that are known to be fully compatible with ezAJAX. The following browser versions are certified to work with ezAJAX: MSIE 6.0, FireFox 1.5.0.x, Netscape 8.x and Opera 9.x. No other browsers are known to be fully functional with ezAJAX, if you know your browser is fully functional with the current version of ezAJAX you may code your own Call-Back method to override the default behavior that pops-up a warning to let users know when their client browser is not known to be compatible with ezAJAX.



⁶ This class method was added to version 0.93.

⁷ This class method was added to version 0.93.

⁸ This Call-Back method was added to version 0.93.

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.

[Top of the 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_block_style

The "const_block_style" variable stores the value of 'block'. This value is used to set the display style to block to allow <div> to be shown using the block style.

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_relative_style

The "const_relative_style" variable stores the value of 'relative'. This value is used to set the position style to relative to allow an HTML element to be positioned in a relative manner.

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 of 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.showAJAXBeginsHrefCallback⁹

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.

[Top of the Document](#)

oAJAXEngine.showAJAXDebugMenuCallback¹⁰

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.

⁹ This Call-Back was added to version 0.91.

¹⁰ This Call-Back was added to version 0.92.

oAJAXEngine.ezAJAX_serverBusyCallback¹¹

The “oAJAXEngine.ezAJAX_serverBusyCallback” is a Call-Back function pointer that is fired whenever the ezAJAXEngine is asked to display the Server Busy indicator before every AJAX Server Command is transmitted to the server. The default behavior for the Server Busy indicator displays an animated GIF in the upper right corner of the browser’s client area, assuming there is a DIV named “oAJAXEngine.ezAJAX_serverBusy_divName” that has been placed in the desired location. The AJAX Server Busy Indicator can be operated in two different modes; the fixed placement mode and the absolute placement mode. The fixed placement mode requires a DIV be placed in the location the programmer wants the AJAX Server Busy Indicator to be placed whenever it needs to be displayed by the Framework. The absolute placement mode does not require the programmer to place a DIV in any specific location as this is done by the Framework. Both methods for displaying the AJAX Server Busy Indicator require some code to be supplied by the programmer in the form of a Call-Back method as shown below:

Table 1 - Fixed Placement Mode Call-Back Sample

```
oAJAXEngine.ezAJAX_serverBusy_divName = 'div ezajax 3d logo';
oAJAXEngine.ezAJAX_serverBusyCallback = function (cObj, resp) { var oO =
    $('iframe showAJAXBegins'); if (!!oO) { oO.contentWindow.document.writeln(resp); if
    (oO.style.display == const none style) { oO.style.display = const inline style; }; }; };
```

As you can see from the code fragment above, the fixed position of the AJAX Server Busy Indicator is based on the known position of a DIV that must be placed in the location the programmer wants the animated GIF to be shown. In this case the DIV’s name is “div_ezajax_3d_logo” however this name could be any unique name the programmer desires. The element named “iframe_showAJAXBegins” is supplied by the Framework as a constant static element and must be the same for every application the programmer codes using the Framework.

Table 2 - Absolute Placement Mode Call-Back Sample

```
oAJAXEngine.ezAJAX_serverBusyCallback = function (cObj) { var oPos =
    ezAnchorPosition.get$('anchor_imageLogoRight'); if (!!oPos) {
    resizeOuterContentWrapper(ezClientWidth()); cObj.style.top = (oPos.y +
    parseInt(this.ezAJAX_serverBusy_height.toString())) + 'px'; cObj.style.left = (oPos.x -
    (parseInt(this.ezAJAX_serverBusy_width.toString()) / 4)) + 'px'; cObj.style.zIndex = 1;
    ezAnchorPosition.remove$(oPos.id); } };
```

As you can see from the code fragment above, the absolute position of the AJAX Server Busy Indicator is based on the known position of an anchor using the known size of the Server Busy Indicator image. The specific code you as the programmer

¹¹ This Call-Back was added to version 0.93.

would write for this function would of course depend on the specific image you wish displayed, assuming you wanted an image to be displayed. Almost any type of indicator could be used however the Framework supports the function of displaying an animated GIF whenever the AJAX Server is placed into a busy mode.

You as the programmer may choose to use the default AJAX Server Busy Indicator or create on yourself. In either case the Call-Back specified in the above section would facilitate whatever means you wish to use when conveying to the end-user the fact that the AJAX Server is Busy and therefore cannot be asked to process another request at the moment.

Bear in mind the fact that the programmer could just as easily queue-up AJAX Server Requests regardless of whether the AJAX Server is "busy" or not and then allow those requests to be executed in order whenever the AJAX Server becomes un-busy. (See also the sections on the topic of "[oAJAXEngine.register_ezAJAX_function](#)".)

It would also be possible to use more than one instance of ezAJAXEngine however doing this would be a bit more difficult because the programmer might have to deal with multiple AJAX Server Busy Indicators or possibly there might be a need to marshal client-side processing to keep from changing client-side variables out of order. In most cases the use of a single instance of an ezAJAXEngine object will be more than sufficient. Background processing such as that which needs to run as a result of a timer-tick (see also: `setInterval()`) can be coordinated with foreground processing by carefully checking the "`oAJAXEngine.isIdle()`" value to make sure it is "true" which means the ezAJAXEngine instance is not currently in the middle of processing an AJAX Server Command and therefore able to accept a new one such as from a background process. When care is taken to always check the "`oAJAXEngine.isIdle()`" value prior to issuing an AJAX Server Command within a loop, for instance, the effect can be quite interesting in that one soon finds the ezAJAXEngine object instance can then be used to marshal server-side processing without the need to use the "[oAJAXEngine.register_ezAJAX_function](#)" however the side-effect might be that the client may become unstable unless care is taken to break out of the loops that watch to see when the ezAJAXEngine object instance has once again become available such as when "`oAJAXEngine.isIdle() == true`".

[Top of the Document](#)

oAJAXEngine.ezAJAX_serverBusy_divName¹²

Alternatively, it is also possible to cause the Server Busy Indicator to be shown in a known relative position within a named div tag. Use the following code fragment to specify the named div tag:

¹² This instance variable was added to version 0.93.

```
oAJAXEngine.ezAJAX serverBusy divName = 'div ezajax 3d logo';
```

As you can see from the code fragment above, the name of the div tag is "div_ezajax_3d_logo" which should be coded as follows:

```
<div id="div ezajax 3d logo">  
  <iframe id="iframe_showAJAXBegins"  
    frameborder="0"  
    marginwidth="0"  
    marginheight="0"  
    scrolling="No"  
    width="30" height="30" style="display: none; z-index: 1;"></iframe>  
</div>
```

Notice the "<iframe>" in the above code fragment. The name for this <iframe> is very important and must appear exactly as it does in this code fragment. This allows the Server Busy indicator to be placed in any location within the browser's client area based on the relative positions of the various tags. Table layouts can be used to control where the Server Busy indicator appears.

oAJAXEngine.ezAJAX_serverBusy_bgColor¹³

The "oAJAXEngine.ezAJAX_serverBusy_bgColor" instance variable allows the background color of the Server Busy indicator to be programmer defined. The following code fragment shows how this should be done:

```
oAJAXEngine.ezAJAX serverBusy bgColor = 'white';
```

Notice the "oAJAXEngine.ezAJAX_serverBusy_bgColor" instance variable in the above code fragment has been set to be the "white" color which matches the color of the region of the browser's client area where the Server Busy Indicator will be shown. This has the effect of making it appear as though the Server Busy indicator is a transparent GIF.

Keep in mind the Server Busy indicator image is an animated GIF that is being shown through an <iframe> to allow the animation to be played regardless of how busy the browser may be at the moment. If an <iframe> was not being used the animated GIF would sometimes fail to animate such as whenever the browser was busy performing an AJAX Server Command.

[Top of the Document](#)

¹³ This instance variable was added to version 0.93.

showAJAXScopesMenuCallback¹⁴

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.

ezWindowOnReSizeCallback¹⁵

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. This Call-Back also handles the layout of the Tabbed Interfaces such that whenever the

¹⁴ This Call-Back was added to version 0.92.

¹⁵ This Call-Back was modified in version 0.93 to allow fixed width layouts to control Tabbed Interfaces – the fixed width for your layout must be returned from this Call-Back if your layout width is not 800 pixels.

client width changes the position of the Tabbed Interfaces changes to maintain the alignment of the Tabbed Interface to the left of the content area which is assume to be centered in the middle of the client area of the browser. Most professional web page layouts do not exceed 800 pixels and are always centered in the middle of the client area of the browser.

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](#)

AJAX made Easy !

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.

[Top of the Document](#)

`$(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.

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.

ezAlert(s)¹⁶

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 dismissible 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.

ezAlertCODE(s)¹⁷

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

¹⁶ 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(), ezDisplaySysMessages(), ezDisplayHTMLSysMessages(), ezAlertError() and ezAlertHTMLError().

¹⁷ This function was added in version 0.92.

dismissible window that automatically repositions as the client browser window is scrolled.

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 dismissible window that automatically repositions as the client browser window is scrolled. This function uses the "ezDisplaySysMessages(s, t)" function except that the keyword "ERROR" clues the system into the fact that the window's title bar should be bright red.

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 dismissible 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.

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 dismissible window that automatically repositions as the client browser window is scrolled. This function uses the "ezDisplayHTMLSysMessages(s, t)" function except that the keyword "ERROR" clues the system into the fact that the window's title bar should be bright red.

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.

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.

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()¹⁸

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.

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.

¹⁸ All references to eClientHeight() were changed to ezClientHeight() to correct a known bug, in version 0.91.

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.

ezDeleteCookie(name, path)

The "ezDeleteCookie(name, path)" function removes the named client-side cookie by forcing the named cookie to expire rather immediately. The problem with client-side cookies is that they can be disabled by the end-user thus causing client-side cookies to possibly become moot and useless. It is easy enough to test for client-side cookies to see if the end-user has disabled them and then issue a pop-up that states the client-side cookies are required but this can annoy end-users. Server-side cookies are better than client-side cookies because end-users cannot disable server-side cookies. The best place to store server-side cookies is in the user's Session however the user's Session should be made persistent in your database. We have produced an exciting Web Server Clustering system we call ezCluster that provides a very nice Session Persistence Framework that stores the user's Session in a database. ezCluster makes server-side cookies very easy to use and the ability to use server-side cookies with a persistent Session makes coding User Authentication Systems very easy to construct. Remember this sort of thing is in your hands, assuming you are the developer, you will want your end-user to feel like your web based application was produced by some people who paid attention to the details rather than not. If you bug your end-user to enable client-side cookies your end-user may conclude the web app he or she is using was produced by some developers who may not have been all that technically astute. You could also simply architect your own server-side cookie storage system by creating a Relational Database Table and then code some Server Commands to store the data you might have been tempted to store in a client-side cookie in your database table instead. ezAJAX Enterprise Edition will soon ship with a slick database abstraction layer code named Geonosis. The Geonosis Database Engine will make it possible for you to manipulate data in your Relational Database without the need to code any SQL Statements yourself. As you can see there are numerous options you may be able to use. Choose the options for your web apps that make the most sense while providing your end-users with the best user experience possible.

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

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.

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.

ezDynamicObjectLoader(vararg_params)

The "ezDynamicObjectLoader(vararg_params)" function takes a variable number of arguments each of which may describe either a CSS or JavaScript file using the ".css" or any other file type. File names that do not end in ".css" are assumed to be JavaScript source files. All specified files are loaded asynchronously and appended to the HEAD Object. Take care when using this feature because specified files are not guaranteed to be loaded immediately and may in fact be loaded with some slight delay.

ezElementPositonX(oObject)¹⁹

The "ezElementPositonX(oObject)" function takes an argument which is an instance of an HTML Element such as a Div. The value returned from this function is the X position of the HTML Element whose instance is oObject.

ezElementPositonY(oObject)²⁰

The "ezElementPositonY(oObject)" function takes an argument which is an instance of an HTML Element such as a Div. The value returned from this function is the Y position of the HTML Element whose instance is oObject.

¹⁹ This function was added with version 0.93.

²⁰ This function was added with version 0.93.

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.

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.

[Top of the Document](#)

ezFilePath2HrefUsingCommonFolder(fPath, hRef, commonFolder)²¹

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.

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.

²¹ A known issue was fixed in version 0.92.

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.

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.

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.

ezGetCookie(name)

The "ezGetCookie(name)" function takes one argument that is the name of a client-side cookie; the value of the cookie is returned if present. It should be noted that client-side cookies can be disabled by the end-user therefore we do not recommend using them – we do recommend using persistent server-side cookies that reside in the User's Session made persistent in a database. See also our product called ezCluster for full support for server-side cookies or our product called Geonosis that ships with the ezAJAX Enterprise Edition.

ezGetFilename()

The "ezGetFilename()" function returns the name of the file name for the current page the browser requested from the server. When the current page is the default page this function will return "/" since there was no file name specified even if the default page is known by the server to be something like "index.html" or the like there is no way for the browser to know anything more than what the browser requested from the server.

ezGetPath()

The "ezGetPath()" function returns the path name for the current page the browser requested from the server.

ezGetScrollLeft()

The "ezGetScrollLeft()" function returns the position of the left side of the browser's viewport based on where the browser has been scrolled to view.

ezGetScrollTop()

The "ezGetScrollTop()" function returns the position of the top of the browser's viewport based on where the browser has been scrolled to view.

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.

[Top of the Document](#)

ezGetViewportHeight()

The "ezGetViewportHeight()" function returns the height of the current browser viewport based on where the browser has been scrolled to view.

ezGetViewportWidth()

The "ezGetViewportWidth()" function returns the width of the current browser viewport based on where the browser has been scrolled to view.

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.

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.

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.

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.

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.

[Top of the Document](#)

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. Prior to version 0.94 this function was not as robust as it is in now. Now this function properly traverses any JavaScript object using a more intelligent manner. If the Object has a toString() method implemented then that Object's toString() method is used otherwise the various members of the Object are traversed one at a time until the whole Object has been completely explained into a single string.

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.

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".

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.

ezSetCookie(name, value, path)

The "ezSetCookie(name, value, path)" function takes three arguments which are the name of the cookie the value of the cookie and the path for the cookie. This function is used to set a client-side cookie. Keep in mind the end-user can disable cookies. Server-Side cookies are better and more professional. Check into our ezAJAX Enterprise Edition for support for handling server-side cookies or feel free to code your own when using ezAJAX Community Edition.

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.

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.

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".

String.prototype.ezStripCrLf()

The "ezStripCrLf()" 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.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.

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.

[Top of the Document](#)

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.

ezBeginTable(aPrompt, vararg_params)

The "ezBeginTable(aPrompt, vararg_params)" function takes at least one argument that is the prompt string for the Table; the prompt can contain HTML statements. The remaining arguments after the prompt describe the Table child element to which certain attributes are applied.

Consider the following sample code snippet:

```
ezBeginTable(' <b>Objects</b>', '&tr=' + 'bgcolor="silver"'.ezURLEncode() + '&td=' +  
'align="center"'.ezURLEncode() + '&span=' + 'class="boldPromptTextClass"'.ezURLEncode());
```

Notice the "&tr=" and the "&td=" and the "&span=" tokens; these are used to describe the specific Table child elements. Many attributes can be applied to the same Table child element however for each attribute one of the tokens "&tr=" or "&td=" or "&span=" must be specified. The above code snippet specifies the "bgcolor" attribute with the silver color for the <tr> element. The above code snippet also specifies the align attribute with the center value for the <td> element. The above code snippet also specifies the CSS class of boldPromptTextClass for the element.

Consider the following sample code snippet:

```
ezBeginTable('Add an Object', '&bool=' + 'includeDismissButton=true'.'.ezURLEncode() +  
'&div=' + 'dismissDiv=div_addObject_panel'.'.ezURLEncode() + '&tr=' +  
'bgcolor="silver"'.ezURLEncode() + '&td=' + 'align="center"'.ezURLEncode() + '&span=' +  
'class="boldPromptTextClass"'.ezURLEncode());
```

For the above sample, we are introducing a new set of tokens: "&bool=" and "&div=". The "&bool=" token is used to specify the "includeDismissButton" attribute that only makes sense to use if the value is "true" otherwise the default is that there is no dismiss button and therefore no way to dismiss the Table. The "&div=" token is used to specify the <div> element that contains the Table so that when the <div> that contains the Table is hidden the Table will appear to be dismissed.

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.

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, DbClick, 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.

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.

String.prototype.ezURLDecode()

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

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.

String.prototype.ezURLEncode()

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

ezURLPrefixFromHref(hRef)²²

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.

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.

String.prototype.ezZeroPadLeading(num)

The “String.prototype.ezZeroPadLeading(num)” function takes one argument which is the number of leading zeros that are to be padded to the beginning of the string to which this message is sent.

+++

[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.onscroll²³

The “window.onscroll” event handler performs the function of handling the “onscroll” event for the “window” object by firing the “ezWindowOnscrollCallback(scrollTop,

²² This function was added in version 0.91.

²³ A known bug was fixed in version 0.91.

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.

[Top of the Document](#)

The const_div_floating_debug_menu variable²⁴

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.

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

²⁴ This feature was added to the version 0.92 release.

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.

[Top of the Document](#)

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.cfm²⁵, index.cfm, cfinclude_index_body.cfm²⁶, 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](#)

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.

²⁵ **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.

²⁶ **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.

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,options Struct)²⁷

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);
```

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".

²⁷ Optional parameter "**optionsStruct**" was added to version 0.92

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)²⁸

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 web space as the file 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) AJAX made Easy !

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.

²⁸ This function was added to version 0.92.

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-numeric 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.

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.

***ezJSONencode(arg)*²⁹**

The "ezJSONencode(arg)" function takes one required argument that is a ColdFusion Object that will be converted into a JSON data block suitable for passing from ColdFusion to the client to be consumed by JavaScript.

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.

²⁹ This function was added with version 0.94.

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.

[Top of the Document](#)

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).**".

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.

It may be more efficient to use ColdFusion to manipulate XML and then use JSON and AJAX to transmit the data to the client; reverse the process to get XML from the client back into the server and on to the place the XML needs to go.

[Top of the Document](#)

A Word about JSON

ezAJAX Version 0.94 and later use JSON when transmitting Query Objects from the Server to the client.

ezAJAX Version 0.94 and later use JSON when interfacing with Web 2.0 API's that return JSON data blocks.

The ezAJAX client automatically converts all JSON data blocks into ezAJAX Query Objects that allow Query of Queries to be processed.

While JSON is an efficient way to transmit data from a server to a client or vice-versa JSON suffers from the inability to consume data from a JSON data block in an abstract manner that is independent from causing possible JavaScript errors whenever an expected data element may not exist. JSON data blocks do not lend themselves to automated Queries however Query Objects resolve these issues rather nicely.

[Top of the Document](#)



A Word about the DoJo Toolkit³⁰

The DoJo Toolkit was easily integrated with ezAJAX with minimal code required to make ezAJAX compatible with DoJo.

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.

³⁰ 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 has more features.

ezAJAX Enterprise Edition will provide an Object-Oriented Networked Data Model code named "Geonosis". Geonosis will make it possible to manipulate persistent data using a Relational Database such as SQL Server (any version) or Oracle (any version) or mySQL (any version) without having to code SQL Statements.

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.

A Word about using ezAJAX to Code Business Apps

Yes, we do have plans for releasing a Business Graphing API for the ezAJAX Enterprise Edition that leverages the power of the JavaScript Graphical API to make Business Charts and graphs come to life using JavaScript.

We envision ezAJAX Enterprise Edition becoming a very nice platform for Interactive Business Reports 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 business apps that have the same level of business graphics support that may be derived from Flash.

ezAJAX and PHP

We do have plans for publishing a PHP Connector for ezAJAX that not only allows PHP Programmers to use EAJAX 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. This can be easily done by constructing an intelligent Proxy using PHP that is able to serialize usage of the ColdFusion MX 7 Developers Edition that only needs to handle connections from a single IP address.

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 EAJAX 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. This can be easily done by constructing an intelligent Proxy using PHP that is able to serialize usage of the ColdFusion MX 7 Developers Edition that only needs to handle connections from a single IP address.

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 EAJAX 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. This can be easily done by constructing an intelligent Proxy using PHP that is able to serialize usage of the ColdFusion MX 7 Developers Edition that only needs to handle connections from a single IP address.

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. This can be easily done by constructing an intelligent Proxy using PHP that is able to serialize usage of the ColdFusion MX 7 Developers Edition that only needs to handle connections from a single IP address.

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.